AFRL-RI-RS-TR-2022-135

SUPER TURING EVOLVING LIFELONG LEARNING ARCHITECTURE (STELLAR)

HRL LABORATORIES, LLC.

SEPTEMBER 2022

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2022-135 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
THOMAS E. RENZ
Work Unit Manager

/ S /
GREGORY J. HADYNSKI
Assistant Technical Advisor
Computing & Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
SEPTEMBER 2022	FINAL TECHNICAL REPORT	JUNE 2018	MAY 2022

4. TITLE AND SUBTITLE

Super Turing Evolving Lifelong Learning ARchitecture (STELLAR)

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
FA8750-18-C-0103	N/A	61101E
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
		R2GU

6. AUTHOR(S)

Praveen K. Pilly, Ph.D.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

HRL Laboratories, LLC
3011 Malibu Canyon Road
Malibu CA 90265-4797

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Air Force Research Laboratory/RITB 525 Brooks Road Rome NY 13441-4505	AFRL/RI	AFRL-RI-RS-TR-2022-135

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The overarching goal of HRL's Super Turing Lifelong Learning ARchitecture (STELLAR) project was to build a general-purpose scalable autonomous system that continually improves its performance during deployment, and rapidly and safely adapts to new tasks and circumstances, without catastrophic forgetting or the complete undoing of previously learned tasks. Human brains are the best available examples of lifelong learners. Leveraging HRL Team's long-standing expertise in neuroscience, cognitive architectures, machine learning, neuroevolution, and robotics, STELLAR integrates several brain-inspired mechanisms that operate across multiple spatial and temporal scales. These include memory consolidation or stabilization of memories during sleep, neuromodulation, which regulates neural activities and synaptic plasticity, adult neurogenesis or the birth of neurons, instincts, and reflexes. During Phase 1, we developed nine innovative components that solve different challenges and requirements of lifelong learning. During Phase 2, we integrated these components into a complete lifelong learning system and performed rigorous testing and evaluation. In particular, we demonstrated the efficacy of the fully integrated STELLAR system for the autonomous driving domain in the CARLA simulator with online adaptation to different weather conditions, different vehicle models (wheel asymmetries, vehicle dynamics), and different driving tasks (driving in the correct lane vs. opposite lane in regular traffic). The STELLAR system achieved significant performance improvements relative to conventional machine learning Single Task Experts, exceeding key program targets: about 25% performance improvement, about 7.5x increase in learning efficiency, and about 3.5x improvement in Forward Transfer (leveraging prior learning to facilitate learning a new task).

15. SUBJECT TERMS

Continual Machine Learning, Reflexive Machine Adaptation, Super Turing Lifelong Learning ARchitecture, Forward and Backward Transfer Machine Learning, Neurogenesis for Machine Learning, Plastic Neuromodulated Network, Neuromodulated Attention

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
U	U	U	SAR	253

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
THOMAS E. RENZ	N/A

Table of Contents

List of Figures . . . iii
List of Tables . . . vii
1.. Phase 1: Lifelong Learning Components . . . 1
1.1. Summary . . . 1
1.2. Introduction . . . 3
1.2.1. MOdulated Hebbian Network . . . 3
1.2.2. Sliced Cramer Preservation . . . 8
1.2.3. Neuromodulated Attention . . . 14
1.2.4. Self-Preserving World Model . . . 15
1.2.5. Context-Skill Model . . . 18
1.2.6. Reflexive Adaptation . . . 20
1.2.7. Probabilistic Program Neurogenesis . . . 20
1.2.8. Meta-Learned Instinct Network . . . 21
1.2.9. Plastic Neuromodulated Network . . . 24
1.3. Methods, Assumptions, and Procedures . . . 26
1.3.1. MOdulated Hebbian Network . . . 26
1.3.2. Sliced Cramer Preservation . . . 28
1.3.3. Neuromodulated Attention . . . 32
1.3.4. Self-Preserving World Model . . . 42
1.3.5. Context-Skill Model . . . 48
1.3.6. Reflexive Adaptation . . . 51
1.3.7. Probabilistic Program Neurogenesis . . . 82
1.3.8. Meta-Learned Instinct Network . . . 88
1.3.9. Plastic Neuromodulated Network . . . 94
1.4. Results and Discussion . . . 97
1.4.1. MOdulated Hebbian Network . . . 97
1.4.2. Sliced Cramer Preservation . . . 104
1.4.3. Neuromodulated Attention . . . 110
1.4.4. Self-Preserving World Model . . . 120
1.4.5. Context-Skill Model . . . 126
1.4.6. Reflexive Adaptation . . . 131
1.4.7. Probabilistic Program Neurogenesis . . . 142
1.4.8. Meta-Learned Instinct Network . . . 144
1.4.9. Plastic Neuromodulated Network . . . 149
1.5. Conclusions . . . 157
1.5.1. MOdulated Hebbian Network . . . 157
1.5.2. Sliced Cramer Preservation . . . 157
1.5.3. Neuromodulated Attention . . . 157
1.5.4. Self-Preserving World Model . . . 158
1.5.5. Context-Skill Model . . . 159
1.5.6. Reflexive Adaptation . . . 160
1.5.7. Probabilistic Program Neurogenesis . . . 162
1.5.8. Meta-Learned Instinct Network . . . 162

1.5.9. Plastic Neuromodulated Network 163
2.. Phase 2: Lifelong Learning System Integration 165
2.1. Summary 165
2.2. Introduction 165
2.3. Methods, Assumptions, and Procedures 168
2.3.1. Month 9 Evaluation 168
2.3.2. Month 12 Evaluation 171
2.3.3. Month 15 Evaluation 179
2.3.4. Month 18 Evaluation 188
2.3.5. Month 21 Evaluation 189
2.4. Results and Discussion 202
2.4.1. Month 9 Evaluation 202
2.4.2. Month 12 Evaluation 203
2.4.3. Month 15 Evaluation 207
2.4.4. Month 18 Evaluation 209
2.4.5. Month 21 Evaluation 212
2.5. Conclusions 215
3.. References 217
4.. Publications and Presentations 234
5.. Software Packages 238
6.. Intellectual Property 239
7.. List of Symbols, Abbreviations, and Acronyms 240

List of Figures

1. STELLAR components contribute to different aspects of L2 during deployment. . . 3
2. Comparative performance of MOHQA on (a) CT-graph and (b) Malmo benchmarks. 6
3. The CT-graph problem. 7
4. Bird's eye view of the maze used in the MOHQA experiments. 7
5. Depiction of how sample-based and distribution-based approaches regularize the learning. 9
6. The energy landscape of various distances as a function of the translation parameter. 13
7. Concept diagram illustrating the World Model framework and the sequential training scheme. 17
8. Graphical representation of MOHQA. 26
9. Network setup for our bottom-up and top-down processes. 32
10. Two example test pairs of noisy MNIST digits and their c-EB highlighted results. . 35
11. Two more example test pairs of noisy MNIST digits and their c-EB highlighted results. 37
12. The neuromodulated procedure for the noisy MNIST-pair experiment. 37
13. The neuromodulated procedure for the indoor robot experiment. 39
14. Depiction of the top-down attentional search process for a guessed action. 40
15. The test scenario for the indoor robot experiment. 41
16. Depiction of lifelong learning facilitated by the World Model. 42
17. Schematic of the full network comprising VAE and the World Model. 43
18. Scenes from Flappy Ball (a), Lunar Lander (b), and CARLA (c) domains. 48
19. The architecture of the Context-Skill Model and its ablations. 49
20. Algorithm for training the Context-Skill Model. 52
21. Overall workflow of the reflex module. 53
22. Typical output of CVT-MAP-Elites on the 10-dimensional Rastrigin function. . . . 54
23. False-negative results for detecting death events. 55
24. False-positive results for detecting death events. 56
25. Successful death-event detections using the MSE metric on latent representations. . 57
26. False-positive results for the MSE metric used on latent representations. 57
27. Schematic representation of the agent model. 58
28. Comparison of (a) true and (b) predicted observations with a well-trained memory model. 59
29. Comparison of (a) true and (b) predicted observations with the current memory model. 60
30. Impact of reflexes on cumulative reward for a random agent. 62
31. Impact of reflexes on cumulative reward for a trained agent. 62
32. Impact of reflexes on the number of deaths for a random agent. 62
33. Impact of reflexes on the number of deaths for a trained agent. 63
34. Prediction performance of the baseline World Model implementation. 64
35. Model performance under BCE weights modification. 65
36. Model performance under terminal value adjustment approach. 66
37. Model performance with soft precision. 67
38. Multi-step future death-event prediction accuracy of the modified World Model. . . 68
39. Multi-step future death-event prediction recall of the modified World Model. . . . 68

40. Multi-step future death-event prediction precision of the modified World Model. . . 69
41. Reflex retrieval of the fast nearest-neighbor search approach for the Space Invaders Atari game. . . . 71
42. Reflex retrieval difference for two different latent vector difference thresholds. . . . 72
43. Schematic of the linear classifier network used to trigger reflexes. . . . 74
44. Training loss for the CNN-based VAE. . . . 74
45. Training loss for the LC network. . . . 75
46. Average absolute error per predicted output. . . . 75
47. Illustration of the MAP-Elites algorithm. . . . 77
48. The reflex module monitors and takes over the controller's actions as required. . . . 79
49. Overall workflow of the reflexive adaptation procedure. . . . 80
50. Workflow of the BO search procedure used during deployment. . . . 81
51. Illustration of neurogenesis used in our approach. . . . 83
52. Our EDA used to execute neurogenesis. . . . 84
53. Illustration of the probabilistic program learner used in our approach. . . . 84
54. Illustration of our experimental setup for neurogenesis. . . . 86
55. Comparison of the average best fitness found over the first 5 minutes of optimization time. . . . 87
56. Comparison of the average best fitness found over the next 5 to 21 minutes of optimization time. . . . 87
57. Comparison of our approach with the GA. . . . 88
58. The topology of the policy network with the instinct module. . . . 90
59. Pre-training and training procedures for IR^{2}L and two baselines. . . . 90
60. A sequence of frames showing the pre-trained policy going over a hazard in the "goal" task. . . . 92
61. Coordination of the policy and instinct networks to avoid hazards in the "goal" task. 92
62. Depiction of the "goal" task in the Safety Gym environment. . . . 92
63. Depiction of "buttons" task in the Safety Gym environment. . . . 93
64. Depiction of "push" task in the Safety Gym environment. . . . 93
65. Overview of the proposed computational framework. . . . 96
66. Decision made by various algorithms at different episodes. . . . 98
67. Decisions made by various algorithms at different episodes in a problem with noise. 99
68. Features learned by DQN without and with MOHQA. . . . 100
69. Performance on the simpler versions of the CT-graph. . . . 102
70. Performance on a more complex CT-graph. . . . 103
71. Comparison of results in Malmo benchmark. . . . 103
72. Comparison among various selective plasticity algorithms. . . . 105
73. Comparison between MAS and SCP on sequential learning of auto-encoders. . . . 106
74. Comparison among various algorithms on semantic segmentation. . . . 106
75. Testing Dice score of various selective plasticity algorithms. . . . 108
76. Comparison among various selective plasticity algorithms in the MNIST-to-SVHN experiment. . . . 109
77. The model used in the MNIST-to-SVHN experiment. . . . 109
78. Comparison of various selective plasticity algorithms in the CIFAR-100 experiments. 110
79. Visualization of goal-driven perception for different levels of the major goal validity. 112

80. Visualization of goal-driven perception with the randomly switching major goal validity. . . . 113
81. Visualization of goal-driven perception with the randomly switching major goal validity after NE and/or ACh ablation. . . . 115
82. Visualization of goal-driven perception with the randomly switching major goal validity for neuromodulated WTA. . . . 117
83. Visualization of goal-driven perception with the randomly switching major goal validity for the "random-or-fixed" benchmark. . . . 117
84. Visualization of action-based goal-driven perception in the robot experiment. . . . 118
85. M network loss over task exposures. . . . 121
86. Reconstruction of test rollouts from River Raid Atari game using M networks after training in each task exposure. . . . 121
87. Cumulative reward in C network over task exposures. . . . 122
88. Results from Supplementary Experiment 1. . . . 123
89. Results from Supplementary Experiment 2. . . . 125
90. Percent of integrated loss plotted for each task. . . . 125
91. Comparing generalization of CS and its ablations C and S in the FB domain. . . . 128
92. Comparing generalization of CS and its ablations C and S in the LL domain. . . . 128
93. Comparing generalization of CS and its ablations C and S in the CARLA domain. . . 128
94. Contrasting the generalization ability of (a) CS, (b) C, and (c) S networks. . . . 130
95. Schematic representation of the planar kinematic arm problem. . . . 132
96. Test results on kinematic arm with variable morphology. . . . 133
97. Example 6-legged robot morphologies. . . . 134
98. Test results on the 6-legged locomotion problem scenario. . . . 135
99. Sample screenshot from (a) Space Invaders and (b) River Raid OpenAI Gym Atari games. . . . 136
100. Test results for a 20-timestep reflex length. . . . 137
101. Test results for a 100-timestep reflex length. . . . 138
102. Test results using the 5-image emergency situation data directly as a task-descriptor. 139
103. Reflex module performance results on the Space Invaders Atari game. . . . 140
104. Reflex module performance results on the River Raid Atari game. . . . 141
105. Reward and collisions during training in the "buttons" task. . . . 144
106. Cumulative reward for the final baselines and IR^2L on the "buttons" task. . . . 145
107. Cumulative collisions on training runs for the final baseline and IR^2L on the "buttons" task. . . . 145
108. Rewards and collisions during training on the "buttons" task. . . . 146
109. Cumulative reward for the final baseline and IR^{2}L on the "push" task. . . . 147
110. Cumulative collisions on training runs for the final baseline and IR^{2}L on the "push" task. . . . 148
111. Agent trajectory on the "push" task. . . . 148
112. Adaptation performance across tasks of SPN and NPN in CAVIA framework. . . . 149
113. Adaptation performance across tasks of SPN and NPN in PEARL framework. . . . 150
114. Adaptation performance, based on success rate metric, across tasks of SPN and NPN in CAVIA and PEARL. . . . 150
115. Adaptation performance across tasks of SPN and NPN for discrete control in CAVIA. 152

116. Representation similarities between tasks in the 2D navigation environment. 153
117. Representation similarities between tasks in the CT-graph depth2 environment. . . 154
118. Representation similarities of neuromodulatory activities h^m between tasks in the 2D navigation environment. 155
119. Representation similarities of neuromodulatory activities h^m between tasks in the CT-graph depth2 environment. 155
120. Representation similarities of neuromodulatory activities h^m between tasks in the Meta-World ML45 environment. 156
121. Control experiments: Adaptation performance of NPN and variants of SPN. 156
122. Illustration of how a new policy can be safely adopted using reflexes. 161
123. A possible simulated driving scenario for demonstrating reflexes in autonomous driving. 162
124. Summary of the innovative components developed in Phase 1. 166
125. Static architecture of the fully integrated STELLAR system for L2 in autonomous systems. 166
126. Non-smooth changes in successive states for the encoded version of the MiniGrid domain. 168
127. Architecture of the Baseline Model. 172
128. System architecture of Month 9 system with two L2 components (highlighted in yellow). 172
129. System architecture of Pre-Month-12 system with six L2 components (highlighted in yellow). 173
130. System architecture of Pre-Month-15 system with 10 L2 components (highlighted in green and yellow). 173
131. System architecture of the Month 15 system. 180
132. System architecture of the Month 21 system. 189
133. Normalized smoothed performance curves from an illustrative lifetime for our Month 9 system. 202
134. Post-hoc pairwise comparison of the Month 9 system relative to the Baseline Model. 203
135. Post-hoc pairwise comparison of the Pre-Month-12 system relative to the Baseline Model. 204
136. Normalized smoothed performance curves from an illustrative Permuted Scenario lifetime for our Month 12 system. 205
137. Normalized smoothed performance curves from an illustrative Alternating Scenario lifetime for our Month 12 system. 205
138. Normalized smoothed performance curves from an illustrative Condensed Scenario lifetime for our Month 15 system. 207
139. Normalized smoothed performance curves from an illustrative Dispersed Scenario lifetime for our Month 15 system. 208
140. Baseline Model suffers from stability-plasticity dilemma unlike the STELLAR system. 213
141. Histograms of matched differences in Performance Maintenance between the STELLAR system and the Baseline Model. 213

List of Tables

1. Phase 1 goals and accomplishment status. . . . 2
2. Relationship between goal actions and object labels. . . . 39
3. Parameter ranges during training of the Context-Skill Model. . . . 51
4. Results summary of the nearest-neighbor search approach. . . . 70
5. Computational cost of DQN. . . . 101
6. Computational cost of MOHQA. . . . 101
7. Prediction for 10,000 test pairs of noisy MNIST digits. . . . 110
8. Average goal-driven perception performance on noisy MNIST pairs for various goal validity settings. . . . 113
9. Average goal-driven perception performance on noisy MNIST pairs for various ablations. . . . 114
10. Average goal-driven perception performance on noisy MNIST pairs among various benchmarks. . . . 116
11. Parameter ranges for evaluating generalization of the Context-Skill Model. . . . 127
12. Change in Context and Skill modules during generalization. . . . 130
13. Differences across the five task variants in the Month 9 Evaluation Protocol. . . . 169
14. Common task elements across the five task variants in the Month 9 Evaluation Protocol. 169
15. Task similarity matrix for the Month 9 evaluation. . . . 170
16. Description of changes from Month 15 to Month 18 evaluation. . . . 188
17. Summary of Month 9 evaluation results. . . . 203
18. Summary of Month 12 evaluation results for the Month 12 system with eight L2 components. . . . 206
19. Summary of Month 12 evaluation results for the Pre-Month-15 system with 10 L2 components. . . . 206
20. Summary of Month 9 evaluation results for various versions of the STELLAR system. 206
21. Summary of evaluation on the Month 12 Permuted Scenario for the Month 12 and Month 15 systems. . . . 209
22. Summary of evaluation on the Month 12 Alternating Scenario for the Month 12 and Month 15 systems. . . . 209
23. Summary of evaluation on the Month 15 Condensed and Dispersed Scenarios for the Month 15 system. . . . 209
24. Summary of evaluation on the Month 18 Condensed Scenario for the Month 15 and Month 18 systems. . . . 210
25. Summary of the effects of SCP++ ablation on the Month 18 Condensed Scenario for the Month 18 system. . . . 210
26. Summary of the effects of the number of replays on the Month 18 Condensed Scenario for the Month 18 system with SCP++ ablated. . . . 211
27. Summary of the effects of PNN ablation on the Month 18 Condensed Scenario for the Month 18 system. . . . 211
28. Summary of the effects of reducing SCP++ stiffness on the Month 15 Dispersed Scenario for the Month 18 system. . . . 212
29. Summary of evaluation results on the Month 21 Condensed Scenario. . . . 212

30. Summary of the effects of adding CSM and PNN to the Month 21 system for the Month 21 Condensed Scenario. 214
31. Summary of the effects of ablating SCP++ in the Month 21 system for the Month 21 Condensed Scenario. 214
32. Summary of evaluation results on the Month 21 Dispersed Scenario for the Month 21 system and the Baseline Model. 214
33. Comparison of the performance of the Month 21 system on the Month 21 Condensed and Dispersed Scenarios. 215

1. Phase 1: Lifelong Learning Components

The overarching goal of HRL's Super Turing Lifelong Learning ARchitecture (STELLAR) project under the DARPA Lifelong Learning Machines (L2M) program was to build a general-purpose scalable autonomous system that continually improves its performance during deployment, and rapidly and safely adapts to an increasingly large number of new tasks and circumstances encountered in sequence through the system's long lifespan, without catastrophic forgetting or the complete undoing of previously learned tasks. STELLAR is applicable to a wide range of autonomous system applications, including autonomous ground vehicles (both on-road and off-road), autonomous undersea vehicles, as well as autonomous aircraft, among others.

Autonomous systems based on conventional machine learning (ML) models have distinct training and deployment phases, with no further learning possible once training is complete. But there is no way to anticipate all potential situations that a system would encounter during deployment. Lifelong learning (L2) is possible for the conventional autonomous systems if all previous experiences are explicitly stored in memory and the system is re-trained on all old and new data whenever a new task or variant is encountered during deployment. But this is not scalable as memory and computing requirements keep growing with an ever-increasing number of new situations. Further, the longer it takes to re-train the system, the more time it is not engaged in the mission. The current concept of operations is not sustainable and scalable without automated in-situ continual learning directly on the platform.

1.1 Summary

To overcome L2 challenges faced by autonomous systems, we proposed to develop a neuromodulated evolving plastic neural network (E-PNN) capable of continual learning, rapid seamless adaptation to new tasks and circumstances, flexible action selection, and rapid reflexive adaptation to unsafe and/or surprising events. And to augment E-PNN with multi-scale memory networks that simulate sleep "replays" for offline (re)consolidation of rapidly acquired information to long-term memory, as well as online "preplays" to inform decision-making and provide early warning signs of potential crashes or other unsafe events that may require potential human intervention. These systems-level processes were to operate in concert with local consolidation of selective synapses by changing plasticity hyperparameters during online learning to eliminate catastrophic forgetting.

Consistent with these goals, we researched and developed several innovative component subsystems that solve different challenges and requirements of L2 (Table 1). In particular, Figure 1 illustrates how these various components can help to eliminate catastrophic drops in performance when new tasks are encountered during deployment, to transfer previous knowledge to new tasks and rapidly learn them, to consolidate the acquired performances for subsequent maintenance, to continually improve previous knowledge, as well as to accommodate a potentially infinite number of new tasks through the long lifespan of autonomous systems. Below we summarize our main scientific accomplishments from Phase 1 in the context of the previous State-of-the-Art (SotA):

- **MOdulated Hebbian Network**: Developed the first integrated reinforcement learning (RL) algorithm combining bio-inspired associative continual learning and backpropagation to

Table 1: Phase 1 goals and accomplishment status.

	Goal	Accomplishment
1	Bottom-up neuromodulation to facilitate rapid learning, as well as synaptic consolidation for performance maintenance of previously acquired tasks	MOdulated Hebbian Network Sliced Cramer Preservation
2	Top-down neuromodulation to rapidly select relevant information to guide robust behavior in face of noisy and unreliable signals	Neuromodulated Attention
3	Brain-inspired multi-scale memory for continual adjustment to previously learned tasks, while incorporating new knowledge	Self-Preserving World Model
4	Rapid seamless adaptation for improved transfer to new tasks	Context-Skill Model
5	Plastic neuromodulated networks for adaptive, flexible responses to new tasks that build off previous knowledge	Plastic Neuromodulated Network
6	Rapid reflexive adaptation for predictive "early warning" signals to avoid dangerous situations and quickly deal with them	Reflexive Adaptation Meta-Learned Instinct Network
7	Epigenetic neurogenesis for on-the-fly increase in network capacity to learn an increasingly large number of new tasks	Probabilistic Program Neurogenesis

rapidly discover high-level associations from cues/causes to effects and rewards [1] to achieve about 2x improvement in very sparse reward partially observable environments (1 reward/290K steps) over several deep RL algorithms

- **Sliced Cramer Preservation**: Achieved about 25% less forgetting of old tasks (overcoming catastrophic forgetting) and about 30% improved performance for new tasks (avoiding intransigence) compared to Elastic Weight Consolidation (EWC) and Memory Aware Synapses (MAS) methods with a distribution-based selective plasticity mechanism that allows for preserving the learned deep representations of a network at arbitrary layers [2]

- **Neuromodulated Attention**: Modeled top-down attention based on cholinergic and noradrenergic systems in the brain to track expected and unexpected uncertainties of tasks, respectively, to recognize a task change (less than 10 samples) and find the new task (within 20-30 samples), achieving about 10x reduction in time to consistently find new tasks compared to 'random or fixed' benchmarks [3]

- **Self-Preserving World Model**: Leveraged self-generated simulated rollouts to not only preserve the agent performance across task variations but also preserve its own temporal prediction loss, and thereby achieved about 2x more cumulative reward when evaluating on old tasks and about 1.5x more cumulative reward when evaluating on new tasks compared to using no generative replays [4]

- **Context-Skill Model**: Learning separate hidden representations at two temporal scales [5] achieves about 30% more forward transfer to new task variants for both interpolation and extrapolation compared to baselines (fully-connected feedforward and context-only recurrent neural networks)

- **Reflexive Adaptation**: Developed Task-Agnostic Reflexive Adaptation that detects dangerous situations using uncertainty within predictions and deploys quick action sequences to minimize such uncertainty (reflexes) to bring the system back to a safer state [6] to achieve about 70% average success-rate in preventing collisions in surprise obstacle scenarios in the CARLA domain, without any prior obstacle-avoidance training by the agent (compared to

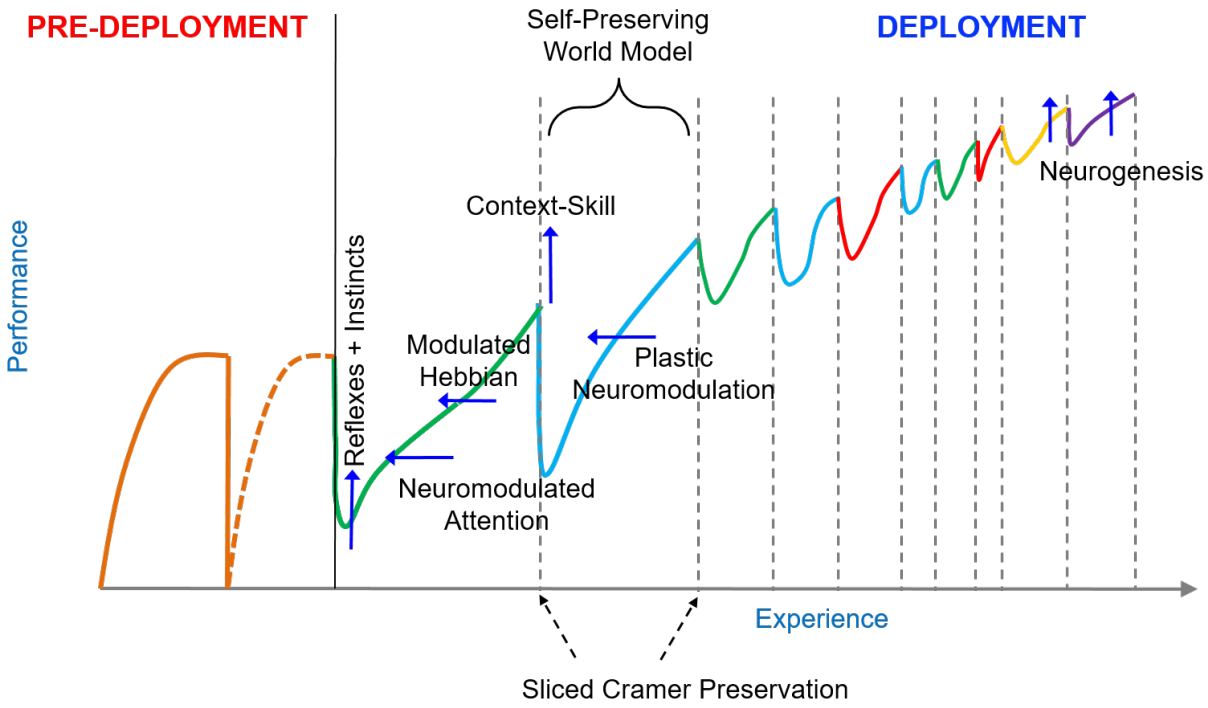


Figure 1: STELLAR components contribute to different aspects of L2 during deployment.

25% by random-response agent and 0% by Proximal Policy Optimization (PPO) agent trained in obstacle-free environment)

- **Probabilistic Program Neurogenesis**: Developed an unique evolutionary process that uses probabilistic program induction to efficiently identify where and how many new neurons to add when performance gains on a new task stagnate due to the system's limited learning capacity [7] to overcome catastrophic forgetting while achieving high performance on all tasks compared to EWC with the network size increased by less than 2%
- **Meta-Learned Instinct Network**: Developed the framework to learn meta-learned instincts during pre-deployment training to ensure safe exploration during adaptation to new RL tasks to achieve about 10K times fewer safety violations in the Safety Gym domain compared to PPO agent while maintaining the ability to learn a new task [8]
- **Plastic Neuromodulated Network**: Developed a layer-wise bio-inspired neuromodulatory mechanism to regulate the sign of neural activities to achieve 3-7x faster adaptation to new tasks in various domains compared to SotA meta-RL algorithms without neuromodulation [9]

1.2 Introduction

1.2.1 MOdulated Hebbian Network

We investigated rapid adaptation in environments with confounding observations and sparse rewards, which are a subclass of Partially Observable Markov decision processes (POMDPs). These environ-

ments, which we termed confounding POMDPs, are challenging for RL algorithms that depend on the Temporal Difference (TD) error. So we developed a new bio-inspired neural architecture that combines a MOdulated Hebbian Network (MOHN) with Deep Q-Network (DQN), which we call MOdulated Hebbian plus Q-network Architecture (MOHQA). The key idea is to use a Hebbian network with rarely correlated bio-inspired neural traces to bridge temporal delays between actions and rewards when confounding observations and sparse rewards result in inaccurate TD errors. In MOHQA, DQN learns low-level features and control, while MOHN contributes to the high-level decisions by associating rewards with past states and actions. Our new architecture, thus, combines two modules with significantly different learning algorithms, a Hebbian associative network and a classical DQN pipeline, exploiting the advantages of both.

Note that not all POMDPs are confounding, for example, using a single frame from Atari Breakout as a state is a POMDP as the agent has no information of the ball's direction and velocity. Non-confounding POMDPs can be solved with a memory system. In Atari Breakout one can stack multiple frames to derive a Markov decision process (MDP) state [10]. However, in confounding POMDPs even memory-based approaches fail [11] because the histories of observations do not repeat (due to observations occurring at random locations through the POMDP), thus the underlying state cannot be easily inferred from observations. The sparsity of the reward implies that many such uninformative observations occur in between rewards. The result is that TD errors cannot be computed if the state cannot be derived either from the present observation or a history, or alternatively, they become quickly inaccurate due to sparse rewards. Confounding observations and sparse rewards are common in many scenarios; for example, when driving across a town from point A to B, cars parked to the side of the road are not useful landmarks, whereas junctions or buildings are. Cars parked on the side of the road create confounding observations as they occur randomly in the problem and are irrelevant to the task of getting from A to B. Also, in this example, the reward might be rarely provided, i.e., not at each correct turn, but only when the destination is reached.

One way to apply RL to POMDP is to try to derive a state from the history of observations, and thus LSTM has been employed in [12, 13, 14, 15]. Another notable LSTM-based approach was shown in [16], where action and states are combined and inserted into the network as a single input resulting in more observable states. LSTMs were even used with hierarchical RL to solve some types of POMDPs [17]. One of the more recent approaches called AMRL [18] combines LSTM-based memory with a latent space averaging over time to boost the signal to noise ratio in the latent space to solve POMDP problems. To deal with sparse rewards (in a non-POMDP setting), [19] proposed the use of "recurrent neural networks (RNNs) with concrete", which remember states for longer by using multiple RNNs connected in series. [20] used differential neural computing to solve POMDPs requiring memory. Other memory types, which do not rely on neural networks, were also developed. For example, in [21] the agent stores a certain number of past states and retrieves relevant states when necessary. [22] introduced continuous memory states to better deal with problems in the continuous domain. Such memory-based approaches, while effective in many cases, suffer in cases in which observations derive from a large set, and thus histories do not repeat. While the training phase for an LSTM could provide the network with the ability to discard confounding observations, in practice this is often hard to achieve.

There exist other methods that do not rely on memory; for example, [23] introduced option-

observation initiation sets to work with options from [24] to make them suitable for solving POMDPs. [25] proposes two additions to the well-known Probabilistic Inference for Learning Control algorithm [26]: (i) direct training and (ii) filtering to deal with a noisy state space. Generative models were used to update beliefs about the environment in [27]. These approaches do not suffer from non-repeating histories, but require dense rewards.

To address the challenges of solving confounding POMDPs, our architecture is composed of two parts: a standard DQN and a layer of reward-modulated Hebbian network [28, 29] with neural eligibility traces (MOHN), parallel to the Q-network head that contributes to decision-making. The combination of two learning modules in one architecture differs from Backpropamine [30, 31]. In Backpropamine, Hebbian learning and backpropagation are used on the same connection, where backpropagation determines the utilization of the Hebbian component. In Backpropamine, the Hebbian component does not result in a permanent weight increase and is nullified at the end of each episode. In essence Hebbian connections are used for a within-episode memory, while backpropagation is used to learn scaling factors, thereby deciding when and how much of a Hebbian component to use within episodes. On the other hand, in MOHQA, Hebbian components result in a permanent weight increase. Also MOHQA utilizes sparse correlations. Moreover, Backpropamine has only been shown to work in a very limited setting in RL with a single-layer network and a one-dimensional state space.

In our architecture, DQN acts as a feature provider via convolutional layers to MOHN, and contributes to the final decision in equal measure as the parallel MOHN. The addition of MOHN helps DQN discriminate between pivotal decision points and confounding observations, thus gaining a learning advantage in confounding POMDP problems. MOHN uses two key mechanisms: (i) rarely correlated eligibility traces to associate a state-action pair with reward and (ii) Hebbian learning for rapid learning from few examples. The ability of the modulated Hebbian-like network with rare eligibility traces to bridge temporal gaps between events and rewards and rapid learning from few examples was demonstrated in a spiking neural model in [32], and an equivalent model for rate-based neurons was shown effective in simulations and robotics applications [29, 33, 34].

Eligibility traces, which is one of MOHQA's key mechanisms, is not new in RL [35, 36]. However, eligibility traces in MOHQA use forward view and implement a rare (or sparse) correlation mechanism that distinguishes MOHQA's traces from traditional RL ones, such as SARSA(λ), Q(λ), or Monte-Carlo methods (which are a special case of the eligibility traces approach when $\lambda = 1$) such as REINFORCE [37]. While forward view and backward view eligibility traces are largely equivalent, rare correlations give MOHQA two distinct advantages. Rare correlation means only a small percentage of all weights are considered for an update for a given state-action pair. This means that, compared to traditional RL, MOHQA (i) has the increased ability to perform well with sparse reward signals and (ii) has the ability to cope with noise in the feature space (a problem that has been recently discussed in great detail in [18]), by ignoring the noisy inputs and only focusing on significant input features.

The architecture is tested on a set of generalized reward-based decision problems that include confounding POMDPs. Tests include comparisons with a baseline DQN [10], QRDQN+LSTM [12], REINFORCE, A2C [38], and AMRL [18]. Our simulations show that MOHQA can solve

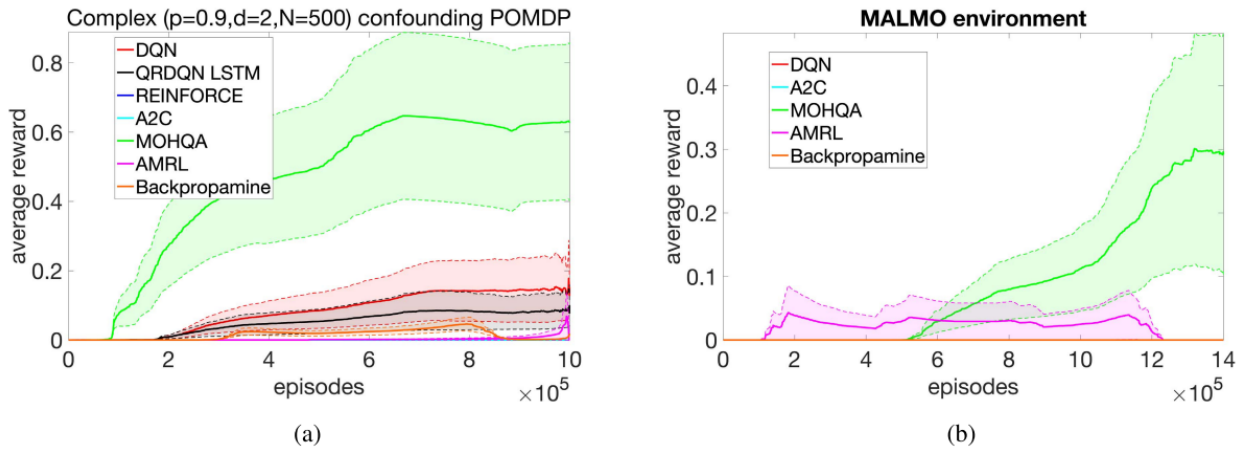


Figure 2: Comparative performance of MOHQA on (a) CT-graph and (b) Malmo benchmarks.

confounding POMDPs that DQN cannot, and is even able to outperform A2C, memory-based AMRL, and REINFORCE by at least 33% in more complex scenarios (Figure 2). To the best of our knowledge, MOHQA is the first application of a combined modulated Hebbian network and DQN to improve learning in the presence of partially observable Markov states.

Confounding POMDP Problems

Confounding POMDPs are represented as environments where leading-to-reward decision points occur occasionally and are separated by confounding wait states where a fixed policy is required; e.g., wait action. This is a fairly common case in robotics and games. In the driving analogy we made above, key decision points are at junctions, while wait states are those along straight segments where parked cars are confounding stimuli. Here we encode the challenges of confounding POMDPs within two benchmarks: (i) a graph-based benchmark with two-dimensional (2D) synthetic inputs we named Configurable Tree graph (CT-graph) and (ii) the well-known three-dimensional benchmark Malmo [21, 18].

CT-graph Environment

The CT-graph is an abstraction of the decision-making process that allows designing confounding POMDPs with quantifiable metrics for complexity and partial observability (see Figure 3 for pictographical representation). A CT-graph can be thought of as a tree-like decision graph with two types of nodes (i) *decision states* (where the CT-graph branches into multiple sub-trees) and (ii) *wait states* (where the tree does not branch). Actions are also of two types: *wait-actions* and *act-actions*. Figure 3(A) shows the agent-environment interaction is similar to the standard RL with the difference that the environment provides observations. Figure 3(B) provides a representation of the hidden Markov process: the wait-action a^0 is required at wait states, while act-actions a^1 and a^2 are required at decision points. The panel shows a unit (with one branching point) that can be repeated to obtain arbitrarily large problems. Figure 3(C) shows a graphical representation of the observations and decision for a CT-graph of depth 2 (i.e., two sequential branching points).

A reward is located at one particular leaf node in the tree graph. The agent is required to perform wait-actions while in a wait state. While at a decision point the agent chooses from a set of act-

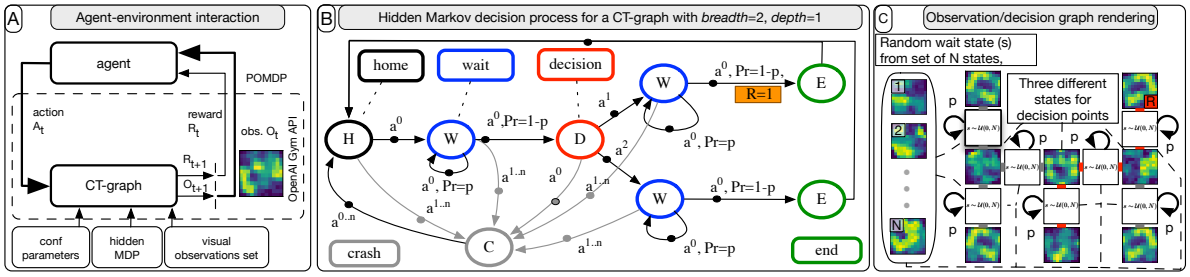


Figure 3: The CT-graph problem.

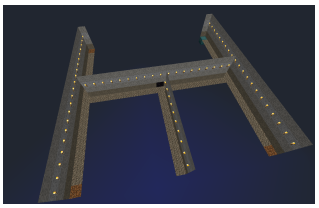


Figure 4: Bird's eye view of the maze used in the MOHQA experiments.

actions, where the specific act-action chosen determines the path that the agent follows along the tree. Wait states lead to themselves with a delay probability p or to the next state in the sequence with probability $1 - p$. At each decision point there are b options corresponding to the branches in the sub-trees of a particular decision point. The number of decision points between home and reward state is the depth d of the tree graph. The environment has an optimal sequence of actions that leads to one unique leaf of the tree graph that returns a reward of one. The branch factor b, the depth d, the delay probability p, and the sets of observations are configurable parameters, making this problem a blueprint for a large set of benchmarks, from simple to extremely difficult problems for medium to large size graphs (Figure 3).

CT-graph as a confounding POMDP problem. The specific configurations of the CT-graph that realize confounding POMDPs are where wait state observations are chosen randomly from a large set N and reward is sparse (the delay probability p is high). Such a setup represents a decision-making process in which confounding observations (wait states) are more common than decision points. With this property, we observed that RL agents cannot easily learn optimal policies as histories do not repeat and TD error estimates are inaccurate.

Malmo Environment

Malmo [39] is an environment based on the three-dimensional game Minecraft. The environment (Figure 4) is configured with a similar outline as the CT-graph, with junctions (similar to decision states) which are separated by long corridors (similar to wait for states). The reward is positioned at one leaf node (see a teal square in Figure 4). The agent receives no reward or punishment for every other step.

At each step, the agent makes a discrete move either left, right or forward. The forward action is equivalent to the wait-action of the CT-graph while left/right are decisions. To help with exploration speed, the backward action is disabled and a step forward after each turn action is performed.

Stepping forward after each turn means: (i) the agent can only take one turn in a decision point and (ii) if a left/right action is taken in a corridor, the episode is terminated. Those two simplifications stopped the agent from rotating on the spot during exploration.

Malmo as a confounding POMDP problem. The problem in Malmo is a confounding POMDP as the some corridor states look similar to each other regardless of the location in the maze, and therefore affect the ability to compute the TD error. For example, the corridor after the first turning point looks the same regardless of whether the agent chose the left or right action. While histories are easier to use here than in the CT-graph, it is still challenging to extract what is useful to remember due to the same observations appearing at different places in the problem space.

1.2.2 Sliced Cramer Preservation

When an autonomous system is exposed to a new task, it is desirable to continue to train the base computational model only on the new data/task and incrementally accumulate knowledge to improve the performance of the system over time, as opposed to retraining the model on the composition of old and new data. Apart from overcoming *catastrophic forgetting* [40] of previously acquired knowledge when learning new tasks as a result of interference between the old and new tasks, a successful lifelong learner should also overcome *intransigence*, which refers to the inability to acquire new knowledge while trying to preserve old knowledge (e.g., reducing the learning rate) [41].

To address this challenge, we focused on selective synaptic plasticity, which is the capability of a network to selectively change individual synaptic weights throughout the network. The standard deep neural network architectures are uniformly plastic; hence, all neurons are prone to changes during training, and this powerful capability is also the demise of these networks and leads to catastrophic forgetting. The idea of selective synaptic plasticity is to partially preserve synapses that are critical for previously learned tasks by rigidifying those synapses (i.e., to enforce critical synapses to change less). Rigidifying the network over time leads to a loss of learning capability for future tasks, which is known as 'intransigence' in the literature. Selective synaptic plasticity, by itself, could not fully overcome intransigence. A combination of strategies like efficient memory replay for reconsolidation, neurogenesis, and selective synaptic plasticity could lead to superior methods that defeat both catastrophic forgetting and intransigence. [41], for instance, provide such a combination of memory replay and selective synaptic plasticity. Also, there have been various efforts toward making the idea of neural resource allocation scalable, the Progress and Compress work [42], and the Incremental Moment Matching [43] work fall under this category.

Inspired by [41], we take a geometric view and devise a new method for selective synaptic plasticity. Our method is fundamentally different from the previous approaches like EWC [44] and MAS [45], which we indicate as sample-based approaches. Instead, we focus on identifying synaptic importance parameters that preserve the '*distribution*' of the latent representation of a task. Focusing on preserving the distribution of the latent representation of a neural network at an arbitrary layer, as opposed to the expected change in network's response for individual samples, enforces a less restrict regularization on the network, and enables a better utilization of the network learning capacity. The primary concept of sample-based versus distribution-based regularizations are visualized in

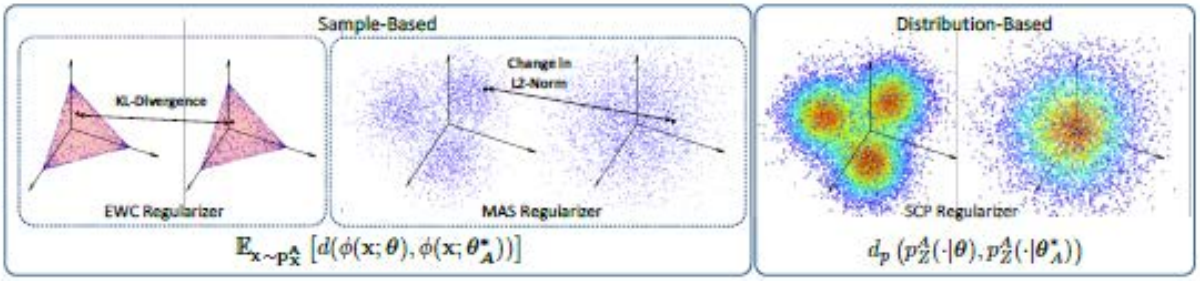


Figure 5: Depiction of how sample-based and distribution-based approaches regularize the learning.

Figure 5. Sample-based approaches regularize the learning by the expected change, as measured by a dissimilarity measure, of the response of the network for individual samples from Task A, after learning Task A and during learning Task B, $\mathbb{E}_{\boldsymbol{x}\sim p_X^A}\left[d(\boldsymbol{\phi}(\boldsymbol{x};\boldsymbol{\theta}),\boldsymbol{\phi}(\boldsymbol{x};\boldsymbol{\theta}_A^*))\right]$, where $d(\cdot,\cdot)$ is a dissimilarity measure between two K-dimensional vectors. Therefore, the regularization is an empirical expected change of the response for samples. EWC and MAS fall under the sample-based category. Our distribution-based approach, on the other hand, regularizes the change in the overall distribution of the network's output for input samples from Task A, after learning Task A and during learning Task B, $d_p\left(p_Z^A(\cdot|\boldsymbol{\theta}),p_Z^A(\cdot|\boldsymbol{\theta}_A^*)\right)$, where $p_Z^A(\cdot|\boldsymbol{\theta})$ is defined in Equation 8, and $d_p(\cdot,\cdot)$ is a distance measure between two probability distributions defined on $\mathscr{Z}\subseteq\mathbb{R}^K$. Similar to the MAS framework, our method is also able to preserve a task representation in any layer of a neural network, hence, enabling its application to various unsupervised or self-supervised learning settings.

Problem Set-up and Preliminaries

Consider data from a stream of tasks $\mathscr{X}^t=\{\boldsymbol{x}_i^t\sim p_X^t\}_{i=1}^{n_t}$, where p_X^t is the probability density function (PDF) for task t defined on $\mathbb{X}\subset\mathbb{R}^d$. We consider both supervised and unsupervised tasks, where in the supervised case the input sample, $\boldsymbol{x}_i^t$, is accompanied with the corresponding label $\boldsymbol{y}_i^t\in\mathbb{R}^K$. Let $\boldsymbol{\phi}(\cdot;\boldsymbol{\theta}):\mathbb{R}^d\to\mathbb{R}^K$ denote a parametric function (e.g., a neural network) that is to be optimized to solve the stream of tasks. In the supervised learning setting, we consider $\boldsymbol{\phi}$ to be the mapping to the logits prior to applying the softmax layer.

Geometric View of Elastic Weight Consolidation

In their seminal work, [44] considered the problem of overcoming catastrophic forgetting in supervised as well as RL scenarios where a supervisory signal $\boldsymbol{y}$ exists, whether in the form of labels/annotations, or environmental rewards, respectively. Here we reiterate the geometric interpretation of the EWC framework following the work of [41], which we will then adapt to define our generic consolidation framework. Let $p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})=\text{softmax}(\boldsymbol{\phi}(x;\boldsymbol{\theta}))$, where $[p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})]_j$ is the softmax probability of the j-th class. For simplicity, let us consider the case where we want to learn only two tasks consecutively, i.e., tasks 'A' and 'B.' Then, EWC ensures that while learning task 'B,' the conditional likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}^A)$ does not drift far from the optimal conditional likelihood $p_{\boldsymbol{\theta}_A^*}(\boldsymbol{y}|\boldsymbol{x}^A)$, where $\boldsymbol{\theta}_A^*$ are the parameters initially optimized for task A:

$$\begin{aligned}\arg\min_{\boldsymbol{\theta}} \tilde{\mathscr{L}}^B(\boldsymbol{\theta}) &= \arg\min_{\boldsymbol{\theta}} \mathscr{L}^B(\boldsymbol{\theta}) + \lambda \mathbb{E}_{\boldsymbol{x}\sim p_X^A}\left[D_{\mathrm{KL}}(p_{\boldsymbol{\theta}_A^*}(\boldsymbol{y}|\boldsymbol{x}) \,||\, p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}))\right] \\ &= \arg\min_{\boldsymbol{\theta}} \mathscr{L}^B(\boldsymbol{\theta}) + \lambda \mathbb{E}_{\boldsymbol{x}\sim p_X^A}\left[\mathbb{E}_{y\sim p_{\boldsymbol{\theta}_A^*}}\left[log(\frac{p_{\boldsymbol{\theta}_A^*}(y|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(y|\boldsymbol{x})})\right]\right] \end{aligned} \tag{1}$$

where λ is the regularization coefficient. Note that Equation 1 is essentially an optimization with a memory replay regularizer. Here, while learning task B, samples from task A (i.e., a memory buffer from this task) are fed through the network and the conditional likelihood is constantly checked against the optimal conditional likelihood for task A, $p_{\boldsymbol{\theta}_A^*}(y|\boldsymbol{x}^A)$, to ensure a minimal deviation from those parameters.

The key question answered by the EWC framework is on how to avoid memory replay and yet achieve a similar result, i.e., not forget the knowledge from old task (task A). The answer lies in the second-order Taylor expansion of the regularizer around the parameters optimized for the old task, $\boldsymbol{\theta}_A^*$. It is straightforward to show [41] that the second-order Taylor expansion of the regularizer is of the form:

$$\mathbb{E}_{\boldsymbol{x}\sim p_X^A}\left[D_{\mathrm{KL}}(p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) \,||\, p_{\boldsymbol{\theta}+\delta\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}))\right] \approx \frac{1}{2}\delta\boldsymbol{\theta}^T F_{\boldsymbol{\theta}}\delta\boldsymbol{\theta} = \|\delta\boldsymbol{\theta}\|^2_{F_{\boldsymbol{\theta}}} \tag{2}$$

where $F_{\boldsymbol{\theta}}$ is the Fisher Information Matrix (FIM) and is defined as:

$$F_{\boldsymbol{\theta}} = \mathbb{E}_{\boldsymbol{x}\sim p_X^A}\left[\mathbb{E}_{y\sim p_{\boldsymbol{\theta}_A^*}}\left[\left(\frac{\partial log(p_{\boldsymbol{\theta}}(y|\boldsymbol{x}))}{\partial\boldsymbol{\theta}}\right)\left(\frac{\partial log(p_{\boldsymbol{\theta}}(y|\boldsymbol{x}))}{\partial\boldsymbol{\theta}}\right)^T\right]\right] \tag{3}$$

Therefore, when $\delta\boldsymbol{\theta} \to 0$ the KL (Kullback-Leibler)-divergence regularizer enforces closeness of $\boldsymbol{\theta}$ to $\boldsymbol{\theta}_A^*$ in a Riemannian pseudo-manifold induced by the FIM. Given that the number of parameters could easily reach several million in standard deep neural networks, it is practically infeasible to store and use the FIM matrix, $F_{\boldsymbol{\theta}}$. Therefore, [44] assume that $F_{\boldsymbol{\theta}}$ is diagonal and further approximate the KL-divergence with:

$$\mathbb{E}_{\boldsymbol{x}\sim p_X^A}\left[D_{\mathrm{KL}}(p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) \,||\, p_{\boldsymbol{\theta}+\delta\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}))\right] \approx \frac{1}{2}\sum_{m=1}^{M}[F_{\boldsymbol{\theta}}]_{m,m}[\delta\boldsymbol{\theta}]^2_m \tag{4}$$

where M is the total number of parameters in the neural network. This leads to the main equation in the EWC framework:

$$\arg\min_{\boldsymbol{\theta}} \tilde{\mathscr{L}}^B(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \mathscr{L}^B(\boldsymbol{\theta}) + \frac{\lambda}{2}\sum_{m=1}^{M}\left[F_{\boldsymbol{\theta}_A^*}\right]_{m,m}[\boldsymbol{\theta}-\boldsymbol{\theta}_A^*]^2_m \tag{5}$$

From our point of view, the critical aspect of these derivations is the connection between memory replay and structural plasticity. Note that we started with Equation 1, which uses the idea of a memory replay regularizer. Then by assuming $\delta\boldsymbol{\theta} \to 0$, using the second-order Taylor expansion of the KL-divergence around $\boldsymbol{\theta}_A^*$, and assuming that FIM is a diagonal matrix we arrived at Equation 5, which provides the idea of synaptic importance parameters and is an embodiment of the selective

synaptic plasticity framework. Next, we use this critical aspect and develop an analogous regularizer (i.e., based on memory replay) for the MAS algorithm.

Generalizing to Unsupervised Learning

The EWC framework as explained above preserves the softmax probability of samples from Task A while learning Task B. This limits the applicability of the method to networks with outputs living on a K-dimensional simplex, e.g., supervised learning and RL where the network outputs a probability over a finite set of actions. More recently, [45] presented their MAS framework, which lifts the requirement for EWC outputs to live on a simplex, and enables calculation of the synaptic importance parameters even in unsupervised learning and also during testing. While the method is exciting and practically very useful, there is no geometric motivation behind the algorithm. Here we reverse engineer the importance term used in MAS and show a simple regularizer that leads to the MAS algorithm and more importantly provides a geometric interpretation for the algorithm.

Let the regularizer for Task B be the expected absolute difference between squared ℓ_2 norms of the output of the network for samples from Task A:

$$\arg\min_{\boldsymbol{\theta}} \tilde{\mathscr{L}}^B(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \mathscr{L}^B(\boldsymbol{\theta}) + \lambda \mathbb{E}_{\boldsymbol{x}\sim p_X^A}\left[\frac{1}{2}(\|\phi(\boldsymbol{x};\boldsymbol{\theta})\|^2 - \|\phi(\boldsymbol{x};\boldsymbol{\theta}_A^*)\|^2)^2\right] \tag{6}$$

It is straightforward to show that using the second-order Taylor expansion of the above regularizer leads to:

$$\arg\min_{\boldsymbol{\theta}} \tilde{\mathscr{L}}^B(\boldsymbol{\theta}) \quad = \quad \arg\min_{\boldsymbol{\theta}} \mathscr{L}^B(\boldsymbol{\theta}) + \lambda \sum_{m=1}^{M} [\Omega]_{m,m} [\boldsymbol{\theta} - \boldsymbol{\theta}_A^*]_m^2 \tag{7}$$

where $[\Omega]_{m,n} = \mathbb{E}_{\boldsymbol{x}\sim p_X^A}\left[(\frac{\partial\|\phi(\boldsymbol{x};\boldsymbol{\theta})\|^2}{\partial[\boldsymbol{\theta}]_m})(\frac{\partial\|\phi(\boldsymbol{x};\boldsymbol{\theta})\|^2}{\partial[\boldsymbol{\theta}]_n})\right]$, which is the importance parameter used by [45]. From a geometric perspective, MAS preserves the norms of the samples from Task A while learning Task B. In other words; MAS enforces closeness of $\boldsymbol{\theta}$ to $\boldsymbol{\theta}_A^*$ in a Riemmanian pseudo-manifold induced by matrix Ω.

The general idea of using the expected value of a "suitable" distance/divergence between samples, $\mathbb{E}_{\boldsymbol{x}\sim p_X^A} d(\phi(\boldsymbol{x};\boldsymbol{\theta}), \phi(\boldsymbol{x};\boldsymbol{\theta}_A^*))$ and leveraging its second-order Taylor expansion to obtain synaptic importance values is crucial here and could lead to various undiscovered algorithms based on new distances/divergences. We refer to these approaches as sample-based regularization methods.

Preserving Distribution of an Arbitrary Layer

We approach the problem of overcoming catastrophic forgetting from the angle of preserving Task A's distribution at an arbitrary layer of the neural network, when learning Task B. Let $\boldsymbol{z}_i^A = \phi(\boldsymbol{x}_i^A;\boldsymbol{\theta}) \in \mathbb{R}^K$ be the output of the network, for a sample from Task A, at the target layer (e.g., output logits in a classifier, or reconstructed image of an autoencoder). The distribution of the random variable $\boldsymbol{z}^A$ in the target layer follows from the Random Variable Transform theorem [46]:

$$p_Z^A(\boldsymbol{z}|\boldsymbol{\theta}) = \int_{\mathbb{X}} p_X^A(\boldsymbol{x})\delta(\boldsymbol{z} - \phi(\boldsymbol{x};\boldsymbol{\theta}))dx \tag{8}$$

Then we propose the following general regularization to overcome forgetting when learning task B:

$$\arg\min_{\boldsymbol{\theta}} \tilde{\mathscr{L}}^B(\boldsymbol{\theta}) \quad = \quad \arg\min_{\boldsymbol{\theta}} \mathscr{L}^B(\boldsymbol{\theta}) + \lambda d(p_Z^A(\cdot|\boldsymbol{\theta}), p_Z^A(\cdot|\boldsymbol{\theta}_A^*)) \tag{9}$$

where $d(\cdot,\cdot)$ is a discrepancy measure between the two probability distributions defined in $\mathbb{R}^K$. Note that Equation 9 could be performed with any discrepancy measure or distance, e.g., the Wasserstein distance [47, 48], using a memory replay strategy. In what follows, we describe a 'suitable' distance $d(\cdot,\cdot)$ that: (1) respects the underlying geometry of the space, and (2) enables a similar strategy to that of the EWC framework to provide importance parameters.

Sliced Cramér Distance for Structural Plasticity

Cramér Distance

The p-Cramér distance [49, 50] between two one-dimensional probability density functions p_0 and p_1 is defined as the ℓ_p-norm between their cumulative distribution functions. Note that here we avoid any measure theoretic notations for simplicity. Let $q_i(t) = \int_{-\infty}^{t} p_i(\tau)d\tau$ denote the cumulative distribution function for p_i, then the p-Cramér distance is defined as:

$$C_p(p_0, p_1) = \left(\int_{\mathbb{R}} |q_0(t) - q_1(t)|^p dt \right)^{\frac{1}{p}} \tag{10}$$

for $p \geq 1$. Similar to the Wasserstein distance and unlike the KL-divergence and its symmetric form Jensen-Shannon distance (i.e., the square root of the Jensen-Shannon divergence), the Cramér distance respects the underlying geometry of the space. Moreover, the Cramér distance provides unbiased sample gradients [51], and for $p = 1$, C_1 is equivalent to the 1-Wasserstein distance, W_1. In addition, and similar to the Wasserstein metric, the dual of the Cramér distance is of the form of an integral probability metric [52].

To further demonstrate the favorable characteristics of this distance, consider the following parametric distribution matching in one-dimension, where the target distribution, p, is a box distribution defined on $\mathbb{R}$ and $p_\tau(t) = p(t-\tau)$ is the shifted version of p and the goal is to optimize τ to minimize the distance between p_τ and p ($\tau^* = 0$). For this simple setting, we calculate the energy landscape (i.e., the distance between p_τ and p) as a function of τ for the Jensen-Shannon distance, W_p, and C_p for $p = 1, 2$. Figure 6 shows the distributions p and p_τ on the left and the energy landscape as a function of τ on the right. It can be clearly seen that the Wasserstein and Cramér distances respect the underlying geometry of the problem, while the Jensen-Shannon (JS) distance fails to do so.

To extend the Cramér distance to higher-dimensional distributions, we utilize the idea of distribution slicing used in various recent publications [53, 54, 55]. We note that the sliced Cramér distance (also known as the Cramér-Wold distance) was recently used in [56] for generative modeling.

Sliced Cramér Distance

The idea of slicing a higher-dimensional distribution has roots in the Radon transform that is commonly used in computational tomography. The idea is to represent a high-dimensional distribution via the infinite set of its marginal distributions. In short, let p_0 and p_1 be d-dimensional probability

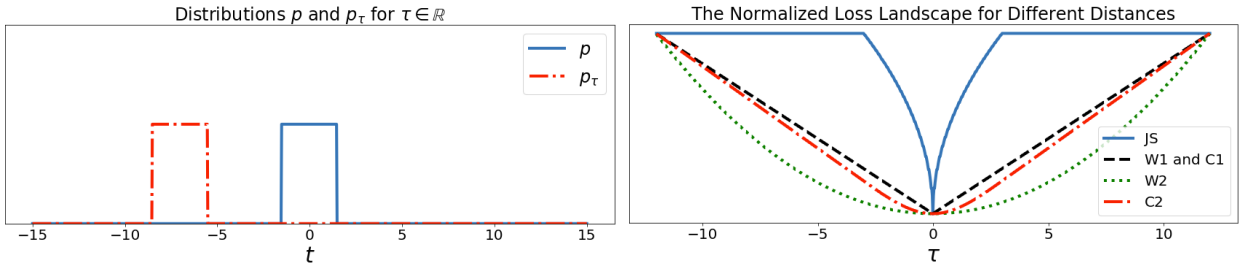


Figure 6: The energy landscape of various distances as a function of the translation parameter.

density functions defined on $\mathbb{X} \subset \mathbb{R}^d$, then their Radon transform is defined as:

$$\mathscr{R}p_i(t,\boldsymbol{\xi}) = \int_{\mathbb{X}} p_i(\boldsymbol{x})\delta(t-\boldsymbol{x}\cdot\boldsymbol{\xi})dx \tag{11}$$

for $\forall t \in \mathbb{R}$ and $\forall \boldsymbol{\xi} \in \mathbb{S}^{d-1}$ where $\mathbb{S}^{d-1}$ denotes the d-dimensional unit sphere. Note that $\mathscr{R}p_i(\cdot,\boldsymbol{\xi})$ is a so called slice of p_i, which is a one-dimensional marginal distribution of p_i. Let $\mathscr{R}q_i(\cdot,\boldsymbol{\xi})$ be the corresponding cumulative distribution function of $\mathscr{R}p_i(\cdot,\boldsymbol{\xi})$:

$$\mathscr{R}q_i(t,\boldsymbol{\xi}) = \int_{-\infty}^{t} \mathscr{R}p_i(\tau,\boldsymbol{\xi})d\tau \tag{12}$$

then the sliced Cramér distance between p_0 and p_1 is the expected value of the Cramér distance between their one-dimensional slices, i.e., $\mathscr{R}p_i(\cdot,\boldsymbol{\xi})$ when $\boldsymbol{\xi} \sim \mathscr{U}_{\mathbb{S}^{d-1}}$ for $\mathscr{U}_{\mathbb{S}^{d-1}}$ being the uniform distribution on the d-dimensional unit sphere. In other words, the sliced Cramér distance is defined as:

$$\begin{aligned} SC_p(p_0,p_1) &= \left(\int_{\mathbb{S}^{d-1}} C_p^p(\mathscr{R}p_0(\cdot,\boldsymbol{\xi}),\mathscr{R}p_1(\cdot,\boldsymbol{\xi}))d\boldsymbol{\xi}\right)^{\frac{1}{p}} \\ &= \left(\int_{\mathbb{S}^{d-1}}\int_{\mathbb{R}} |\mathscr{R}q_0(t,\boldsymbol{\xi}) - \mathscr{R}q_1(t,\boldsymbol{\xi})|^p dt d\boldsymbol{\xi}\right)^{\frac{1}{p}} \end{aligned} \tag{13}$$

Now we are ready to propose our method for overcoming representation forgetting.

Overcoming Representation Forgetting

The critical point here is that the sample-based regularizers could over-estimate the importance of synapses, leading to intransigence faster. More importantly, a substantial expected change in the network's output (over individual samples) would not necessarily mean catastrophic forgetting. Changes in the network's output within a mode (e.g., in supervised classification within the distribution of a particular class) are harmless, so long as the representation of the data is not significantly changing. Our proposed regularizer tolerates such changes and therefore has the potential for better utilization of the network's learning capacity. We propose the following regularization for overcoming catastrophic forgetting:

$$\arg\min_{\boldsymbol{\theta}} \tilde{\mathscr{L}}^B(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \mathscr{L}^B(\boldsymbol{\theta}) + \lambda SC_2^2(p_Z^A(\cdot|\boldsymbol{\theta}_A^*), p_Z^A(\cdot|\boldsymbol{\theta})) \tag{14}$$

Equation 14 requires memory replay from the old task(s) while learning the new one. To transition from memory replay to selective synaptic plasticity, we derive the second-order Taylor expansion of the regularizer SC_2^2 around the optimal parameters for the previous tasks, θ_A^*. Assuming that $\boldsymbol{\theta} = \boldsymbol{\theta}_A^* + \delta\boldsymbol{\theta}$ where $\delta\boldsymbol{\theta} \to 0$, it is straightforward to show that:

$$SC_2^2(p_Z^A(\cdot|\boldsymbol{\theta}_A^*), p_Z^A(\cdot|\boldsymbol{\theta})) \approx (\delta\boldsymbol{\theta})^T \Gamma_{\boldsymbol{\theta}_A^*}(\delta\boldsymbol{\theta}) = \|\delta\boldsymbol{\theta}\|_{\Gamma_{\boldsymbol{\theta}_A^*}} \tag{15}$$

where $\Gamma_{\boldsymbol{\theta}_A^*}$ is defined as:

$$\Gamma_{\boldsymbol{\theta}_A^*} := \int_{\mathbb{S}^{d-1}} \int_{\mathbb{R}} \left(\frac{d\, \mathscr{R}q_Z^A(t, \boldsymbol{\xi}|\boldsymbol{\theta}_A^*)}{d\boldsymbol{\theta}} \right) \left(\frac{d\, \mathscr{R}q_Z^A(t, \boldsymbol{\xi}|\boldsymbol{\theta}_A^*)}{d\boldsymbol{\theta}} \right)^T dt d\boldsymbol{\xi} \tag{16}$$

Note that similar to $F_{\boldsymbol{\theta}_A^*}$, $\Gamma_{\boldsymbol{\theta}_A^*}$ is also positive semi-definite (PSD) and therefore our Sliced Cramér regularizer enforces closeness of $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^*$ in a Riemannian pseudo-manifold induced by the PSD matrix $\Gamma_{\boldsymbol{\theta}}$.

While the definition of Γ in Equation 16, regardless of its similarity to Equation 3, may seem intimidating, it leads to a straightforward empirical algorithm, which we discuss below. Before that, we point out that similar to the FIM, calculating Γ is also practically infeasible, and we approximate Γ with a diagonal matrix that simplifies the regularization into:

$$\arg\min_{\boldsymbol{\theta}} \tilde{\mathscr{L}}^B(\boldsymbol{\theta}) \quad = \quad \arg\min_{\boldsymbol{\theta}} \mathscr{L}^B(\boldsymbol{\theta}) + \lambda \sum_{m=1}^{M} [\Gamma]_{m,m} [\delta\boldsymbol{\theta}]_m^2 \tag{17}$$

Finally, we emphasize that while Equation 17 is similar to Equation 5, it enforces a very different constraint on the neural network. Equation 5 enforces conditional class likelihoods to be preserved, which is sample-based; however, our proposed formulation in Equation 17 preserves the distribution of network's outputs at a particular layer for old tasks, i.e., it preserves the distribution of the previously learned representations.

1.2.3 Neuromodulated Attention

Artificial attentional mechanisms in neural networks tend to respond to sensory inputs similarly regardless of context and goals [57, 58, 59]. However, biological systems select relevant information to guide behavior in the face of noisy and unreliable signals, as well as rapidly adapt to unforeseen situations. Goal-driven perception treats the same situation differently based on context and effectively directs attention to goal-relevant inputs. Often, these goals are unknown and must be learned through experience. Moreover, these goals or contexts can shift without warning. Goal-driven perception helps prevent overemphasis on less relevant stimuli and instead focus on critical stimuli that require an immediate response.

In the brain, neuromodulators are important contributors to attention and goal-driven perception. In particular, the cholinergic (ACh) system drives bottom-up, stimulus-driven attention, as well as top-down, goal-driven attention [60]. Furthermore, the ACh system increases attention to task-relevant stimuli, while decreasing attention to distractions [61, 62]. This procedure is similar to the

core idea behind contrastive Excitation Backpropagation (c-EB). In c-EB, a top-down excitation mask increments attention to the target features, and an inhibitory mask decrements attention to distractors [57]. The noradrenergic (NE) system responds to surprises or large deviations from priors [63]. When the NE system responds phasically, where the neural activity rapidly and transiently increases, it causes a network to reset (e.g., re-initializing activities) that allows rapid adaptation under unseen/new conditions [64, 65].

We modified a c-EB network for use in a goal-driven perception task, where the system had to increase attention to the intended goal object and decrease attention to the distractor. In the first experiment, we presented pairs of noisy Modified National Institute of Standards and Technology (MNIST) digits to the neural network. One goal class was to attend to the digit based on its parity (i.e., even or odd goal), and another goal class was to attend based on the magnitude of the digit (i.e., low- or high-value goal). In addition, we added a neuromodulatory model to the head of the network architecture that regulated goal selection. Similar to the model of the ACh and NE neuromodulatory systems proposed by [63], we framed the task as an attentional task where the goal (even, odd, low or high value) had to be learned from experience (*goal identity*) and the goal might be noisy and rewarded with some probability (*goal validity*). In the second experiment, we generalized our model to an action-based attention scenario, where "eat", "work-on-computer", "read", and "say-hi" were goal actions and the robot needed to attend to and retrieve objects that corresponded to the action.

1.2.4 Self-Preserving World Model

The power to simulate the dynamics of a given environment provides a learning agent the ability to not only re-experience past episodes, but also to generate potentially unseen experiences in preparation for encountering them. This idea has its foundations in model-based RL, however modern explorations have taken on many forms. One proposed approach seeks to learn an unsupervised temporal prediction model that compresses the complete history of (current state, current action, current reward, next state) transitions an agent experiences [66]. This 'World Model' is then provided to the agent as a tool to better inform its decision-making process in various ways. Recent explorations of this theoretical framework has shown the approach is feasible at least with a single environment, and has illustrated the potential for using training based on simulated rollouts of possible episodes from the learned World Model in an entirely off-line fashion [67].

One critical aspect of this framework, which has yet to be fully addressed, is the need to continually learn in the potentially very different domains of the environment the agent is experiencing. Particularly when using neural networks, the World Model learned in this fashion would be subject to catastrophic forgetting when an incomplete history of all previous transitions are stored (which inevitably would be the case in a true L2 scenario). [66] suggests some ideas for how to address this, however they remain under-specified and untested to date.

As noted above, recent approaches to the catastrophic forgetting problem propose to learn various forms of plasticity parameters such that once a set of weights are learned for a given distribution of input samples (e.g., a particular task) those weights are made static as new distributions of input samples arrive [44, 68]. Intrinsic to these approaches is the necessity to segment and label collections of input samples into discrete sets or 'tasks'. This is both difficult to do in the continuous

flow of experience, and undesirably inflexible given the potential benefit of transferring learned knowledge from one task to another.

Previous work in psychology and neuroscience have developed theories for how mammalian brains can perform this type of continual learning. One of the more prominent computational theories is the idea of Complimentary Learning Systems [69, 70], which posits that recent experiences are 'replayed' during both sleep and quiet rest. This allows the brain, or learning system, to interleave previous experiences so they can be slowly integrated into a more comprehensive and robust internal representation. This in-turn inspired connectionist models that used a form of this replay, referred to as 'pseudo-rehearsal', to preserve previous learned weights [71, 72]. The utility of such a mechanism to modern learning systems has been previously posited [73], however the limit and scope of its applications have yet to be fully understood.

Here we explore a potential solution to preserve the unsupervised learning of a World Model-like architecture by adapting the strategy of pseudo-rehearsal [71, 72]. Fundamentally this is accomplished by having the World Model generate rollouts of previous experiences that are then interleaved with new experiences, thereby making the input distribution reflect b oth p ast and present experiences. Critically, we show this method capable of preserving the previously learned experiences without the need for segmenting experience into discrete tasks. To do this we use a published version of the World Model architecture trained in sequence to capture the dynamics of a set of Atari 2600 games.

The major contributions of this work are (1) illustrating pseudo-rehearsal as a potential label-free approach to continual learning in pixel-based environments; (2) providing foundations for continual learning in model-based RL; and (3) exploring the relationship between proportion of interleaved pseudo-samples and performance retention. Figure 7 shows the concept of operations for the sequential learning of the World Model. Here 'i' indexes the tasks being learned, where the previously learned network (i.e., M_{i-1} is used to generate simulated rollouts to be interleaved with samples from the current environment i. The weights from M_{i-1} are used as an initialization for the current M_i where gradients are being applied (indicated by the dashed blue box).

Related Work

The original ideas surrounding pseudo-rehearsal were first developed within a relatively simple connectionist framework that relied on input and output to be sparse low-dimensional binary patterns. In general, these methods used random input patterns to generate corresponding output patterns to create a set of input/output pairs that can be interleaved with new samples to preserve the current state of the network [71]. Later work explored the idea of capturing a whole sequence of experiences with a single pseudo-sample [72]. These methods were both inspired by and inspiring to theories of the how biological systems continually learn, namely the Complimentary Learning Systems theory. This theory posits that the hippocampus replays recent experiences so that the slower learning neocortex can consolidate that information into a more stable and robust representation [70].

This idea of replay based preservation of past experiences has taken root in recent work looking to accomplish the goal of sequentially learning multiple datasets. Most of this has been focused on classification tasks, and uses a form of pseudo-rehearsal referred to as 'deep generative replay' [74].

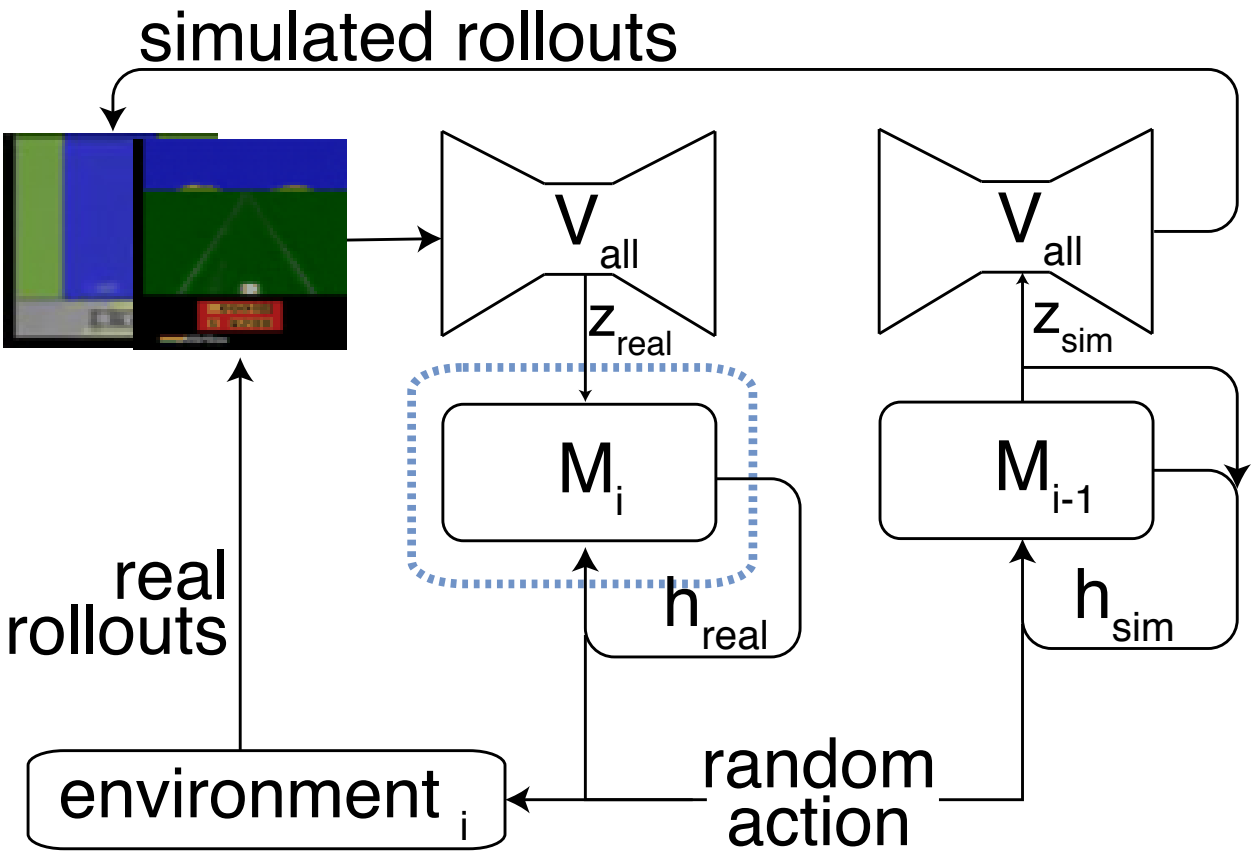


Figure 7: Concept diagram illustrating the World Model framework and the sequential training scheme.

The basic approach is the same, however the input/output patterns can be non-sparse continuous values, including pixel-based representations. The architecture used is a generative model of some type and is usually a Generative Adversarial Network [75]. Pushing this approach further, and taking inspiration from the Complimentary Learning Systems framework, a dual memory system used a form of deep generative replay for continual learning on a digits classification tasks [76]. Notably, this implementation showed increased accuracy and backward transfer compared to EWC, however the apparent difference in tasks tested was relatively small compared to the arbitrary set of Atari games used in the current work.

The Compress and Progress framework [42] can continually train RL agents in a series of complex pixel-based environments using a more scalable form of EWC along with Policy Distillation [77] to iteratively transfer learned policies into a single network. As mentioned above these plasticity based preservation methods require task labels to avoid catastrophic forgetting. Recent work, referred to as the Reinforcement-Pseudo-Rehearsal (RePR) model, has shown that the pseudo-rehearsal approach is also viable in RL [78]. Using a GAN based generative architecture paired with a modified DQN this approach used pseudo-rehearsal to iteratively maintain a single generative network trained on self-generated pseudo-samples and samples from an expert generative network trained on a specific task. For the integrated policy, similar to Progress and Compress [42], a single agent network was learned by using Policy Distillation to generate target output action distributions from both itself and an agent network trained on a specific task. This approach is very promising for iterative training in RL, however, it segments experience into tasks to train expert generative agent networks, and cannot provide temporally connected samples essential for n-step gradient based learning, e.g., backpropagation through time.

Within the World Model framework there have been a couple recent works that illustrate its effectiveness within RL. In particular, [67] showed that this framework could be used, within a limited set of environments, to train evolved neural networks to achieve SotA performance even when trained entirely on simulated rollouts. More recently, an approach that similarly learns a latent World Model used to inform action selection was able to learn simultaneously in six separate environments provided they were interleaved with each other [79].

1.2.5 Context-Skill Model

To enable autonomous agents to adapt safely to unseen situations, we developed a principled approach where a context module is coevolved with a skill module. The context module recognizes the variation and modulates the skill module so that the entire system performs well in unseen situations. The approach is evaluated in a challenging version of the Flappy Bird game where the effects of the actions vary over time. The Context-Skill Model leads to significantly more robust behavior in environments with previously unseen effects.

One popular approach to address this problem is few-shot learning, in particular meta-learning, either by utilizing gradients [80, 81, 82] or evolutionary procedures [83, 84]. In meta-learning, systems are trained by exposing them to a large number of tasks, and then tested for their ability to learn relevant but previously unseen tasks. There are also a number of approaches mostly for supervised learning setting where new labels need to be predicted based on limited number of

Algorithm 1 Sequential learning using World Model-based pseudo-rehearsal

Input: E set of potential environments, hyperparameters $nRealRollouts$, $nSimRollouts$
Parameter: network parameters V and M

Random initialization of V, M_i network parameters, initialize M_{i-1} as empty
while error in V is decreasing **do**
    optimize V on random samples from all E environments
end while
for e_i in E **do**
    while loss in M_i is decreasing **do**
        optimize M_i on $nRealRollouts$ drawn from e_i using random actions
        if M_{i-1} is not empty **then**
            seed M_{i-1} with $nSimRollouts$ random points in latent space
            optimize M on $nSimRollouts$ drawn from M_{i-1}
        end if
    end while
    $M_{i-1} = M_i$
end for

training data. However, applications of few-shot learning to control and decision-making, including RL problems, are limited so far [85].

Our approach presented here is motivated by work on opponent modeling in poker [86, 87]. In that domain, an effective approach was to evolve one neural network to decide what move to make, and another to modulate those decisions by taking the opponents playing style into account. When trained with only a small number of very simple but different opponents, the approach was able to generalize and play well against a wide array of opponents, include some that were much better than anything seen during training.

In poker, the opponent can be seen as the context for decision-making. Each decision needs to take into account how the opponent is likely to respond, and select the right action accordingly. The player can thus adapt to many different game playing situations immediately, even those that have not been encountered before. Here this approach is generalized and applied to control and decision-making more broadly. The main idea is that the history of the system is the context. A skill network reacts to the current situation, and a context network integrates observations over a longer time period. Together they learn to represent a wide variety of situations in a standardized manner so that a third, controller, network can make decisions robustly. Such a Context-Skill system can thus generalize to more situations than any of its components alone.

The Context-Skill Model was evaluated on several game domains: (1) Flappy Bird game extended to include more actions and physical effects (i.e., flap forward and drag in addition to flap upward and gravity); (2) Lunar Lander (LL) (from OpenAI Gym) extended with variations in main and side engine power as well as mass of the lander; and (3) CARLA autonomous driving environment where the steering curve, torque curve, and map can vary. Such extensions allow generating a range of unseen scenarios both by extending the range of effects of those actions as well as their

combinations. The approach generalizes remarkably well to new situations, and does so much better than its components alone. Context-Skill Model is thus a promising approach for building robust autonomous agents in real-world domains.

1.2.6 Reflexive Adaptation

When autonomous systems encounter new circumstances during deployment, which they have not encountered during factory training, the likelihood of emergency increases. We developed a reflexive adaptation module, intended to provide fast but adaptive reaction in case of immediate danger. Analogous to how they benefit humans and animals, reflexes in an autonomous agent are meant to be a rapid sequence of actions over a short time-span that allow the agent to react quickly and safely to sudden, surprise situations. In the context of video games, for example, this could be a scenario where the primary agent suddenly encounters a new type of fast-moving enemy fire that it must avoid in order to stay alive. If the agent is new to the environment, or has not encountered this type of adversary before, it will not yet have learned how to optimally respond to the given scenario and could take dangerous actions that lead to its destruction. In this case, a reflex module should recognize this imminent danger and takeover the agent's actions with a short-term reflexive response.

The motivation here is that this reflexive response, although perhaps sub-optimal and/or simple, would still be good enough to circumvent the emergency situation and allow the agent to survive and thus continue learning its task. This is particularly important during the initial stages of learning when an agent is introduced to a new environment or a new task. In such cases a performance drop is expected as the agent learns, and it is more likely to take dangerous actions that could lead to catastrophic situations such as destruction of the agent or loss of human life. A reflex module would allow an agent to survive longer in these situations, thereby exploring more of its environment/task and for longer periods of time. Not only can this be expected to expedite learning, but would also minimize repair/re-build cost and time associated with damage resulting from bad decisions.

The reflex module thus promotes safe exploration when an agent is learning a new task or is exposed to a new environment. However it is important to note here that reflexes are not simply a database of fixed, hand-designed action-sequences. Instead, the module is meant to provide an adaptive response, customized online to the particular emergency situation that is encountered during deployment. Here we report our seminal work towards devising an approach to incorporating reflexive adaptation into an autonomous agent.

1.2.7 Probabilistic Program Neurogenesis

Agents that act and learn in the real world must develop the skills to handle a wide variety of different tasks. These tasks are inevitably associated with different distributions of observations and responses. The diversity of tasks encountered in complex environments may make the task of continual learning seem improbable at best, and impossible at worst. However, tasks are often related to one another. For example, some of the visual attributes that an autonomous vehicle learns for detecting cars on the road may also be used when learning to identify motorcycles. Thus, an agent may be able to avoid learning a new task from scratch. The key is for the agent to efficiently

determine what information from previous tasks is reusable for a new task and to update its internal decision-making models with new, task-relevant information while avoiding catastrophic forgetting. Succinctly, an agent must be able to learn new tasks efficiently while not forgetting how to perform previously learned tasks.

Here, we focus on a class of methods for tackling the continual learning problem that dynamically change the structure of the neural network to accommodate new knowledge [88, 89, 90]. In particular, we address the challenge of continual learning by intelligently adding new neurons to an existing neural network (neurogenesis) so that it learns to solve a new task without forgetting how to solve previously learned tasks. The challenge is in determining how many neurons to add and where to add them so that a desired accuracy on the new task is achieved with as few new neurons as possible. In our approach the network is able to learn to leverage knowledge that was acquired when learning previous tasks, but which is relevant to the new task. This information sharing can substantially reduce the number of new neurons that need to be added to achieve a desired level of performance.

Our approach takes a unique angle towards neurogenesis by treating it as an optimization problem and using a novel combination of probabilistic programming and evolutionary computation. Specifically, we leverage the Estimation of Distribution Algorithm (EDA) framework. However, instead of representing the distribution over solutions using a Bayesian network, which is most common, we use a probabilistic program designed specifically for neurogenesis.

1.2.8 Meta-Learned Instinct Network

Deep RL has allowed many complex tasks to be solved, from playing video games to robotics [91, 92, 93, 94, 95, 96]. However, developing RL agents that can learn new tasks quickly while respecting safety restrictions is an unsolved challenge [97, 98, 99]. Most RL approaches rely on trial and error in order to solve tasks, and often these trials are based on random actions. Executing random actions to learn tasks is inherently problematic, especially in the real world, since it can cause damage to the agent and its surroundings. For example, a self-driving car cannot randomly try actions until it learns a new task because it will likely cause death and material damage.

In contrast to common RL approaches, animals in nature developed instinctual behaviors that prevent them from trying out actions that are likely dangerous to their lives. These instincts are innate behaviors provided by evolution to reduce the cost of first having to learn to avoid common dangers. For example, human infants have a congenital fear of spiders and snakes [100], likely because the evolved instinctual fear improved our ancestors' chances of survival. Other animals, such as rats, instinctively and without any learning avoid a specific compound found in carnivore urine [101].

We built on [102], where a policy neural network is split into two major components: a main network trained for a specific task, and a fixed pre-trained instinct network that transfers between tasks and overrides the main policy if the agent is about to execute a dangerous action. But meta-learning can be quite expensive since it relies on two nested learning loops: an inner task-specific loop and an outer meta-learning loop. Such high computation demands can limit the type of applications that

meta-learning can be applied to.

We found that the expensive meta-learning loop in [102] is not necessary to learn safely. We can train an instinct network efficiently on a single task where it is acceptable to make mistakes and then combine this pre-trained instinct network with a random policy to learn another task safely. An important aspect of our Instinct-Regulated RL (IR^{2}L) approach is the balance between learning and staying safe. The instinct network cannot be too restrictive (i.e., blocking the policy from doing anything) and has to make sure that the policy is still able to adapt. We show that this balance can be achieved by carefully tuning the hyperparameters of the reward used to train the instinct network, which includes both hazard risk minimization and task reward.

The results in a modification of the OpenAI Safety Gym environment [97] demonstrate that an instinct network allows an agent to learn new tasks while avoiding hazards. We also show that while a typical baseline approach that consists of pre-training a policy on a task with hazards (without an instinct network) can reduce safety violations to some extent, it performs significantly worse when compared to our IR^{2}L approach. In the future, the idea of combining a pre-trained instinct network with other RL methods could enable safer forms of artificial intelligence (AI) across a range of different tasks.

Background

Reinforcement Learning

We define an environment and the task that needs to be solved as a MDP denoted as a tuple of five elements $\mathscr{T} = \langle \mathscr{S}, \mathscr{A}, r, P, s_0 \rangle$; where $\mathscr{S}$ is the set of possible environment state observations, $\mathscr{A}$ is the set of actions the agent can execute, r is the numerical reward the agent receives executing an action at a certain state, $r : \mathscr{S} \times \mathscr{A} \to \mathbb{R}$, $P(\cdot|s,a)$ is a probability distribution of states reached by executing action a in the state s, and $s_0(\cdot)$ is the distribution of initial states.

Often, the agent interacts with the environment in episodes, where an episode is a sequence of actions that the agent executes starting with the initial state sampled from $s_0(\cdot)$ until a terminal state s_T. After the agent reaches s_T, the environment resets and the agent is initialized in one of the s_0 states. A sequence of $(\langle s_0, a_0, r_0, s_1 \rangle, \langle s_1, a_1, r_1, s_2 \rangle \ldots \langle s_i, a_i, r_i, s_T \rangle)$ tuples is called a trajectory and it is the data used to train the policy. The cumulative reward the agent collects in an episode is called return. Return from (s_i, a_i) to s_T is calculated as $R_i = r_i + \sum_{j=i+1} \gamma r_j$, where r_i the reward received by executing a_i at state s_i, γ is the reward discount factor that is treated as a fixed hyperparameter. The agent has to find a policy that maximizes the expected return $\mathbb{E}[R_0]$. In online reinforcement learning the task that the agent needs to solve can change, thus requiring the agent to re-adapt to maximize the expected j-th task-specific return $\mathbb{E}_{\mathscr{T}_j}[R_0]$. Here we assume that the tasks differ only in the task reward r that the environment gives to the agent.

Policy gradient methods [37] are a family of reinforcement learning methods that optimize policy parameters applying a gradient-based optimization algorithm with respect to expected episode returns $\mathbb{E}[R_0]$. A policy is an action probability distribution $\pi_\theta(a_i|s_i)$ conditioned on the current observed state s_i, where θ are the policy parameters. Normally, the policy is modeled with an artificial neural network [10], where parameters are the weights of the n etwork. The algorithm estimates the policy gradient and passes it to a gradient-based optimization algorithm like Stochastic

Gradient Descent [103, 104] or Adam [105]. The equation for the basic policy gradient estimator [37] is:

$$\hat{g}(\theta) = \mathbb{E}_\theta[\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(a_i|s_i)\hat{R}_i)], \tag{18}$$

where N is the total number of steps over all trajectories, and $\hat{R}_i$ is the return estimate from state s_i. The expectation $\mathbb{E}_\theta$ is approximated with a finite batch of sampled trajectories.

We used an upgraded policy gradient method called PPO [106, 107] since original policy gradient methods are prone to catastrophically large policy updates. PPO limits the policy gradient updates to a "trust region" to prevent catastrophic fall in a performance that can be caused by large policy changes.

AI Safety

An overview of AI safety methods can be found in [108] and [109]. A large body of work in the area of Safe AI focuses on constrained RL [110, 111]. Constrained RL depends on a-priori defined safety constraints which are states or actions that the agent should avoid. The constraints are often encoded within the environment reward functions. However, a challenge here is that when agents are transferred to a new task, learning again requires stochastic actions that could break safety. Additionally, there is the risk that the agent might forget its hazard avoidance skills during re-training on a new task.

In other related work, [112] introduced an approach in which a module is trained in a supervised way to predict the probability of catastrophic events. This module is integrated within the Q-learning objective. Another recent work that implements a similar modular idea of safety was introduced by [113], where a separate critic module is pre-trained on a random policy to approximate a Q-function that predicts a likelihood of hazard violation for an action. If the probability that an action will lead to a catastrophe exceeds a safety threshold, the probability to execute that action is set to 0. This approach is suitable for discreet action spaces. [114] provides an in-depth mathematical analysis of such approaches. Our work differs in that we in effect have two policies: the main policy and the instinct policy, both with their own action probability distributions and a modulation signal that can change the main policy's output. Furthermore, our approach is applicable to continuous action spaces.

In [115] a system called "shield" monitors the agent's actions and overrides them if they would violate the pre-specified safety constraints. The safety constraints are specified through temporal logic. [116] also employs goal specifications in temporal logic for safe RL. Other approaches to safe deep RL include estimating the safety of trajectories through Gaussian process estimation [117] or reducing catastrophic events through ensembles of neural models that capture uncertainty and classifiers trained to recognize dangerous actions [118].

Here we follow [97] and define an unsafe set of states as a subset of all states, $S_h \subset S$. The unsafe states represent situations or areas where we would like to minimize or completely prevent the agent from entering. For example, these states could represent walls that the agent should not collide with because of the resulting damage. Another example could be sidewalks and opposites lanes in

the case of self-driving vehicles. While not always damaging, the car entering those areas exposes other traffic members to an increased risk. We would like to make sure that the car is avoiding those areas even when trying actions to learn a new task. We can define hazard violations as a binary variable $h(s_i) \in 0, 1$, where $h(s_i)$ is 1 if the current state is undesirable, and 0 if it is not undesirable. We would like to minimize the expected violation return $V_{s_0} = \mathbb{E}[\sum_{s=s_0}^{s_T} h(s)]$ during the trajectory sampling, while still maintaining a reasonably good performance on the reward return R_0. The agent needs to know when it is in a hazards' neighborhood and to suppress the exploratory actions that can violate safety. Here, we assume that the subset S_h stays the same across different tasks $\mathcal{T}_j$ and $\mathcal{T}_k$.

1.2.9 Plastic Neuromodulated Network

Human intelligence, though specialized in some sense, is able to generally adapt to new tasks and solve problems from limited experience or few interactions. The field of meta-RL seeks to replicate such a flexible intelligence by designing agents that are capable of rapidly adapting to tasks from few interactions in an environment. The recent progress in the field such as [119, 120, 82, 121, 122, 123] has showcased SotA results. Agents endowed with such adaptation capabilities are a promising venue for developing much desired and needed AI systems and robots with lifelong learning dynamics.

When an agent's policy for a meta-RL problem is encoded by a neural network, neural representations are adjusted from a base pre-trained point to a configuration that is optimal to solve a specific task. Such dynamic representations are a key feature to enable an agent to rapidly adapt to different tasks. These representations can be derived from gradient-based approaches [82], context-based approaches such as memory [124, 119, 120] and probabilistic [123], or hybrid approaches (i.e., combination of gradient and context methods) [122]. The hybrid approach obtains a task context via gradient updates and thus dynamically alters the representations of the network. Context approaches such as Context Adaptation via Meta-Learning (CAVIA) [122] and Probabilistic Embeddings for Actor-Critic Meta-RL (PEARL) [123] are more interpretable as they disentangle task context from the policy network, thus the task context is used to achieve optimal policies for different tasks.

One limitation of such approaches is that they do not scale well as the problem complexity increases because of the demand to store many diverse policies to be reached within a single network. In particular, it is possible that, as tasks grow in complexity, the tasks similarities reduce and thus the network's representations required to solve each task optimally becomes dissimilar. We hypothesize that standard policy networks are not likely to produce diverse policies from a trained base representation because all neurons have a homogeneous role or function: thus, significant changes in the policy require widespread changes across the network. From this observation, we speculate that a network endowed with modulatory neurons (neuromodulators) has a significantly higher ability to modify its policy.

Our approach to overcome this limiting design factor in current meta-RL neural approaches is to introduce a neuromodulated policy network to increase their ability to encode rich and flexible dynamic representations. The rich representations are measured based on the dissimilarity of the representations across various tasks, and are useful when the optimal policy of an agent (input-to-

action mapping) is less similar across tasks. When combined with the CAVIA and PEARL meta-RL frameworks, the proposed approach produced better dynamic representations for fast adaptation as the neuromodulators in each layer serve as a means of directly altering the representations of the layer in addition to the task context.

Several designs exist for neuromodulation in the context of RL [125], either to gate plasticity [126, 31], gate neural activations [127] or alter high-level behavior [128]. The proposed mechanism in this work focuses on just one simple principle: modulatory signals alter the representations in each layer by gating the weighted sum of inputs of the standard neurons.

Our contribution is a neuromodulated policy network for meta-RL for solving increasingly difficult problems. The modular approach of the design allows for the proposed layer to be used with existing layers (such as standard fully connected layers and convolutional layers) when stacking them to form a deep network. The experimental evidence in this work demonstrates that neuromodulation is beneficial to adapt network representations with more flexibility in comparison to standard networks. Experimental evaluations were conducted across high-dimensional discrete and continuous control environments of increasing complexity using CAVIA and PEARL meta-RL algorithms. The results indicate that the neuromodulated networks show an increasing advantage as the problem complexity increases, while they perform comparably on simpler problems. The increased diversity of the representations from the neuromodulated policy network are examined and discussed below.

Related Work

Meta-RL. This work builds on the existing meta learning frameworks [129, 130, 81, 131] in the domain of reinforcement learning. Recent studies in meta-RL can be largely classified into optimization and context-based methods. Optimization methods [82, 132, 133, 134] seek to learn good initial parameters of a model that can be adapted with a few gradient steps to a specific task. In contrast, context-based methods seek to adapt a model to a specific task based on few-shot experiences aggregated into context variables. The context can be derived via probabilistic methods [123, 135], recurrent memory [120, 119], recursive networks [124] or the combination of probabilistic and memory [136, 137]. Hybrid methods [122, 121] combine optimization and context-based methods whereby task-specific context parameters are obtained via gradient updates.

Neuromodulation. Neuromodulation in biological brains is a process whereby a neuron alters or regulates the properties of other neurons in the brain [138]. The altered properties can be in either the neural activities or synaptic weights of the neurons. Well-known biological neuromodulators include dopamine, serotonin, acetylcholine, and noradrenaline [139, 140]. Such neuromodulators were described in [125] within the reinforcement learning computation framework, with dopamine loosely mapped to the reward signal error (like TD error), serotonin representing discount factor, acetylcholine representing learning rate and noradrenaline representing randomness in a policy's action distribution. Several studies have drawn inspiration from neuromodulation and applied it to gradient-based RL [128, 31] and neuroevolutionary RL [141, 126, 142] for dynamic task settings. In broader machine learning, neuromodulation has been applied to goal-driven perception [3], and also in continual learning setting [127] where it was combined with meta-learning to sequentially learn a number of classification tasks without catastrophic forgetting. The neuromodulators used in these studies have different designs or functions: plasticity gating [126, 31], activation gating [127],

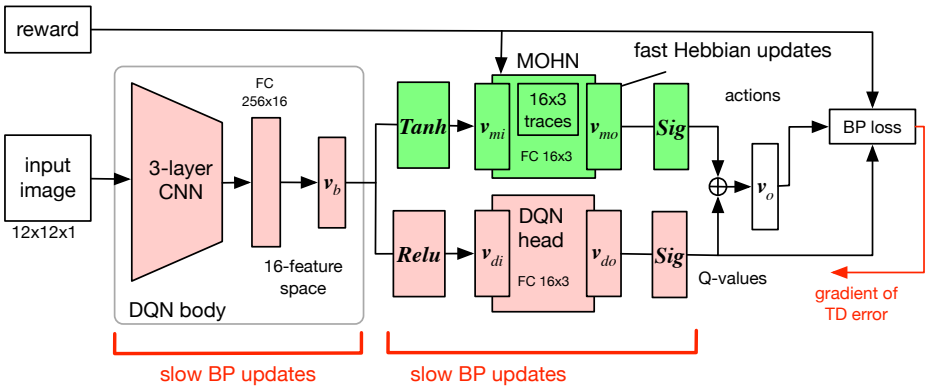


Figure 8: Graphical representation of MOHQA.

direct action modification in a policy [128].

1.3 Methods, Assumptions, and Procedures

1.3.1 MOdulated Hebbian Network

MOHQA is composed of two main parts: a Deep Q-Network (DQN) [143], and a MOdulated Hebbian Network (MOHN) that is plugged into the Q-network as a parallel unit to the DQN head (Figure 8). DQN's main contribution is providing high-level features to MOHN, while MOHN's main contribution is decision-making. In particular, the DQN body feeds the feature space to both the DQN head and MOHN. The action-value vector is the sum of the outputs of the two heads. The learning in MOHN is regulated by modulated Hebbian plasticity. The learning in DQN is performed by backpropagation.

DQN Description
MOHN is a one layer network which is intended to work on high-level features, thus it needs a 'supplier' of high-level features. In this work DQN [143] is used but, in theory, any deep RL approach could have been used instead of DQN. A DQN is used to approximate the optimal Q-value function, defined as

$$Q(s,a)^{\star} = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s, a_t = a, \pi], \tag{19}$$

where r_t is the reward at time t, s_t is the state at time t, a_t is the action at time t and π is the policy. To solve instability issues caused by representing action-value pairs with the network, [143] proposed two innovations: (i) experience replay and (ii) a target network that is only periodically updated. Note, Φ^{-} and Φ denote set of parameters of target and predictor networks respectively. The parameters Φ^{-} are updated with Φ every K steps. The output of the DQN body is a set of features that are used as input to both MOHN and DQN heads. Both the DQN head and the DQN body are trained by back propagation to minimize the TD error [143].

MOHN Description
Associating stimulus-action pairs with distal rewards by means of MOHN

MOHN is an adaptation of a bio-inspired, unsupervised and modulated Hebbian network proposed in [32, 29]. In those studies, a Hebbian modulated network was shown to cope with two challenges of confounding POMDPs — sparse rewards and confounding stimuli — outside a RL framework, i.e., when neither states nor TD errors are defined. It is worth reiterating that, in confounding POMDPs, the TD error computation is inaccurate and cannot be used to propagate the Q-values. Moreover, even memory does not help to solve the problem as the history of observations maps to a large space due to stochastic observations, making learning history of action-observations ineffective. MOHN solves sparse reward and confounding stimuli problems by using rarely correlated eligibility traces and Hebbian learning. Eligibility traces do not rely on TD error computation, instead they directly associate activation of certain neurons with obtaining rewards later; while rare correlation allows MOHN to cope with noise in inputs, grow weights without instability and utilize all the weights efficiently. Additionally, as MOHN is utilizing Hebbian learning, it can have a high learning rate without instability issues. The result is that observations-action pairs are effectively associated with later rewards by means of traces, weights responsible for an action are updated rapidly, and intervening events between actions and rewards are "ignored".

MOHN learning mechanism using STDP-inspired plasticity and neural eligibility traces
The learning mechanism in MOHN can be briefly summarized as follows. An action taken in a state triggers a number of traces for weights between highly correlated input/output neurons. The traces are decayed exponentially with each step. At each step, weights are either decreased if the agent did not score or increased if the agent scored. This mechanism enables the network to find the correct association between actions and rewards with intervening confounding stimuli, thus avoiding shortcomings of propagating TD error.

To be more specific, neural eligibility traces generation mechanisms use two principles: (i) causal relationships between observation and actions and (ii) sparse correlations. The causal relationships are derived by applying the Hebbian multiplication rule to successive, rather than simultaneous, simulation steps, so that Hebbian terms capture the contribution of presynaptic activity to the activity of a postsynaptic neuron, similarly to the spike-timing-dependent plasticity (STDP) rule. This is also sometimes called an asymmetrical learning window [144]. Sparse correlations are explicitly imposed by selecting the top $\theta\%$ and bottom $\theta\%$ of correlations/decorrelations that loosely map the concept of the STDP time window to a rate-based model [33]. A modified Hebbian term Θ between a presynaptic neuron i and a postsynaptic neuron j is updated according to the equation:

$$\Theta_{pre\rightarrow post}(t) = \begin{cases} 1 & \text{if } v_i \cdot v_j \text{ is in top } \theta\%, \\ -1 & \text{if } v_i \cdot v_j \text{ is in bottom } \theta\%, \\ 0 & \text{otherwise,} \end{cases} \tag{20}$$

where v_i is the output values of the presynaptic neurons equivalent to the input to MOHN, $\mathbf{v}_{mi}$ (Figure 8), minus its own running average to enhance the detection of changes in the feature space and v_j is defined as:

$$\mathbf{v}_j = \Gamma\big(\mathbf{v}_{mo}\big), \tag{21}$$

where $\mathbf{v}_{mo}$ is an output of MOHN head and Γ is a function that returns a one-hot vector with

the 1 value at the index of the maximum value (note that v_i is used both in forward pass and weight updates, while v_j is used only for weight update and v_{mo} is used in forward pass). The one-hot function has the purpose of increasing the traces for the weights that are afferent to the action-triggering neuron. Finally, the neural eligibility traces matrix $E(t)$ is formulated based on Θ and a time decay constant τ_E as

$$\dot{E}(t) \quad = \quad -E(t)/\tau_E + \Theta(t) \quad . \tag{22}$$

To update weights, the traces matrix $E(t)$ is multiplied with modulatory signal (reward plus a small baseline modulation, i.e., r(t) + b):

$$\Delta \mathbf{w}(t) \quad = \quad (r(t)+b) \cdot E(t). \tag{23}$$

The weights are clipped in the interval [-1,1] to contain Hebbian updates [145, 146].

Integration of DQN with MOHN
To integrate DQN with MOHN it is necessary to devise an action selection mechanism between two head modules of Figure 8. Thus, the action is chosen by combining the outputs of the two heads.

Outputs of both heads are combined to create the MOHQA Q-values, $\mathbf{v}_o$, which are defined as:

$$\mathbf{v}_o = \mathbf{v}_{mo} + \mathbf{v}_{do} = \mathbf{v}_{mo} + Q(s, A; \Phi_i^-), \tag{24}$$

where s indicates that an observation is used to approximate the state, even when this is incorrect due to partial observability. Then action is chosen from MOHQA Q-values as follows:

$$a_b = \arg\max_a(\mathbf{v}_o), \tag{25}$$

To help DQN learn features in the desired state, the loss function uses the difference between best action as indicated by the Q-value of both DQN and MOHN and the Q-value indicated by DQN alone:

$$\begin{aligned} L(\Phi) = & \mathbb{E}_{(s,a,r,s')\ U(D)} \Big(r + \gamma \max_{a_b} \mathbf{v}_o(s', a_b, \Phi_i^-) \\ & - Q(s, a; \Phi_i) \Big)^2, \end{aligned} \tag{26}$$

where Φ^- are parameters from the target network and Φ are parameters from the prediction network.

1.3.2 Sliced Cramer Preservation

Here we derive the algorithmic steps required to calculate Γ empirically. The empirical distribution at the network's output can be written as, $p_Z^A(\boldsymbol{z}|\boldsymbol{\theta}) \approx \frac{1}{N}\sum_{n=1}^{N} \delta(\boldsymbol{z} - \phi(\boldsymbol{x}_n^A; \boldsymbol{\theta}))$, and the slices of this empirical distribution are defined as:

$$\mathscr{R} p_Z^A(t, \boldsymbol{\xi}|\boldsymbol{\theta}) \approx \frac{1}{N} \sum_{n=1}^{N} \delta(t - \boldsymbol{\xi} \cdot \phi(\boldsymbol{x}_n^A; \boldsymbol{\theta}))$$

Let $u(\cdot)$ denote the step function, which is the cumulative distribution of the Dirac delta function. Then the cumulative distribution of $\mathscr{R}p_Z^A(t,\boldsymbol{\xi}|\boldsymbol{\theta})$ can be written as, $\mathscr{R}q_Z^A(t,\boldsymbol{\xi}|\boldsymbol{\theta}) \approx \frac{1}{N}\sum_{n=1}^{N} u(t-\boldsymbol{\xi}\cdot\phi(\boldsymbol{x}_n^A;\boldsymbol{\theta}))$. Therefore we have:

$$\frac{d\,\mathscr{R}q_Z^A(t,\boldsymbol{\xi}|\boldsymbol{\theta})}{d\boldsymbol{\theta}} \approx \frac{1}{N}\sum_{n=1}^{N}\left(\frac{-d\,\boldsymbol{\xi}\cdot\phi(\boldsymbol{x}_n^A;\boldsymbol{\theta})}{d\boldsymbol{\theta}}\right)\delta(t-\boldsymbol{\xi}\cdot\phi(\boldsymbol{x}_n^A;\boldsymbol{\theta})) \tag{27}$$

substituting Equation 27 into Equation 16 and using a Monte-Carlo approximation of the integration of $\mathbb{S}^{d-1}$, with L samples, leads to:

$$\Gamma = \frac{1}{L}\sum_{l=1}^{L}\left(\frac{d\,\boldsymbol{\xi}_l\cdot\bar{\boldsymbol{z}}}{d\boldsymbol{\theta}}\right)\left(\frac{d\,\boldsymbol{\xi}_l\cdot\bar{\boldsymbol{z}}}{d\boldsymbol{\theta}}\right)^T \tag{28}$$

where $\boldsymbol{\xi}_l$s are randomly drawn from $\mathbb{S}^{K-1}$, and $\bar{\boldsymbol{z}} = \frac{1}{N}\sum_{n=1}^{N}\phi(\boldsymbol{x}_n^A;\boldsymbol{\theta}_A^*)$. Given that calculation of matrix Γ is not practically feasible (due to the large number of parameters of a deep neural network), we follow the work of [44] and approximate Γ to be a diagonal matrix, which simplifies to:

$$\left[\Gamma_{\boldsymbol{\theta}_A^*}\right]_{i,i} = \frac{1}{L}\sum_{l=1}^{L}\left(\frac{1}{N}\sum_{n=1}^{N}\frac{d\,\boldsymbol{\xi}_l\cdot\phi(\boldsymbol{x}_n^A;\boldsymbol{\theta}_A^*)}{d\boldsymbol{\theta}_i}\right)^2 = \frac{1}{L}\sum_{l=1}^{L}\left(\frac{d\,\boldsymbol{\xi}_l\cdot\bar{\phi}_A^*}{d\boldsymbol{\theta}_i}\right)^2$$

where $\bar{\phi}_A^* = \frac{1}{N}\sum_{n=1}^{N}\phi(\boldsymbol{x}_n^A;\boldsymbol{\theta}_A^*)$.

Online extension of the algorithm

To extend the framework into sequential learning of multiple tasks and to avoid memorizing task-specific Γs (or their diagonals), we follow the EWC++ framework proposed by [41], which is, in essence, identical to the online-EWC proposed by [42]. The EWC++ (and online-EWC) methods ameliorate the need for predicting task identities and calculating task-specific FIMs and keep the memory requirement of the method constant (EWC requires linear growth of memory as a function of number of tasks). In EWC++, given $F_\theta^{(t-1)}$ at task $(t-1)$, the accumulated FIM after learning task t is calculated as $F_{\boldsymbol{\theta}}^{(t)} = \alpha F_{\boldsymbol{\theta}_t^*} + (1-\alpha)F_{\boldsymbol{\theta}}^{(t-1)}$, where $F_{\boldsymbol{\theta}_t^*}$ is the task-specific FIM for task t, and $\alpha \in [0,1)$ is a hyperparameter that indicates the importance of preserving the most recent task over the older ones. Similarly, we use,

$$\Gamma_{\boldsymbol{\theta}}^{(t)} = \alpha\Gamma_{\boldsymbol{\theta}_t^*} + (1-\alpha)\Gamma_{\boldsymbol{\theta}}^{(t-1)} \tag{29}$$

to obtain the sliced Cramér regularizer for task $(t+1)$.

Taylor Expansion of the KL-Divergence

For the sake of completion, here we derive the second-order Taylor expansion of D_{KL},

$$D_{\mathrm{KL}}(p_{\boldsymbol{\theta}_0}||p_{\boldsymbol{\theta}}) = \int_X p_{\boldsymbol{\theta}_0}(x)log\left(\frac{p_{\boldsymbol{\theta}_0}(x)}{p_{\boldsymbol{\theta}}(x)}\right)dx$$

around $\boldsymbol{\theta}_0$ where we can write $\boldsymbol{\theta} = \boldsymbol{\theta}_0 + \delta\boldsymbol{\theta}$. The second-order Taylor expansion is:

$$D_{\mathrm{KL}}(p_{\boldsymbol{\theta}_0}||p_{\boldsymbol{\theta}}) \approx D_{\mathrm{KL}}(p_{\boldsymbol{\theta}_0}||p_{\boldsymbol{\theta}_0}) + \delta\boldsymbol{\theta}^T\left(\frac{dD_{\mathrm{KL}}(p_{\boldsymbol{\theta}_0}||p_{\boldsymbol{\theta}})}{d\boldsymbol{\theta}}|_{\boldsymbol{\theta}_0}\right) + \frac{1}{2}\delta\boldsymbol{\theta}^T\left(\frac{d^2D_{\mathrm{KL}}(p_{\boldsymbol{\theta}_0}||p_{\boldsymbol{\theta}})}{d\boldsymbol{\theta}^2}|_{\boldsymbol{\theta}_0}\right)\delta\boldsymbol{\theta}$$

1. Where for the first-order term we have:

$$\frac{dD_{\mathrm{KL}}(p_{\boldsymbol{\theta}_0}||p_{\boldsymbol{\theta}})}{d\boldsymbol{\theta}} = -\int_X p_{\boldsymbol{\theta}_0}(x)\frac{d\ log(p_{\boldsymbol{\theta}}(x))}{d\boldsymbol{\theta}}dx = -\int_X \frac{p_{\boldsymbol{\theta}_0}(x)}{p_{\boldsymbol{\theta}}(x)}\frac{dp_{\boldsymbol{\theta}}(x)}{d\boldsymbol{\theta}}dx$$

   Therefore at $\boldsymbol{\theta} = \boldsymbol{\theta}_0$ we have:

$$\frac{dD_{\mathrm{KL}}(p_{\boldsymbol{\theta}_0}||p_{\boldsymbol{\theta}})}{d\boldsymbol{\theta}}|_{\boldsymbol{\theta}_0} = -\int_X \frac{dp_{\boldsymbol{\theta}}(x)}{d\boldsymbol{\theta}}dx = \frac{d}{d\boldsymbol{\theta}}\left(\int_X p_{\boldsymbol{\theta}_0}(x)dx\right) = 0$$

2. and for the second-order term:

$$\begin{aligned}\frac{d^2D_{\mathrm{KL}}(p_{\boldsymbol{\theta}_0}||p_{\boldsymbol{\theta}})}{d\boldsymbol{\theta}^2} &= \frac{d}{d\boldsymbol{\theta}}\left(-\int_X \frac{p_{\boldsymbol{\theta}_0}(x)}{p_{\boldsymbol{\theta}}(x)}\frac{dp_{\boldsymbol{\theta}}(x)}{d\boldsymbol{\theta}}dx\right) \\ &= \int_X \frac{p_{\boldsymbol{\theta}_0}(x)}{p^2_{\boldsymbol{\theta}}(x)}\left(\frac{dp_{\boldsymbol{\theta}}(x)}{d\boldsymbol{\theta}}\right)\left(\frac{dp_{\boldsymbol{\theta}}(x)}{d\boldsymbol{\theta}}\right)^T dx - \\ &\quad \int_X \frac{p_{\boldsymbol{\theta}_0}(x)}{p_{\boldsymbol{\theta}}(x)}\frac{d^2\ p_{\boldsymbol{\theta}}(x)}{d\boldsymbol{\theta}^2}dx\end{aligned}$$

   Therefore at $\boldsymbol{\theta} = \boldsymbol{\theta}_0$ we have:

$$\begin{aligned}\frac{d^2D_{\mathrm{KL}}(p_{\boldsymbol{\theta}_0}||p_{\boldsymbol{\theta}})}{d\boldsymbol{\theta}^2}|_{\boldsymbol{\theta}_0} &= \int_X p_{\boldsymbol{\theta}_0}(x)\left(\frac{1}{p_{\boldsymbol{\theta}_0}(x)}\frac{dp_{\boldsymbol{\theta}_0}(x)}{d\boldsymbol{\theta}}\right)\left(\frac{1}{p_{\boldsymbol{\theta}_0}(x)}\frac{dp_{\boldsymbol{\theta}_0}(x)}{d\boldsymbol{\theta}}\right)^T dx - \\ &\quad \int_X \frac{d^2\ p_{\boldsymbol{\theta}_0}(x)}{d\boldsymbol{\theta}^2}dx \\ &= \int_X p_{\boldsymbol{\theta}_0}(x)\left(\frac{d\ log(p_{\boldsymbol{\theta}_0}(x))}{d\boldsymbol{\theta}}\right)\left(\frac{d\ log(p_{\boldsymbol{\theta}_0}(x))}{d\boldsymbol{\theta}}\right)^T dx \\ &= \mathbb{E}_{x\sim p_{\boldsymbol{\theta}_0}}\left[\left(\frac{d\ log(p_{\boldsymbol{\theta}_0}(x))}{d\boldsymbol{\theta}}\right)\left(\frac{d\ log(p_{\boldsymbol{\theta}_0}(x))}{d\boldsymbol{\theta}}\right)^T\right] = F\end{aligned}$$

   which is the FIM.

Finally, putting everything together we have:

$$D_{\mathrm{KL}}(p_{\boldsymbol{\theta}_0}||p_{\boldsymbol{\theta}}) \approx \frac{1}{2}\delta\boldsymbol{\theta}^T F\delta\boldsymbol{\theta}$$

which concludes the derivation.

Taylor Expansion of the MAS Regularizer

Here we drive the second-order Taylor expansion of the MAS regularizer in Equation 6,

$$\begin{aligned}MAS_{reg} &= \mathbb{E}_{\boldsymbol{x}\sim p_X}\left[\frac{1}{2}(\|\phi(\boldsymbol{x};\boldsymbol{\theta})\|^2 - \|\phi(\boldsymbol{x};\boldsymbol{\theta}_0)\|^2)^2\right] \\ &= \frac{1}{2}\int_X p_X(\boldsymbol{x})(\|\phi(\boldsymbol{x};\boldsymbol{\theta})\|^2 - \|\phi(\boldsymbol{x};\boldsymbol{\theta}_0)\|^2)^2 d\boldsymbol{x}\end{aligned}$$

around $\boldsymbol{\theta}_0$. The second-order Taylor expansion is:

$$MAS_{reg} \approx \delta\boldsymbol{\theta}^T \left(\frac{d\ MAS_{reg}}{d\boldsymbol{\theta}}|_{\boldsymbol{\theta}_0} \right) + \frac{1}{2}\delta\boldsymbol{\theta}^T \left(\frac{d^2 MAS_{reg}}{d\boldsymbol{\theta}^2}|_{\boldsymbol{\theta}_0} \right) \delta\boldsymbol{\theta}$$

1. Where for the first-order term we have:

$$\left[\frac{dMAS_{reg}}{d\boldsymbol{\theta}} \right]_m = \int_X p_X(\boldsymbol{x}) \frac{d\ \|\phi(\boldsymbol{x};\boldsymbol{\theta})\|^2}{d\ [\boldsymbol{\theta}]_m} (\|\phi(\boldsymbol{x};\boldsymbol{\theta})\|^2 - \|\phi(\boldsymbol{x};\boldsymbol{\theta}_0)\|^2)$$

   which for $\boldsymbol{\theta} = \boldsymbol{\theta}_0$ is zero.

2. For the second-order term we have:

$$\begin{aligned} \left[\frac{d^2 MAS_{reg}}{d\boldsymbol{\theta}^2} \right]_{m,n} &= \int_X p_X(\boldsymbol{x}) \frac{d\ \|\phi(\boldsymbol{x};\boldsymbol{\theta})\|^2}{d\ [\boldsymbol{\theta}]_m} \frac{d\ \|\phi(\boldsymbol{x};\boldsymbol{\theta})\|^2}{d\ [\boldsymbol{\theta}]_n} d\boldsymbol{x} + \\ & \int_X p_X(\boldsymbol{x}) \left[\frac{d^2\ \|\phi(\boldsymbol{x};\boldsymbol{\theta})\|^2}{d\ \boldsymbol{\theta}^2} \right]_{m,n} (\|\phi(\boldsymbol{x};\boldsymbol{\theta})\|^2 - \|\phi(\boldsymbol{x};\boldsymbol{\theta}_0)\|^2) d\boldsymbol{x} \end{aligned}$$

   where evaluated at $\boldsymbol{\theta} = \boldsymbol{\theta}_0$ the second term on the right-hand-side vanishes.

Finally, putting everything together we have:

$$MAS_{reg} \approx \delta\boldsymbol{\theta}^T \underbrace{\mathbb{E}_{\boldsymbol{x}\sim p_X} \left[\left(\frac{d\ \|\phi(\boldsymbol{x};\boldsymbol{\theta})\|^2}{d\ \boldsymbol{\theta}} \right) \left(\frac{d\ \|\phi(\boldsymbol{x};\boldsymbol{\theta})\|^2}{d\ \boldsymbol{\theta}} \right)^T \right]}_{\Omega} \delta\boldsymbol{\theta}$$

Taylor Expansion of the Sliced Cramér Distance

Here we derive the second-order Taylor expansion of the squared 2-Sliced Cramér distance,

$$SC_2^2(p_{\boldsymbol{\theta}_0}, p_{\boldsymbol{\theta}}) = \int_{\mathbb{S}^{d-1}} \int_{\mathbb{R}} |\mathscr{R}q_{\boldsymbol{\theta}_0}(t,\boldsymbol{\xi}) - \mathscr{R}q_{\boldsymbol{\theta}}(t,\boldsymbol{\xi})|^2 dt d\boldsymbol{\xi}$$

around $\boldsymbol{\theta}_0$, as reported in Equation 16. The second-order Taylor expansion is:

$$SC_2^2(p_{\boldsymbol{\theta}_0}, p_{\boldsymbol{\theta}}) \approx SC_2^2(p_{\boldsymbol{\theta}_0}, p_{\boldsymbol{\theta}_0}) + \delta\boldsymbol{\theta}^T \left(\frac{d\ SC_2^2(p_{\boldsymbol{\theta}_0}, p_{\boldsymbol{\theta}})}{d\boldsymbol{\theta}}|_{\boldsymbol{\theta}_0} \right) + \frac{1}{2}\delta\boldsymbol{\theta}^T \left(\frac{d^2 SC_2^2(p_{\boldsymbol{\theta}_0}, p_{\boldsymbol{\theta}})}{d\boldsymbol{\theta}^2}|_{\boldsymbol{\theta}_0} \right) \delta\boldsymbol{\theta}$$

1. Where for the first-order term we have:

$$\frac{d\ SC_2^2(p_{\boldsymbol{\theta}_0}, p_{\boldsymbol{\theta}})}{d\boldsymbol{\theta}} = 2 \int_{\mathbb{S}^{d-1}} \int_{\mathbb{R}} \frac{d\ \mathscr{R}q_{\boldsymbol{\theta}}(t,\boldsymbol{\xi})}{d\boldsymbol{\theta}} (\mathscr{R}q_{\boldsymbol{\theta}}(t,\boldsymbol{\xi}) - \mathscr{R}q_{\boldsymbol{\theta}_0}(t,\boldsymbol{\xi})) dt d\boldsymbol{\xi}$$

   which for $\boldsymbol{\theta} = \boldsymbol{\theta}_0$ is equal to zero.

2. For the second-order term we have:

$$\begin{aligned} \frac{d^2\ SC_2^2(p_{\boldsymbol{\theta}_0}, p_{\boldsymbol{\theta}})}{d\boldsymbol{\theta}^2} &= 2 \int_{\mathbb{S}^{d-1}} \int_{\mathbb{R}} \frac{d^2\ \mathscr{R}q_{\boldsymbol{\theta}}(t,\boldsymbol{\xi})}{d\boldsymbol{\theta}^2} (\mathscr{R}q_{\boldsymbol{\theta}}(t,\boldsymbol{\xi}) - \mathscr{R}q_{\boldsymbol{\theta}_0}(t,\boldsymbol{\xi})) dt d\boldsymbol{\xi} + \\ & 2 \int_{\mathbb{S}^{d-1}} \int_{\mathbb{R}} \left(\frac{d\ \mathscr{R}q_{\boldsymbol{\theta}}(t,\boldsymbol{\xi})}{d\boldsymbol{\theta}} \right) \left(\frac{d\ \mathscr{R}q_{\boldsymbol{\theta}}(t,\boldsymbol{\xi})}{d\boldsymbol{\theta}} \right)^T dt d\boldsymbol{\xi} \end{aligned}$$

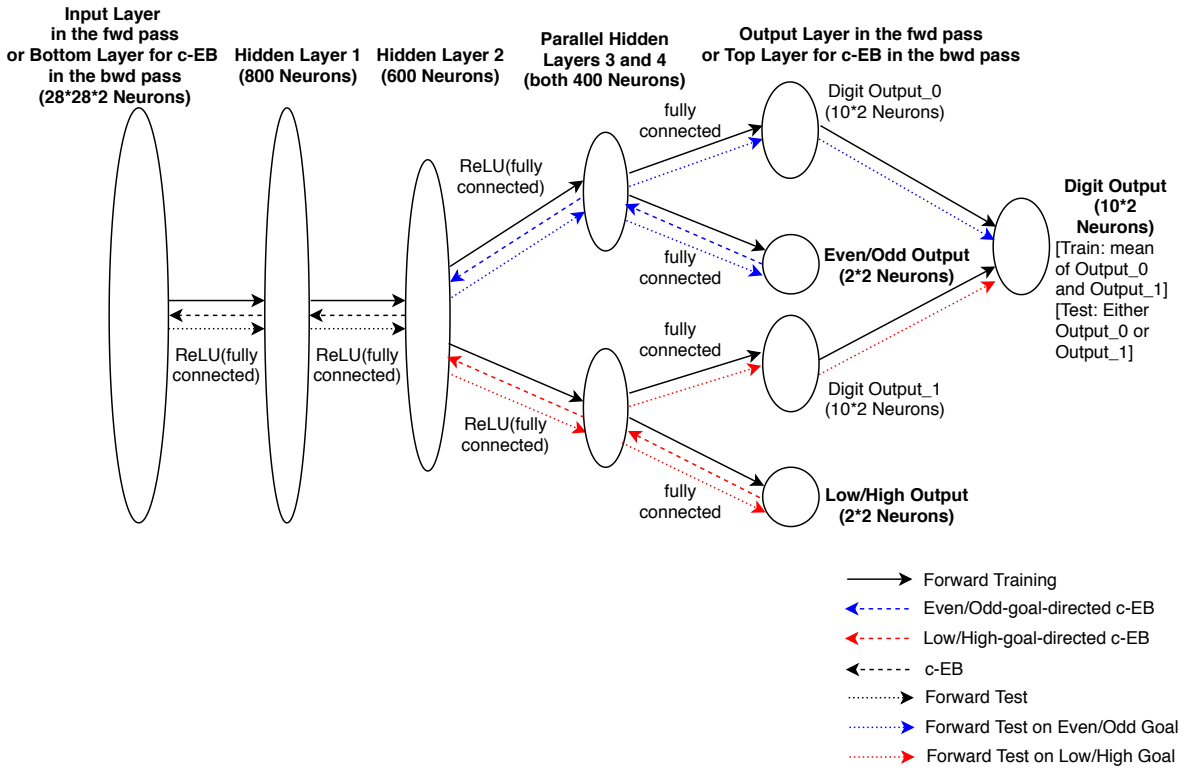


Figure 9: Network setup for our bottom-up and top-down processes.

which evaluated at $\boldsymbol{\theta} = \boldsymbol{\theta}_0$ is:

$$\begin{aligned} \frac{d^2\, SC_2^2(p_{\boldsymbol{\theta}_0}, p_{\boldsymbol{\theta}})}{d\boldsymbol{\theta}^2}|_{\boldsymbol{\theta}_0} &= 2\int_{\mathbb{S}^{d-1}}\int_{\mathbb{R}} \left(\frac{d\,\mathscr{R}q_{\boldsymbol{\theta}_0}(t,\boldsymbol{\xi})}{d\boldsymbol{\theta}}\right)\left(\frac{d\,\mathscr{R}q_{\boldsymbol{\theta}_0}(t,\boldsymbol{\xi})}{d\boldsymbol{\theta}}\right)^T dt d\boldsymbol{\xi} \\ &= 2\Gamma \end{aligned}$$

Finally, putting everything together we have:

$$SC_2^2(p_{\boldsymbol{\theta}_0}, p_{\boldsymbol{\theta}}) \approx \delta\boldsymbol{\theta}^T \Gamma \delta\boldsymbol{\theta}$$

1.3.3 Neuromodulated Attention

Network Architecture

We modified the c-EB neural network model to attend to different goals. Figure 9 shows our bottom-up classification process and our top-down attentional search process. In the forward pass, the input layer received a pair of 28×28-pixel noisy MNIST digits and thus had $28\times28\times2 = 1568$ neurons [147]. To test the network's ability to filter out distractions, noise that was randomly set between 0 and 0.7 was added to normalized pixel values (between 0 and 1) of the original MNIST digits. The final pixel values were then normalized again between 0 and 1.

Following the input layer were two sequential fully connected hidden layers with 800 and 600 neurons, respectively. Next, there were two parallel fully connected hidden layers, each with

400 neurons. All neurons in these layers implemented a Rectified Linear Unit (ReLU) as the activation function [148]. Each of the two parallel hidden layers led to the output in one goal class (parity/magnitude) along with the digit output. For each (left/right) side of the input image, after the third hidden layer, there were two parity (an even and an odd) output neurons and ten digit output neurons, which contributed to the parity/digit prediction using winner-take-all (WTA) on the activation probability of each parity/digit output neuron. Similarly the magnitude/digit prediction was obtained after the fourth hidden layer for each side of the input image. During training, the final digit output took the average of the digit output generated from the two parallel hidden layers. During testing, the final digit prediction was the digit output generated from one of the two parallel hidden layers, depending on the cued goal class.

In our top-down attentional search process (see backward arrows in Figure 9), one of the four goals (even, odd, low, and high) was selected, which excited the corresponding goal neuron for each of the two digits in the image and inhibited all the other goal neurons at the top layer of the backward pass (i.e., the output layer of the forward pass). The weights were backpropagated from the top layer to one of the parallel hidden layers below to excite the neurons corresponding to the goal (see dashed arrows from the top layer to parallel hidden layers 3 and 4 in Figure 9). Then the weights at the top layer were converted from excitatory to inhibitory in order to create a mask (note that the weights were originally non-negative). This inhibitory mask was used in an additional backpropagation from the top layer to the parallel hidden layer below corresponding to the goal. The result of a subtraction between the two backpropagations was a contrastive signal [57]. This contrastive signal was then used to perform regular EB over the remaining layers, which finally generated the probability of each given pixel in the input layer for exciting the cued goal neurons. In addition to exciting the goal neurons and inhibiting non-goal neurons, the contrastive signal canceled out common winner neurons. Such a contrastive extension of the backpropagation could effectively ignore noisy distractors and lead to more accurate attention focus on the goal [57].

Modification of c-EB

Excitation Backpropagation (EB) was developed as a goal-driven attentional framework for a Convolutional Neural Network (CNN) classifier based on a probabilistic WTA process [57]. It could visualize the features at each layer in the hierarchy that were relevant to a given output neuron. An important extension of EB was to have c-EB, which discriminated the goal pixels from distractors by cancelling out common winner neurons for different goals and amplifying discriminative neurons for the target goal [57].

The EB mechanism kept non-negative weights between activation neurons and used these excitatory connections to transmit top-down signals. The top-down relevance of a neuron a_n in the layer L_l was defined by its probability of being chosen as a layer-wise winner, which was called the Marginal Winning Probability (MWP) $P(a_n)$ [57]:

$$P(a_n) = \sum_{a_m \in (L_0, L_1, \ldots, L_{l-1})} \left(P(a_n|a_m) \cdot P(a_m) \right), \tag{30}$$

where a_m denoted each parent neuron in the preceding layer(s). The winner neurons were recursively

sampled in the top-down direction according to the conditional winning probability $P(a_n|a_m)$ [57]:

$$P(a_n|a_m) = \begin{cases} \frac{\widehat{a_n} \cdot w_{nm}}{\sum_{n:w_{nm}\geq 0} (\widehat{a_n} \cdot w_{nm})} & \text{if } w_{nm} \geq 0, \\ 0 & \text{otherwise,} \end{cases} \tag{31}$$

where w_{nm} was the weight between a parent neuron a_m and one child neuron a_n, and $\widehat{a_n}$ denoted a non-negative activation response.

For c-EB, the contrastive signals were transmitted in the top-down fashion to obtain highly discriminative attention maps (i.e., contrastive MWP maps) in the target layer (i.e., the bottom layer in Figure 9). Extending from Equation 30, [57] defined the contrastive MWP (c-MWP) of the target layer L_l as

$$A - \overline{A} = P_0 \cdot (P_1 - \overline{P_1}) \cdot P_1 \cdot \ldots \cdot P_{l-1}, \tag{32}$$

where A represented the MWP, $\overline{A}$ was the dual MWP for the contrastive units, P_0 was the signal from the guessed goal, $\overline{P_1}$ was the conditional probability of the inhibition mask from the top layer. The weights for the inhibition mask were the negation of the original weights from the top layer. Therefore, the threshold condition for $\overline{P_1}$ was the reverse of that for P_1 in Equation 31.

We extended the PyTorch [149] implementation of c-EB [150], whereas the original code for c-EB [57] was written in Caffe [151]. Different from their implementation, our network included different goal classes labeled for each of the two noisy MNIST digits. In addition, the c-EB was processed through one of the two parallel hidden layers immediately below the top layer in the backward pass of our network.

The system increased attention to the digit corresponding to the selected goal and decreased attention to the distractor digit. One goal class attended to the digit based on its parity (i.e., either odd or even), whereas the other goal class attended to the digit based on its magnitude (i.e., low values between 0 and 4 inclusively or high values between 5 and 9 inclusively). This resulted in two goals within each goal class. After supervised training on noisy pairs generated from the MNIST training dataset, c-EB was applied to the top-down attentional process on the test pairs and driven by one of the four goals to excite only the pixels relevant to the goal digit.

Figure 10 shows the two noisy test pairs and their c-EB generated attention maps according to each goal. c-EB driven by a goal went through the backward pass and excited the pixel neurons only related to the goal digit. On the irrelevant digit side, most pixel neurons were inhibited instead. Furthermore, background noise on both sides were ignored. Therefore, the goal digits and goal identity neurons could all be predicted correctly with high certainty in the end of the forward pass in these examples. However, even if two goal identities targeted the same goal digit, their highlighted pixels in c-EB visualization were not all the same. It is reasonable because our model had different output heads for the different goal classes.

With the first noisy test pair of "5" and "4" in Figure 10 as an example, if an even goal was selected, then both even goal neurons in the left and right digit sides were activated and all other goal neurons were inhibited in the top layer of the backward pass. After the c-EB process that sent contrastive

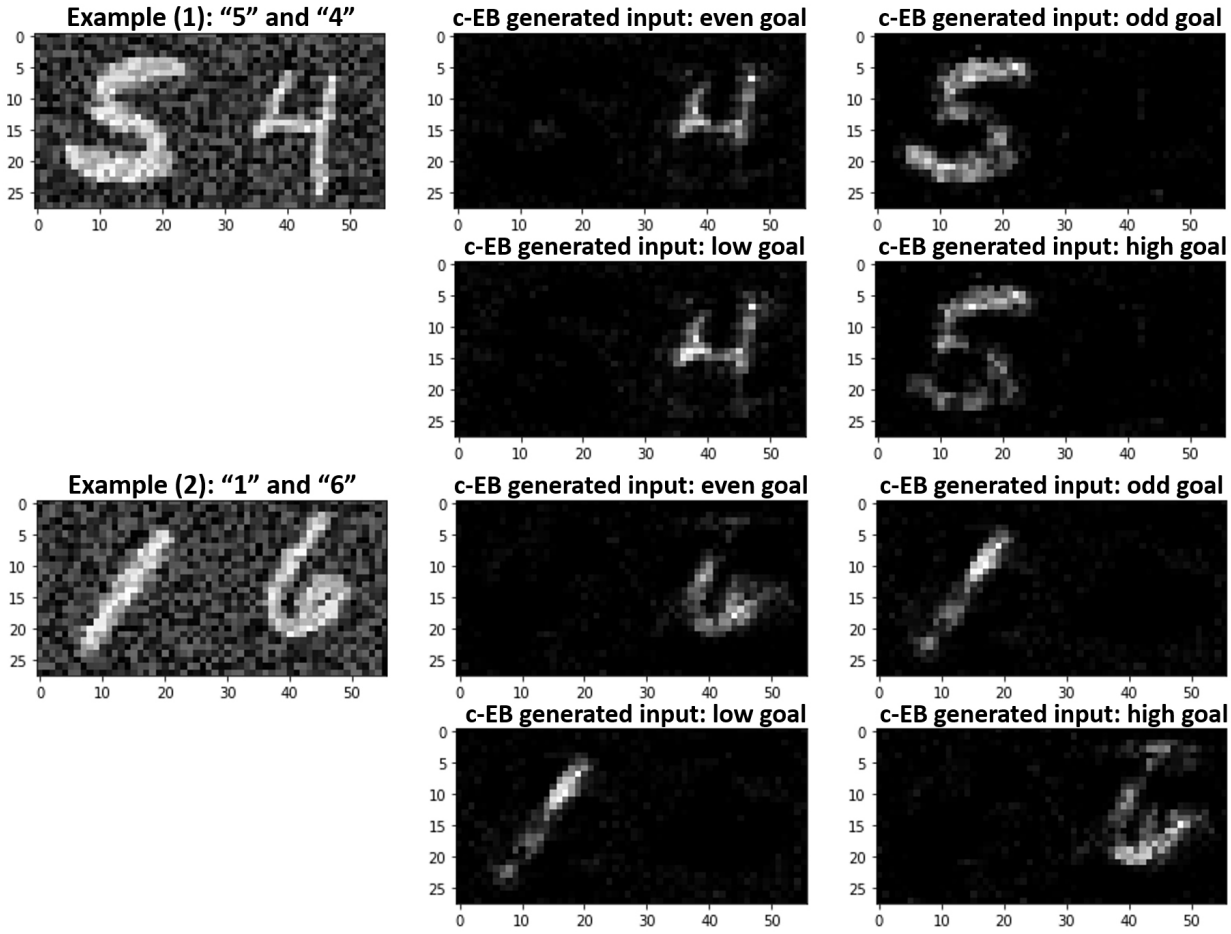


Figure 10: Two example test pairs of noisy MNIST digits and their c-EB highlighted results.

signals through the third hidden layer and added them up for normal EB via the second and first hidden layers to reach the bottom layer (Figure 9), pixels related to the digit “4” were highlighted, whereas pixels related to the distractor digit “5” and the background noise were inhibited.

Training and Testing Process

The training process consisted of incremental learning on noisy MNIST pairs. The original MNIST dataset was split into 60,000 digit images for training and 10,000 digit images for testing [147]. At each training step, 256 pairs of noisy MNIST digits were randomly selected and modified from the original MNIST training set. Every 200 training steps, 2,000 noisy MNIST pairs, randomly selected and modified from the original MNIST test set, were used for validation to evaluate the current training progress. The training process stopped after 4400 steps, when the validation accuracy was the highest. Therefore, a total of 1,126,400 noisy MNIST pairs were used for training. The following test process consisted of 10,000 pairs randomly selected and modified from the original MNIST test set. Given the sizes of the original MNIST training and test sets, there must be digit overlap within some training pairs or within some test pairs. However, there was no digit overlap between training and test. The two digits in each training pair could have the same or opposite parity (even/odd) and the same or opposite magnitude (low/high), whereas those in each test pair all

had the opposite parity and the opposite high and low values.

During training, a log-softmax function, followed by a negative log likelihood function, was applied after the forward pass to the neurons in the output layers that represented a digit, even parity, odd parity, low value, or high value. Then the sum of loss was used to calculate the gradient for each parameter in the model. At the end of each training step, a parameter update was performed based on the current gradient calculated using the Adam optimizer [105] with a learning rate of 0.001.

During testing, c-EB drove goal-driven perception by increasing the activity of input neurons corresponding to the goal digit and masking out the neurons corresponding to the distractor digit. In the top layer of our network (Figures 9 and 12), either two out of the four parity neurons or two out of the four magnitude neurons were activated, depending on the selected goal. For example, if an "odd" goal was selected, the odd neuron for the left digit and the odd neuron for the right digit were both excited, whereas all other goal neurons for both digits were inhibited. This resulted in c-EB in the backward pass to increase attention to this goal and its corresponding digit, and to ignore all other goals (see blue arrows for a parity goal and red arrows for a magnitude goal in Figure 9). Such c-EB generated input, with pixels highlighted for the goal digit (Figure 10), then went through the forward pass again in a way similar to the training process. However, according to the goal identity, only the parallel hidden layer related to the target goal class was used to predict the goal digit.

If a test pair had same parity or same high or low values (Figure 11), an existing goal would drive c-EB to highlight pixels from both digits. As expected, it shows that this ambiguous situation would cause confusion in the attention system. We assume it would cause confusion and random selection by a human faced with the same stimuli. Thus, we did not present same parity or same high or low values as test pairs. After this training and testing procedure, the parameters of the fully trained model were fixed for the neuromodulated goal-driven perception experiments.

Neuromodulated Goal-driven Perception

The overall neuromodulated procedure of goal-driven perception is shown in Figure 12. As described above, the network was trained with pairs of noisy MNIST digits to learn the digits and their parity (even/odd-value) and magnitude (low/high-value) goal classes. Then it was tested by selecting one of the even, odd, low-value, and high-value goals to trigger c-EB in the backward pass and generate an attention map, which further led to prediction of the digit and goal in the succeeding forward pass. After the robustness of c-EB prediction was verified, we applied ACh and NE neuromodulatory neurons to track the expected and unexpected uncertainties respectively and guess the goal for each trial. The guessed goal was applied to the top layer for c-EB as the intended goal for the current test pair. The guessed goal and predicted digit were compared with the true goal and true goal-related digit. The prediction was used to modify the neuromodulatory activities for the next trial.

ACh and NE Neuromodulation

For goal-driven perception, the network must select a goal when they are uncertain and unknown *a priori*. Similar to a model of the ACh and NE neuromodulatory systems proposed by [63], the goal target (even, odd, low, or high value) was rewarded with a probability (goal validity), but that goal would change periodically (goal identity). ACh neurons tracked *expected uncertainties* of the potential goals. NE neurons tracked *unexpected uncertainties*, and responded phasicially when a

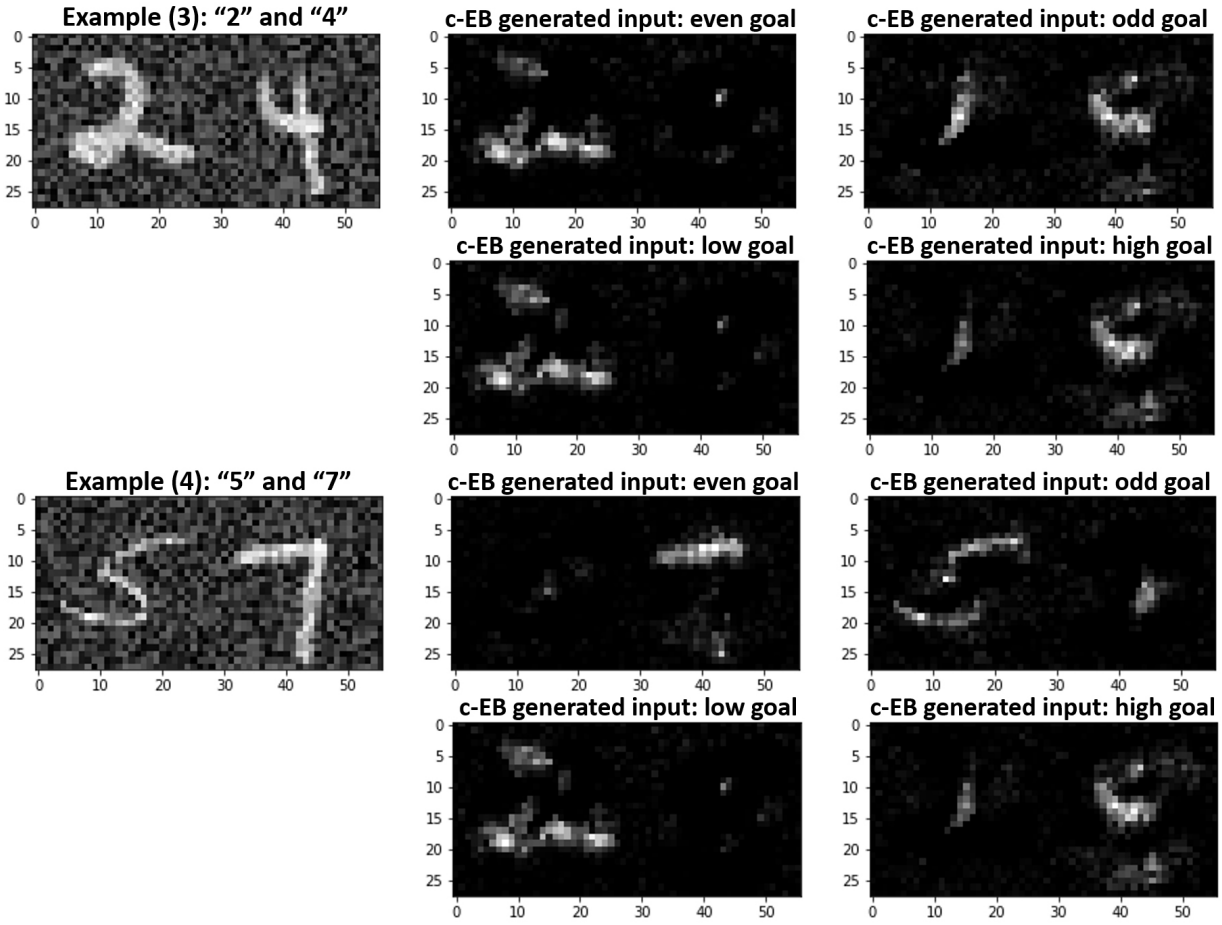


Figure 11: Two more example test pairs of noisy MNIST digits and their c-EB highlighted results.

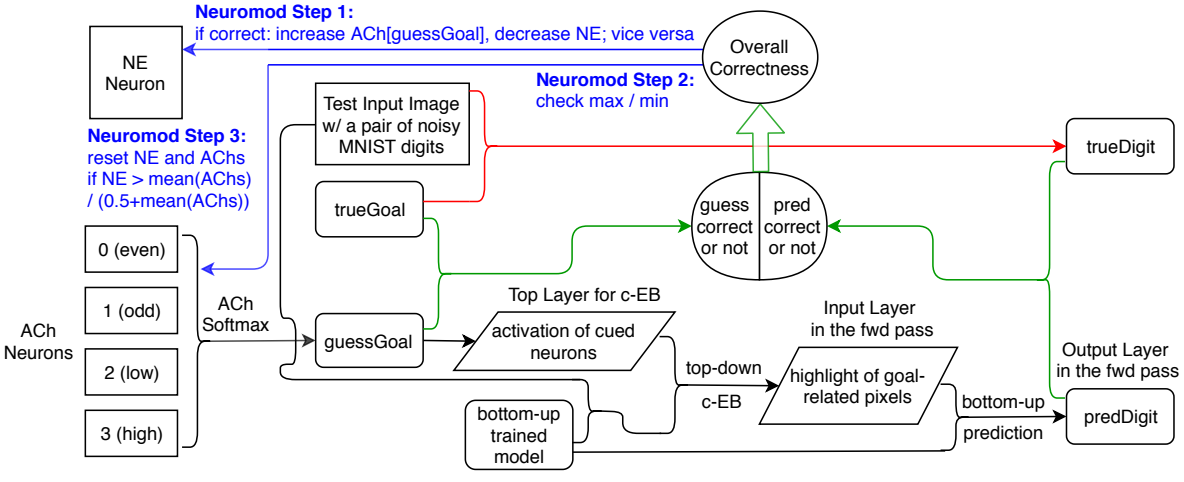


Figure 12: The neuromodulated procedure for the noisy MNIST-pair experiment.

goal identity change is detected. When the NE system responded phasically, it caused a network reset by re-initializing the ACh and NE neural activities, which allowed rapid adaptation under novel conditions.

There were $K = 4$ ACh neurons, each neuron corresponding to a goal (i.e., even, odd, low, or high value), and one NE neuron. One of the four attentional goal tasks was selected as the major goal for each goal switch. The true goal identity in each trial was set to either the major goal or the minor goal according to the goal validity. The true goal digit was obtained from the labels of the test pair by using the true goal identity. The activities of ACh neurons were input to a softmax function for goal selection:

$$p(goal)_i = \frac{\exp(\beta \cdot ACh_i)}{\sum_{j=1}^{K} \exp(\beta \cdot ACh_j)} \text{ for } i = 1, 2, ..., K, \tag{33}$$

where β is the temperature governing exploration versus exploitation and $p(goal)_i$ is the probability of selecting goal i. This guessed goal activated two neurons related to the goal in the top layer of our network architecture (Figures 9 and 12), which directed c-EB in the backward pass to activate the goal-relevant pixels in the test pair and then predicted the digit in the forward pass. If the prediction was correct (which means that the guessed goal identity matched the true goal identity and the predicted digit matched the true goal digit), the ACh level corresponding to the guessed goal (i.e., ACh_g) increased and the NE level decreased; the opposite would happen otherwise:

$$(ACh_g)_t = \begin{cases} \min(ch_{correct}(ACh_g)_{t-1}, ch^{max}) & \text{if correct,} \\ \max(ch_{wrong}(ACh_g)_{t-1}, ch^{min}) & \text{otherwise,} \end{cases} \tag{34}$$

$$NE_t = \begin{cases} \max(ne_{correct} \cdot NE_{t-1},\ ne^{min}) & \text{if correct,} \\ \min(ne_{wrong} \cdot NE_{t-1},\ ne^{max}) & \text{otherwise,} \end{cases} \tag{35}$$

where $[ch^{min}, ch^{max}]$ and $[ne^{min}, ne^{max}]$ were ranges for ACh and NE levels. $ch_{correct}$ and ne_{wrong} must be set within $[1.0, 2.0)$, and ch_{wrong} and $ne_{correct}$ must be within $(0, 1.0]$. If the NE level was above a threshold θ^{reset}, ACh and NE activities were reset to baseline levels [63]:

$$\theta^{reset} = \frac{\left(\sum_{i=1}^{K} ACh_i\right)/K}{0.5 + \left(\sum_{i=1}^{K} ACh_i\right)/K}. \tag{36}$$

While we used a particular set of parameters for the neuromodulation process, there was a wide range of parameter values that could be used to produce stable results. And while the randomness in the algorithm followed a uniform distribution, the selection of *guessGoal* required the softmax distribution.

Goal Selection

We added an online neuromodulatory model (Figure 12) to the head of the network architecture in the backward pass to regulate goal selection automatically. In these experiments, the goal (with goal identities of even, odd, low, or high value) had to be learned from experience. It might be noisy and rewarded with some probability (i.e., goal validity).

Automatic goal selection was tested in 10 runs to measure the average performance. In each run, one of the four attentional goal tasks was randomly selected as the major goal, which stayed the

Table 2: Relationship between goal actions and object labels.

Action	Role	Objects in Role
eat	obj	banana, apple, sandwich, orange, donut, carrot, broccoli, hot dog, pizza, cake
	instr	fork, knife, spoon, bowl, cup
work	instr	laptop, tv, mouse, keyboard
read	obj	book
	instr	laptop, cell phone
say-hi	obj	person

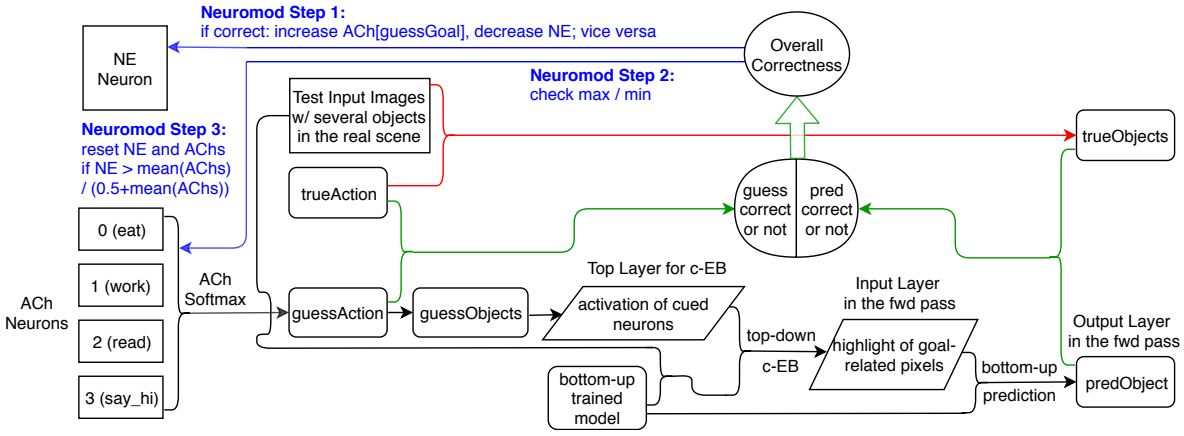


Figure 13: The neuromodulated procedure for the indoor robot experiment.

same every 400±30 trials for 10 switches. The minor goal identity came from the same goal class as the major goal identity. For example, if the major goal was "high", then the minor goal became "low" in the same magnitude goal class; or if the major goal was "even", then the minor goal became "odd" in the same parity magnitude class. The true goal identity was set to either the major goal or the minor goal randomly according to the validity distribution per trial. The true goal digit was obtained from the labels of the test pair of noisy MNIST digits using the true goal identity.

The goal validity values (i.e., 0.99, 0.85, and 0.70) were chosen to correspond with [63]. The major goal validity was randomly chosen among the three values each time the major goal identity got switched in a run. The minor goal validity was (1− major goal validity).

Action-Based Attention in a Robot Experiment

To test whether the goal-driven perception model could generalize to a more real-world application, we tested the model in an action-based attention task on the Toyota Human Support Robot (HSR). For the second experiment, we had four goal actions (i.e., "eat", "work-on-computer", "read", "say-hi") that were associated with images of objects seen by the HSR. Given a desired action, the task for the HSR was to guess the action and direct attention to the object in the scene that could achieve that action. For example, the action "eat" might result in attention to an "apple".

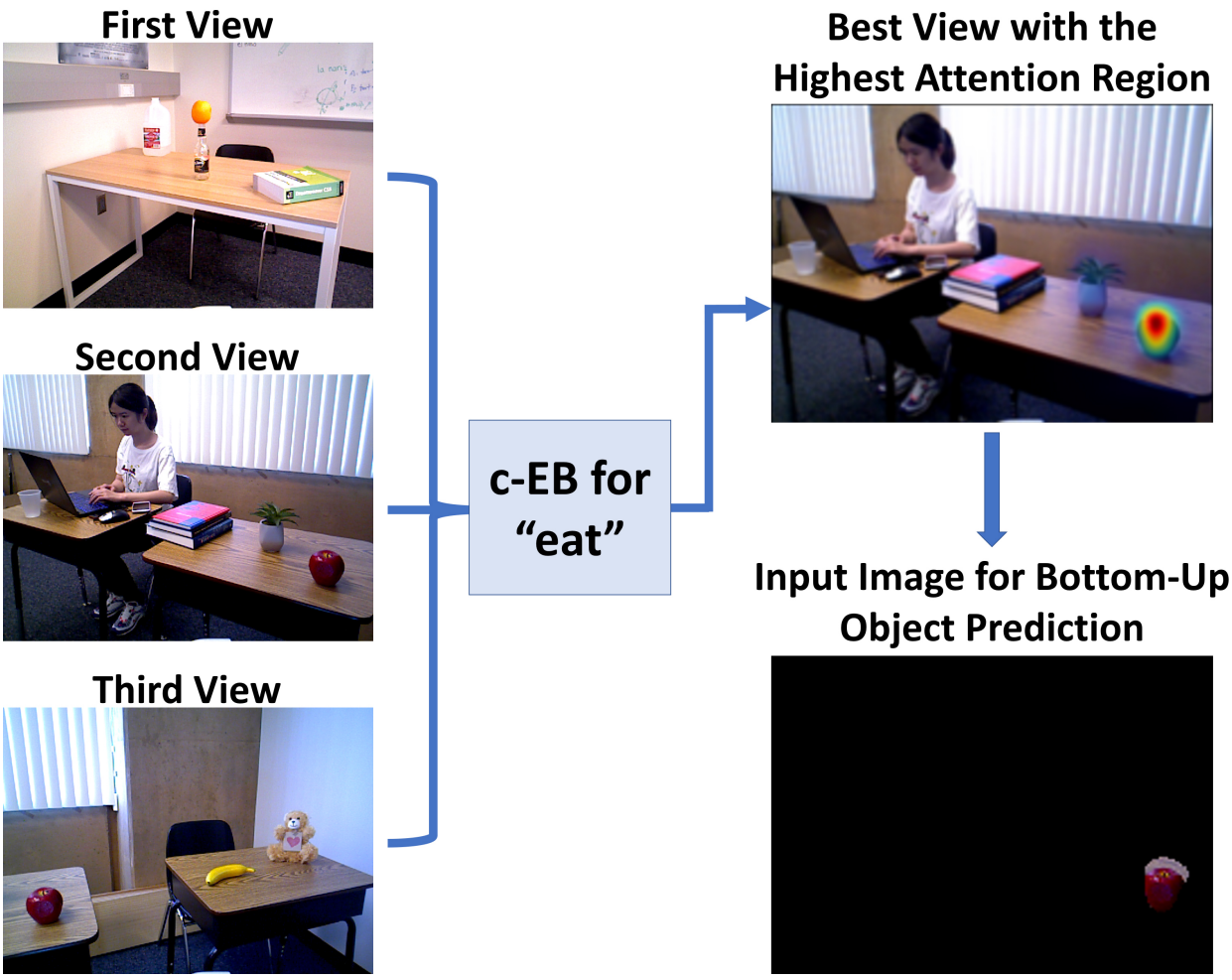


Figure 14: Depiction of the top-down attentional search process for a guessed action.

Figure 15: The test scenario for the indoor robot experiment.

For object classification, we used the Microsoft Common Objects in COntext (COCO) dataset [152] to train a GoogLeNet [153] via the Caffe framework [151] instead of the MNIST-pair network shown in Figure 9. An advantage of the COCO dataset was it used segmentation to localize individual object instances in the image, which was more accurate and more helpful for top-down attention than using bounding boxes.

For each run of this experiment, desired actions were randomly switched every 50 trials for 10 switches. In each trial, each image from three capture angles within an indoor scene was loaded as the input to the pretrained bottom-up model and went through a forward pass with the output layer specified as "loss3/classifier" in Caffe. The number of output prediction classes was set to 80, same as the number of object labels available for the COCO dataset [152]. Then the c-EB method was applied for the top-down attention process. As in the MNIST-pair experiment, the NE-ACh neuromodulation process with softmax on the ACh activities was applied for goal (action) selection (Figure 13).

Table 2 linked the guessed action with all related objects – from 80 COCO labels regardless of their semantic roles – which might or might not exist in the test scenario. Those activated object neurons in the top layer drove the c-EB through the second top layer "pool5/7x7_s1" and then normal EB to the bottom layer "pool3/3x3_s2" to generate attention maps related to this guessed goal action. The notation "pool5/7x7_s1" referred to a pooling layer with a kernel size of 7x7 and stride of 1. Only the highest attention region (with normalized attentional strength above the threshold of 0.1) corresponding to one of the three real captures maintained its original pixel values, whereas all other parts of the image became black (Figure 14). This attention-modulated image became the

Algorithm 1: World Model based Pseudo-Rehearsal

T set of potential tasks
initialize $\boldsymbol{V}$ model parameters
while $\mathcal{L}_{VAE}(V)$ *is decreasing* **do**
  $\boldsymbol{D_{all}} \leftarrow s \sim T(a_{rand})$
  $\boldsymbol{V} \leftarrow \nabla\mathcal{L}_{VAE}(V, D_{all})$
end
O $\leftarrow$ random training order over T
initialize $\boldsymbol{M}, \boldsymbol{C}$ model parameters
for i *in* O **do**
  $task_i, duration_i \leftarrow O(i)$
  for *n_episodes* **do**
    #collect training data
    $\boldsymbol{D_{real}} \sim task_i, C$
    if $i > 0$ **then** $\boldsymbol{D_{sim}} \sim M^*, C^*$
  end
  for $duration_i$ **do**
    #Mixture Density Network updates
    $\boldsymbol{M} \leftarrow \nabla\mathcal{L}_{MDN}(M, D_{real})$
    if $i > 0$ **then** $\boldsymbol{M} \leftarrow \nabla\mathcal{L}_{MDN}(M, D_{sim})$
  end
  for *n_steps* **do**
    #Reinforcement Learning
    $\boldsymbol{C} \leftarrow \nabla\mathcal{L}_{RL}(C, M(V(task_i)))$
    #Cross-Entropy Distillation
    if $i > 0$ **then** $\boldsymbol{C} \leftarrow \nabla\mathcal{L}_{CE}(C, D_{sim})$.
  end
  $\boldsymbol{M^*}, \boldsymbol{C^*} \leftarrow M, C$
end

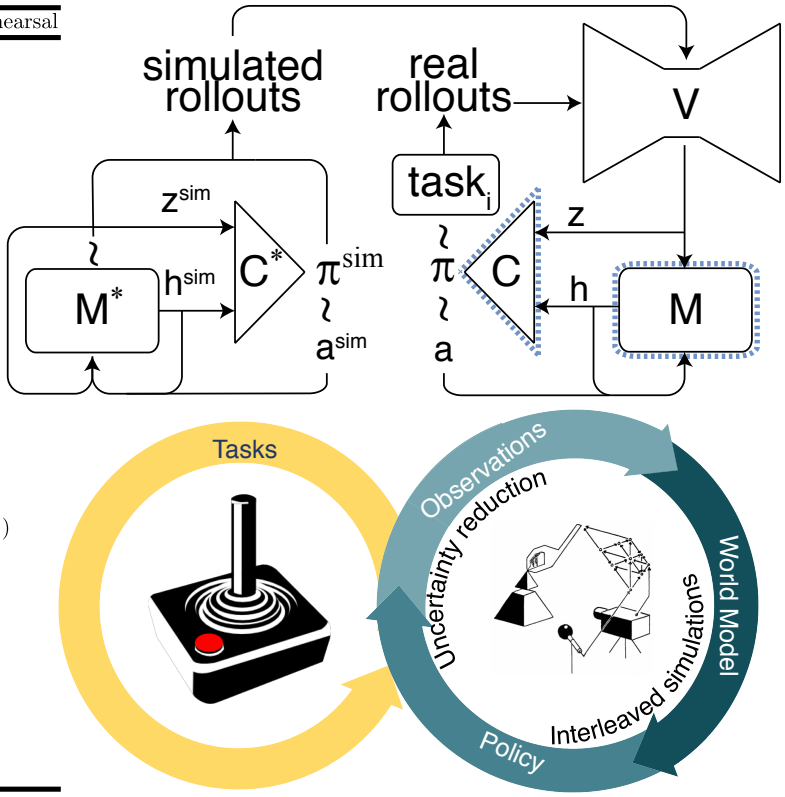


Figure 16: Depiction of lifelong learning facilitated by the World Model.

input to the forward pass of the network and generated object prediction via the top layer. Three conditions needed to be satisfied to generate an overall correct match for that trial: (1) the guessed action matched the true action; (2) the predicted object matched the real object in the scene; (3) the predicted object was associated with the guessed action.

The test environment for this experiment was a classroom scenario as shown in Figure 15. The test agent was a Toyota HSR [154]. During each trial, the HSR first guessed an action using the activity of the neuromodulatory neurons and linked it with objects using the semantic network. The HSR then moved from a starting point to the center of the testing scenario, where it captured three images from different view angles using the Red Green Blue-Depth (RGB-D) camera. After the attention network predicted an object as described above, the HSR moved towards the object and either picked up the object if it can be grabbed (e.g., apple) or pointed at the object with its arm (e.g., laptop). At the trial end, the HSR would present the object to a user who would respond with a "YES" if it the object matched his/her desired action or otherwise with a "NO". This feedback would be used by the neuromodulatory model to adjust the activity levels of both the NE neuron and the ACh neuron related to the current guessed action before guessing the action for the next trial.

1.3.4 Self-Preserving World Model

Our main focus in this work is testing the hypothesis that interleaving pseudo-rehearsal based replays can be sufficient to preserve the learned knowledge within a World Model framework. As

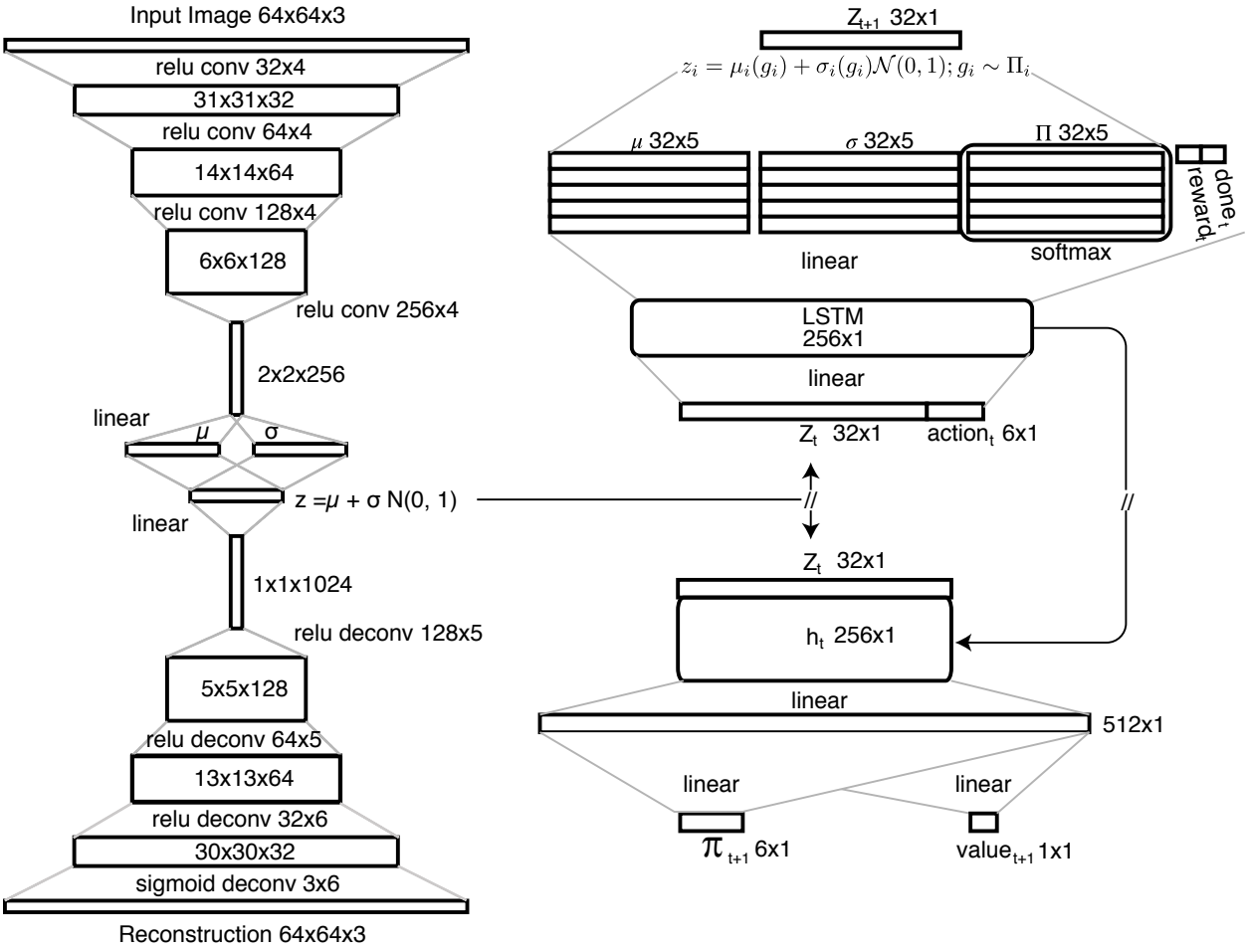


Figure 17: Schematic of the full network comprising VAE and the World Model.

illustrated in Figure 16, and similar to [67], a Variational Auto-Encoder (VAE, V) is trained to compress a high-dimensional input (e.g., images) into a smaller latent space (z) while also allowing for a reconstruction of that latent space back into the high-dimensional space. This latent space representation is then fed into a temporal prediction network (M) that is trained to predict one time step into the future. The output of these networks is then used as a latent state-space for a RL-based controller (c). An illustration of the network architecture used is shown in Figure 16 and more detailed in Figure 17. Sequentially learning a set of tasks is done by iteratively training the World Model M and the agent C. These neural networks are optimized based on samples from the current task interleaved with samples from simulated past experiences generated by preserved copies of those networks, M^* and C^*. Figure 17 shows the schematic of the full network. Input starts on the top left through the V network. Once a particular z vector is sampled, it is provided to the M network (top right) along with the current action to produce distribution of potential next states. The hidden state of the M along with the current z is provided as input to the controlling agent C (bottom right). This relatively simple network consists of a single linear layer of 512 units which splits into an actor and critic heads. Stops on the flow of gradients through the network are indicated by a '//' symbol. Here 'π_t' is an unnormalized vector of logits used to determine a distribution over potential actions, while '$action_t$' is a one-hot vector representing the particular action taken.

Model Architecture

The V network learns to both encode and reconstruct observed samples into a latent embedding by optimizing a combination of reconstruction error of the samples from the embedding back into the original observation space, and the KL-divergence of the samples from the prior ($\mathcal{N}(\mu = 0, \sigma = I)$) on the embedding space those samples are encoding into; this framework is generally known as a VAE [155]. We use a convolutional VAE with the same architecture as [67], where the input image is passed through four convolutional layers (32, 64, 128, and 256 filters, respectively) each with a 4x4 weight kernel and a stride of 2. Finally, the output of the convolution layers is passed through a fully connected linear layer onto a mean and standard deviation value for each of the dimensions of the latent space, which is used to then sample from that space. For reconstruction a set of deconvolution layers mirroring the convolution layers takes this latent representation as input and produces an output in the same dimensions as the original input. All activation functions are rectified linear except the last layer which uses a sigmoid to constrain the activation between 0 and 1.

The M network, also based on [67], takes the latent space observation and passes it through an LSTM layer [156]. The LSTM output is then concatenated with the current action as input to a Mixture Density Network (MDN) [157], which passes the input through a linear layer onto an output representation that is the mean (μ) and standard deviation (σ) used to determine a specific Normal distribution ($\mathcal{N}$), and a set of G mixture parameters (Π) used to weight those separate distributions in each of V's latent space dimensions. Two additionally output units are present, one for predicted reward and the other for the predicted episode termination probability. In total that is $3GL + 2$ output units, where L is the size of the latent space.

Finally, in departure from [67], a stochastic gradient-descent-based RL controller, C, was used for action selection. The C network takes as input the current hidden state (h) of the M network concatenated with the current latent vector (z). Internally there is a single fully connected 512 unit

hidden layer that splits off to the Actor and Critic heads. For learning we choose the A2C algorithm, which is the synchronous adaption of the original A3C algorithm [38], as we wanted to illustrate the effectiveness of our approach in a policy gradient based method, however, in theory any architecture would work.

World Model-Based Pseudo-Rehearsal

The training of these networks is usually done in sequence (V then M then C) and is entirely unsupervised (i.e., no labeled data is required). This framework allows for an offline simulation of experiences as the M network can predict one time step into the future, and can use its own predictions as the seed for the subsequent predictions provided some action derived from C. After sampling a specific prediction z^{sim} from the distribution of potential outcomes prescribed by the M network's output these samples feedback as input and are allowed to rollout for several time steps. These rollouts are used for pseudo-rehearsal, and are interleaved with the real samples from the environment.

The basic training algorithm for continual learning in this framework is outlined in Figure 16. Continual learning in the M and C networks is done by first learning/providing an auto-encoder, V, that can embed high-dimensional samples from all potential environments into a sufficiently low-dimensional space. If the input dimensions are already sufficiently small this embedding may potentially be unnecessary, however it is as yet unclear what a sufficiently low-dimensional space would be.

The first iteration of training starts by sampling (with replacement) some particular task and duration to train on. Data collection is done in this task using a random action selection policy and rollouts of $[[z_t, a_t, r_t, d_t]_{Tmax}]_N$ are saved. Here for a given times-step t, z_t is the latent representation of the current observation, a_t is the chosen action, r_t is the observed reward and d_t is the binary done state of the episode. For each task exposure, N rollouts (1,000 in this work) are collected, where each rollout is allowed to proceed until d_t is 1 or it reaches the $Tmax$ (1,000 in this work) maximum number of recorded time steps. The M network is then optimized to perform 1-time step prediction on these rollouts. Following this, the C network is trained to produce an action distribution π such that sampled actions maximize the expected reward on the same task that M was just trained on to. This C network uses the latent embedding of the current observation (z_t) and the current hidden state (h_t) of the now trained M network as input. The C network is consistently trained for `n_steps` (1e6 in this work) within the current task. This initial training is meant to generally mimic the Car Racing experiment of [67]. At the end of each iteration of continual learning the current M and C networks are preserved as M^* and C^*.

At the start of the next iteration, a new task and duration are sampled, and the two networks are used to generate pseudo-samples to be interleaved with real samples from incoming new tasks. First, a new set of real rollouts are generated using the current task processed through M^* and C^*. Here, the M^* network is provided a V encoded observation from the current task which is in turn passed on to C^* to produced a particular action, again yielding rollouts of the form: $[[z_t, a_t, r_t, d_t]_{Tmax}]_N$. Then, the simulated rollout generation process starts by picking a random point in the latent space here sampled based on the prior of the VAE (i.e., $\mathcal{N}(0, I)$), along with a zeroed out hidden state and a randomly sampled action. Feeding this through M^* produces the first simulated observation

(z_0^{sim} and hidden state h_0^{sim} which are in turn provided to C^* to yield the first distribution of potential actions π_t^{sim} and the particular action sampled from that distribution a_0^{sim}. Using the last sample as input to the M^* network this process continues and the $[z_t^{sim}, a_t^{sim}, r_t^{sim}, d_t^{sim}, \pi_t^{sim}]$ tuples are stacked in time to produce simulated rollouts of pseudo-samples.

These rollouts are in effect simulations of the tasks the network has already been exposed to, and can then be interleaved with new experiences to preserve the performance with respect to previous tasks in both the M and C networks. Here, pseudo-rehearsal updates in M are exactly the same as from real samples, just using the simulated rollouts in place of real rollouts. Updates in the C network are done using policy distillation, using a cross-entropy loss function with a temperature τ [88]. Specifically, provided a given simulated sample z_t^{sim} as input, the temperature modulated softmax of C's output distribution ($softmax(\pi_t/\tau)$) is forced to be similar to the temperature modulated softmax of the simulated output distribution ($softmax(\pi_t^{sim}/\tau)$) from C^*. Based on [88] a constant τ of 0.01 was used in our experiments.

Experiments

Testing of continual learning in the M and C networks was done by first generating 1,000 rollouts from all potential tasks, which in this case were a set of three Atari games. Each random rollout was generated using a series of randomly sampled actions with a probability of 0.5 that a the last action will repeat. These rollouts were constrained to have a minimum duration of 100 and maximum duration of 1,000 samples. The first 900 of these rollouts, for each game, were used for training data and the last 100 are reserved for testing. All image observations were reduced to 64x64x3 and rescaled from 0 to 1, and all games were limited to a six-dimensional action space: "NOOP", "FIRE", "UP", "RIGHT", "LEFT" and "DOWN". Each game is run through the Arcade Learning Environment (ALE) and interfaced through the OpenAI Gym [158, 159]. All rewards were clipped as either -1, 0, or 1 based on the sign of the reward, the terminal states were labeled in reference to the ALE game-over signal, and a non-stochastic frame-skipping value of 4 was used. The same environment parameters are used through-out the experiment.

All training images are then fully interleaved to train a VAE (V network) that can encode into and decode out of a 32 dimensional latent space. Training was done using a batch size of 32 and allowed to continue until 30 epochs of 100,000 samples showed no decrease in test loss greater than 10^{-4}.

Using this pre-trained V network to encode the original rollouts into the latent space, the M network is then trained over a series of randomly determined task exposures. First, a random training order is determined such that all tasks have the same exposure to training, which is in total 30 epochs per task. This total is split over the course of three randomly determined training intervals which each has a minimum of three epochs and a maximum determined by the floor of the total epochs left divided by the number of training exposures left for a given task. The order over task exposures is then randomized with the exception that the first task and training duration (which has no pseudo-rehearsal) is always the same across random replications. Each epoch of training in the M network is done using rollouts of length 32 in 100 batches of 16. Once M training is finished for a given task exposure, the output of this trained network is then used as input to the C network for the same task. In contrast to the random training duration of the M network, training in the C network was consistently set to one million frames per task exposure.

After every task exposure, the M and C networks are preserved as M^* and C^* as shown in Figure 16. This pair of networks is then used to generate a set of 1,000 simulated rollouts, or pseudo-samples. For efficiency, these rollouts were saved into RAM at the start of each task exposure, but could easily be generated on-demand. These generated rollouts are then interleaved with the next task's training set. Similarly, a set of 1,000 real rollouts from the next task are generated using M^* and C^*, mimicking the training scheme of [160].

Then on the next task exposure, the M network is updated with one simulated rollout to one real rollout for the duration determined by the current task exposure. Similarly, after training M, the C network is allowed to explore the current task, however for every 30,000 frames from the current task, a batch of 30,000 simulated frames is trained using policy distillation. C training continues in each task exposure until 1e6 frames from the real task have been seen.

The average loss per output unit in the M network was used to asses performance, and median reward over 30,000 frames was used in the C network. A baseline measure of catastrophic forgetting was established by performing the same training as described above with no pseudo-samples interleaved. Here, the area under the performance metric curve is integrated over all training epochs and divided by the sum over the two experimental conditions (training with and without pseudo-rehearsal) to achieve a percent performance that sums to one within each task. Performance statistics were calculated over 10 replications where a new random task exposure order was sampled for each replication. Little to no hyperparameter tuning was done in the M or C networks. Here we were trying to replicate the M and V from [67] as close as possible, and were interested in C as a form of performance metric rather than pursuing a SotA level of performance.

M Network Optimization

A more detailed description of the M optimization method (Equations 37-39) is provided for extra clarity beyond the description in [67]. All symbols and indexes match those shown in Figure 17. Optimization of the LSTM and MDN is done comparing the output of the MDN to a V encoded real observation (z_{t+1}) from one time step into the future predicated on the chosen action (a_t). Specifically, the combined LSTM (with hidden state h_t) and MDN, referred to as the M network, models each dimension (indexed by i) of the next latent state as the weighted sum of G Gaussians:

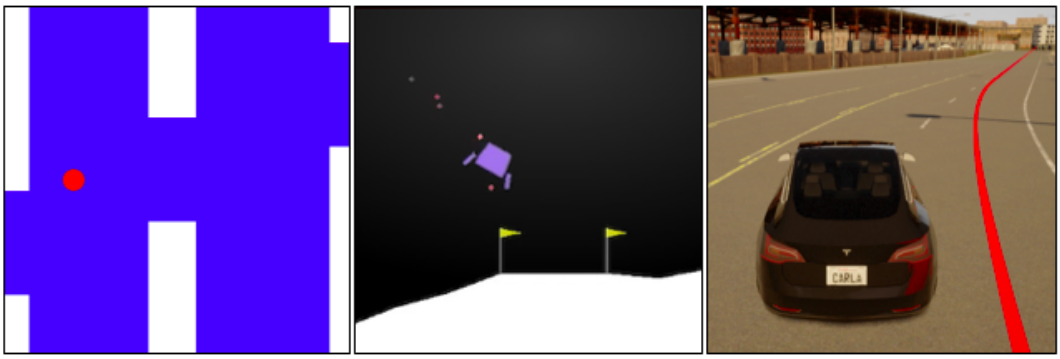
$$P_i(z_{t+1}|x=(z_t,a_t,h_t)) = \sum_g^G \Pi_g(x)\mathcal{N}(\mu_g(x),\sigma_g(x)). \tag{37}$$

Output units corresponding to each Π_i set are normalized using a softmax, and each $\sigma_{g,i}$ unit is passed through an exponential function to ensure they are appropriate for the mixture model. This set of Gaussian mixture models (GMMs) can be optimized using the average log likelihood over latent dimensions as the loss function:

$$\mathscr{L}_{GMM}(z_{t+1},x) = 1/L\sum_i^L -\log(\sum_g^G \Pi_{g,i}\mathcal{N}(\mu_{g,i}(x),\sigma_{g,i}(x))). \tag{38}$$

Additionally, reward and terminal state units are optimized using mean squared error ($\mathscr{L}_{MSE}$) and binary cross-entropy ($\mathscr{L}_{BCE}$), respectively. Finally, the total loss optimized by the M network is the average of the three losses:

$$1/3(\mathscr{L}_{MSE}+\mathscr{L}_{BCE}+\mathscr{L}_{GMM}). \tag{39}$$



(a) Flappy Ball (b) Lunar Lander (c) CARLA

Figure 18: Scenes from Flappy Ball (a), Lunar Lander (b), and CARLA (c) domains.

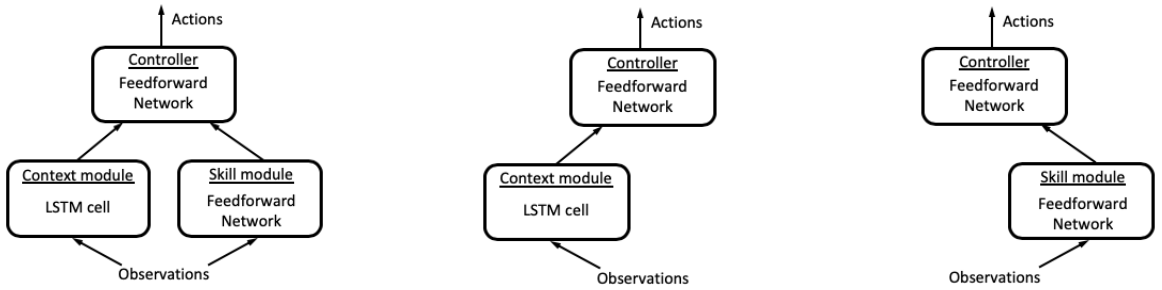
1.3.5 Context-Skill Model

Experimental Domains

Three different environments are used for the experiments. The first one, Flappy Ball (FB; Figure 18(a)), is an extension of the popular Flappy Bird computer game [161]. The red circle in the FB domain represents the agent and the white columns are pipes that move from right to left as the game progresses. The agent, controlled by a neural network, aims to navigate through the openings between pipes without hitting them for a certain length of time. The agent can flap forward and flap upward; gravity will pull it down and drag will slow it down. The agent gets a reward of +1 every time it passes a pipe successfully, and a penalty of -1 at each time step it crashes into the pipes and -5 when it crashes in the ceiling or ground. At every time step, the agent receives six-dimensional sensory information: vertical position, horizontal and vertical velocities, horizontal distance to the right edge of the closest pipe, and the height of the top and bottom pipes, normalized to [0,1]. The effects of flap upward and forward as well as gravity and drag can change between episodes; Therefore, the agent has to infer such variations from its interactions with the environment over time.

The second domain is LunarLander-v2 (Figure 18(b)) from OpenAI Gym suite, modified to allow variations in mass of the spacecraft and the effect of the main and two side engine thrusters. The purple object in the LL domain represents the spacecraft that controls its main and side engines to land safely on the white surface designated with flags. As in the original LL domain, the agent receives eight-dimensional numerical sensory information as its input (i.e., position and velocity of the lander in 2D, its angle and angular velocity, and whether each leg touches the ground). The purpose is to land on the lunar surface in a designated region indicated with the flags safely and in minimal time. The episode finishes if the lander crashes or comes to rest.

The third domain is the CARLA open-source autonomous driving simulation environment (Figure 18(c)). The agent has both lateral (steering) and longitudinal (throttle) control of the car while driving between two points in a given certain amount of time. The steering and the torque curves are modified to evaluate generalization. Furthermore the agent is placed in completely unseen tracks as it tries to complete the same task albeit under more difficult driving conditions (as imposed by the modified steering and torque curves). As in the other environments, the agent receives



(a) Context-Skill Model (b) Context-only Network (c) Skill-only Network

Figure 19: The architecture of the Context-Skill Model and its ablations.

numerical sensory information as five numerical values: Rangefinder coordinates which describe the distance between the agent and the lane boundaries along five axes. All control actions are given as continuous values varying within [-1,1]. The objective is to minimize distance from the lane center, wobbliness, and final distance from a target zone.

These domains can be seen as proxy for control and decision-making problems where the changes in the environment require immediate adaptation, such as operating a vehicle under different weather conditions, configuration changes, wear and tear, or sensor malfunctions. The challenge is to adapt the existing policies to the new conditions immediately without further training, i.e., to generalize the known behavior to unseen situations.

The core idea evaluated in this work is to implement the agent as a Context-Skill network (CS) that takes advantage of an explicit representation of context. The Context-Skill Model consists of three components: the Skill and the Context modules and the Controller (Figure 19(a)). The first two modules receive sensory information from the environment as numerical values, as described above. They send their output to the Controller, a fully connected feedforward neural network that makes the decisions on which actions to take. The Skill module processes the current situation, the Context module integrates observations over the entire task, and the Controller combines the outputs of both modules, thereby using context to modulate actions. This architecture is compared to (b) context-only ablation, and (c) skill only ablation in the experiments. Each component is found to play an important role, allowing the CS network to generalize better than its ablations.

The Skill module is also a fully connected feedforward network. Together with the Controller they form the Skill-only Network (S; Figure 19(c)). The Skill module used in this study has 10 hidden and five output nodes and the Controller has 20 hidden n odes. S is used as the baseline model throughout the study. In principle it has all the information for navigating through the pipes, but does not have the benefit of explicit representation of context.

The other main component in the Context-Skill Model is the Context module. It is composed of a vanilla Long Short-Term Memory (LSTM) cell [156]. There are three gates in this recurrent memory cell: input, forget, and output. The gates are responsible for learning what to store, what to throw away, and what to read out from the long-term memory of the cell. Thus, the cell can learn to retain information from the past, update it, and output it at an appropriate time, thereby making it

possible to learn sequential behavior [162, 163].

The Context module used in this study consists of an LSTM cell size of 10. The memory of the Context module (h_{t-1} and c_{t-1}) is reset at the beginning of each new task, and accumulated (transferred) across episodes within each task. It can therefore form a representation of how actions affect the environment. The output of the LSTM (h_t) is sent to Controller as the context. Together the Context module and the Controller form the Context-only network (C; Figure 19(b)). It serves as a second baseline, allowing integration of observations over time, but without a specific Skill network to map them directly to action recommendations.

The complete Context-Skill Model, CS (Figure 19(a)) consists of both the Context and Skill modules as well as the Controller network of the same size as in C and S. The motivation behind the CS architecture, i.e., of integrating the Context module with S, is to make it possible for the system to learn to use an explicit context representation to modulate its actions appropriately. The method for discovering these behaviors is discussed next.

Evolutionary Learning Process

The goal in each domain is to find a safe solution that optimizes the performance objective. Although it is possible to formulate the optimization process based on that single objective, it turns out the diversity resulting from the multi-objective search speeds up training and helps discover well-performing solutions [164]. In the experiments, the first solution that reaches a satisfactory level in the safety objective is returned as the final result of evolution, and its generalization ability evaluated.

In each domain, the first objective, f_0, measures safety, and the second, f_1, measures performance, and is in conflict with safety. Accordingly in the FB domain, the number of any type of collisions (or hits) is minimized, whereas the number of successfully passed pipes is maximized. In the LL domain, the total rewards (indicating a successful landing) is maximized, whereas the landing time is minimized. In the CARLA domain, safety f_0 is optimized by minimizing a cumulative weighted sum of a lane-distance penalty and a wobbliness penalty:

$$f_0 = \sum_{t=1}^{t=T} (d(t) + \lambda \text{abs}(s(t) - s(t-1))), \tag{40}$$

where $d(t)$ is the distance from the center of the lane at time t (shown as a red line in Figure 18(c)), $s(t)$ is the steering output at time t, and λ is a proportionality coefficient ($= 5.5$ in the experiments below). Time is incremented in 1/30 sec intervals; each episode lasts $T = 20$ seconds, or less if the agent reaches a specified target zone (identifying the end of the track) before that. P erformance f_1 is optimized by minimizing the Euclidean distance between the agent and the target zone at the end of the episode.

In the evolutionary learning process, the weights of all three neural networks described above are evolved while the network architecture remains fixed. The goal is to maximize the average fitness across multiple tasks, where each task is based on different physical parameters in the FB, LL, and CARLA domains. The FB domain has four parameters (Flap, Fwd, Gravity, Drag), the LL domain

Table 3: Parameter ranges during training of the Context-Skill Model.

FB (±20%)			LL (±10%)			CARLA (±15%)		
$\text{Flap}_{\text{base}}$	=	-12.0	$\text{Main}_{\text{base}}$	=	20.0	α_{base}	=	1.0
Fwd_{base}	=	5.0	$\text{Side}_{\text{base}}$	=	1.0	β_{base}	=	1.0
$\text{Gravity}_{\text{base}}$	=	1.0	$\text{Mass}_{\text{base}}$	=	8.0			
$\text{Drag}_{\text{base}}$	=	1.0						

has three parameters (Main, Side, Mass) and the CARLA-domain has two parameters (α and β, which control the steering angle and torque curves of the car, respectively). In each task during evolution, only one parameter is subject to change, while the rest are fixed at their base values (given in Table 3). Thus, the number of tasks is equal to the number of parameters. The parameters in each domain are varied during training within $\pm 20\%$, $\pm 10\%$ and $\pm 15\%$ in the FB, LL, and CARLA domains, respectively.

Each task, and therefore each parameter, is uniformly sampled n_{episodes}=5 (except in the CARLA domain where each parameter is sampled six times) times within the limits specified above. Thus, there are 20 fitness evaluations per individual in each generation in the FB domain, 15 evaluations in the LL domain and 12 evaluations in the CARLA-domain. The fitness of every individual in the population, i.e., average score of all episodes, is evaluated in parallel on the same task distribution for a fair comparison. The memory of Context module in CS and C is reset at the beginning of each task, and transferred from episode to episode otherwise.

Non-dominated Sorting Genetic Algorithm (NSGA-II) [165] was implemented in the DEAP evolutionary computation framework [166] as the optimization method. The overall procedure is shown in Figure 20. It receives n_{tasks}, n_{episodes}=5, perturb=0.2 in FB, 0.1 (in LL), and n_{episodes}=12, perturb=0.15 (in CARLA), base parameter values, μ = 96 (48 in CARLA), $p_{\text{crossover}}$ = 0.9, n_{gen} = 2,500 as input. The population size is chosen as a multiple of 24 since the fitness evaluations are distributed among 24 threads on a cluster (i.e., Dell PowerEdge M710, 2× Xeon X5675, six cores @ 3.06GHz). Default NSGA-II settings, including Simulated Binary Crossover, Polynomial Mutation, Tournament Selection Based on Dominance, and ($\mu + \lambda$) elitist selection strategy, were used [165]. The process is robust to minor variations of these choices.

1.3.6 Reflexive Adaptation

The overall approach that was taken to reflexive adaptation was to consider a monitoring-and-intervention strategy. Thus, the reflex module continually monitors the output of the agent's controller and evaluates the actions to determine whether or not they pose a threat to the safety of the agent or the surroundings. If it is found that the actions the agent would take in the immediate future could lead to a catastrophic event (i.e., death or destruction), then an emergency situation is declared and the reflex module works to find a suitable sequence of actions to address this particular situation and circumvent the danger. Upon finding an appropriate reflexive response the module takes-over the output of the controller and the agent is made to apply the reflexive actions for the duration of the action-sequence devised. Upon completing this action-sequence, control is returned to the controller and the reflex module reverts to its monitoring phase.

Algorithm 1 Evolutionary process for training networks

```
procedure EVOLVE
    stop := False
    parents := random_init_individuals(μ)
    task_params = prepare_task_params(n_episodes,n_tasks)
     fitness = distribute(eval_fitness(), (parents, task_params))
    for gen from 1 to n_gen do
        offspring = tournament_sel_DCD(parents, μ)
        for i from 1 to λ do
            if random() ≤ p_crossover then
                SBX(offspring[i], offspring[i+1])
            Polynomial_Mutation(offspring[i])
            Polynomial_Mutation(offspring[i+1])
        params = prepare_task_params(n_episodes,n_tasks)
         fitness = distribute(eval_fitness(), (parents, task_params))
        for j from 1 to λ do
            if fitness[j][0] ≥ pipes_max then
                if fitness[j][1] ≤ hits_max then
                    stop := True
         parents := tournament_sel_DCD(parents + offspring, μ)
        if stop == True then
            return parents
            break
```

Figure 20: Algorithm for training the Context-Skill Model.

In order to realize this strategy, the reflex module was broken down into the following three components, which constitute the three sub-problems that had to be addressed: (1) Emergency Prediction; (2) Reflex Generation; and (3) Reflex Retrieval.

Figure 21 outlines the workflow within the reflex module and the role that each component plays. The first sub-problem looks at prediction into the future to detect whether an emergency situation is imminent given the agent's current policy. The concept of a World Model was used here, which captures the dynamics of a given environment in order to predict the results of actions taken on that environment at different states. With each step that the agent takes using its current controller, the reflex module can use such a model (as well as a copy of the current controller) to determine what subsequent actions the agent will take and what the impact will be on the environment given those actions. This can be done for a few time steps into the future (e.g., 10 or 20) to gauge the impact of the agent's current policy. If an emergency is predicted to occur, Reflex Generation deals with the sub-problem of finding a suitable action-sequence to circumvent it. This is an online optimization that takes place to devise a customized response to the particular emergency situation predicted. Finally, Reflex Retrieval looks at the problem of matching a given emergency situation prediction to the closest one in a possible database that is maintained. While not strictly necessary, having such an archive of recorded emergencies, and the best solutions found to address them, could help to expedite the online search for an optimal reflexive response.

Setup and Preliminary Investigations

Computing Infrastructure

For the purposes of experimentation and testing, we used the classic Atari 2600 game benchmarks.

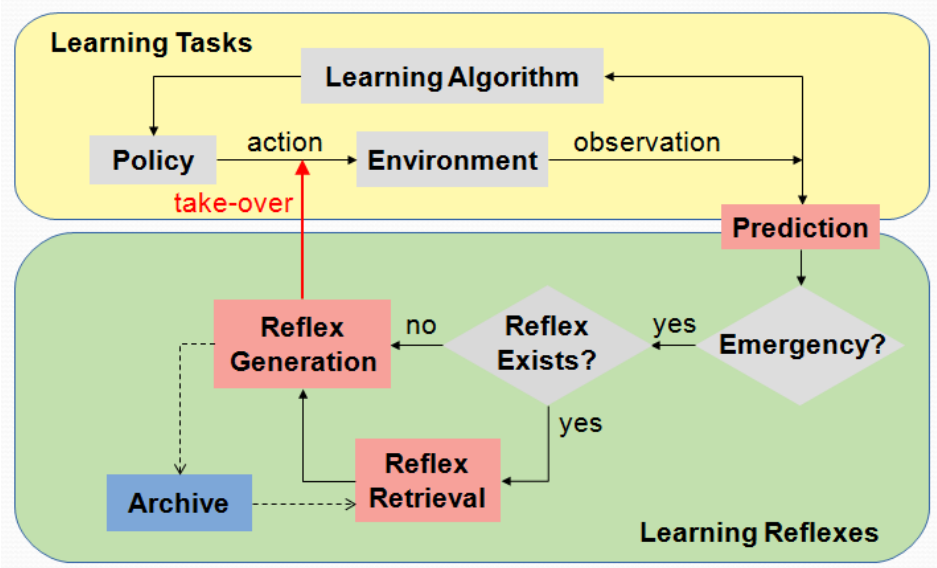


Figure 21: Overall workflow of the reflex module.

We implemented the Centroidal Voronoi Tessellation-Multi-dimensional Archive of Phenotypic Elites (CVT-MAP-Elites) algorithm [167] in Python 3. This evolutionary algorithm is designed to find thousands of diverse but high-performing s olutions. It can be used to design repertoires [168, 169] that allow robots to adapt online in only a few trials [170]. The CVT-MAP-Elites algorithm first divides the behavior space into niches of equal size with a CVT computed with Lloyd's algorithm. It then enters a variation-selection loop in which: (1) a random niche is selected among the non-void niches; (2) a variation (mutation) is applied to the elite of the niche; (3) the performance of the new candidate solution is evaluated; and (4) the behavior of the new candidate solution is used to determine the niche to which it will belong: if it performs better than the current elite of the niche, it replaces it; otherwise, the new candidate is discarded.

Good results were obtained with this algorithm in previous work [167, 168, 169, 170]. Due to its possible extension and application to rapid adaptation for L2, effort was made to produce a working version of it within the computing infrastructure adopted herein. The Python implementation of CVT-MAP-Elites was parallelized using Python's "multiprocessing" library. The algorithm was tested on the 10-dimensional Rastrigin function, where the two behavioral dimensions (which define the niches) were the first two genotypic dimensions. Figure 22 shows a typical result (the yellow and blue regions indicate high- and low-performing solutions, respectively).

Investigating Concepts for Emergency Detection

While the first sub-problem that was finally established was prediction, initial investigations also looked at other concepts for detecting an imminent death event. The first a pproach t hat was considered was to develop a database of potential "death scenarios", as well as a corresponding sequence of "best" actions that have been found to successfully address those scenarios. Both these knowledge-bases can be built online through observation of game-play during the policy training sessions.

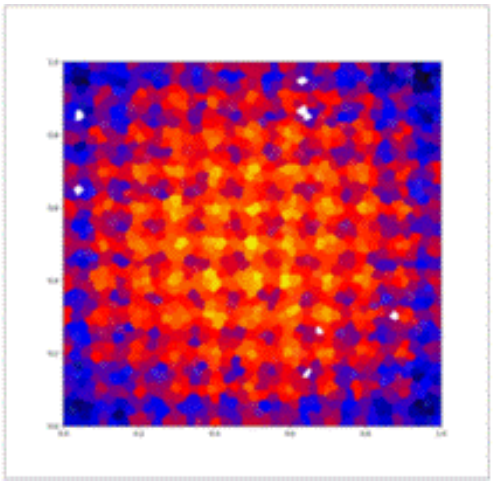


Figure 22: Typical output of CVT-MAP-Elites on the 10-dimensional Rastrigin function.

Within the context of Atari games, moments before instances in which the main player loses a life are taken to be emergency situations that need to be reliably detected in advance in order to be circumvented through an appropriate reflex. The database of death scenarios thus consists of a set of observation frames from the game, where each frame is the first in a set of $N = 10$ frames before an observed death event.

The Space Invaders Atari game environment was chosen as the first test case on which to train a neural network policy, as it was felt to present a reasonable challenge in which to observe the potential benefits that reflexes can provide. Moreover, the sample-efficient Actor-Critic with Experience Replay (ACER) algorithm was chosen as the training algorithm, as it was found to perform reasonably well with different games, and could quickly and easily be applied using the Stable Baselines implementation (https://stable-baselines.readthedocs.io/en/master/), making it a good starting point for our investigations. In any case, since the concept of reflexes is general, the corresponding module, once fully-developed, can be tested on a variety of game environments and learning algorithms. The ACER training algorithm was run on this game environment until 10 death events were observed. Each such event had a corresponding 10-frame history, and the first frame from each history formed the 10-image death-scenarios database.

Detection via Image Pixel Errors

To begin with, a simple image-matching approach was implemented to detect the death scenarios. In particular, the observation obtained from each step of a new training session was compared to the images in the database to find a match. The first tests that were done focused solely on detecting death scenarios, without taking any action.

In order to establish a baseline for comparison, basic pixel-by-pixel error based methods were tried, namely, Mean Squared Error (MSE) and Structural Similarity Index Metric (SSIM). Upon obtaining a 10-image database of death scenarios, a new training session with ACER was run until a total of 50 death events occurred. With each step of training, the two pixel error based methods were used to evaluate whether a match could be found between the observation output at each step and any

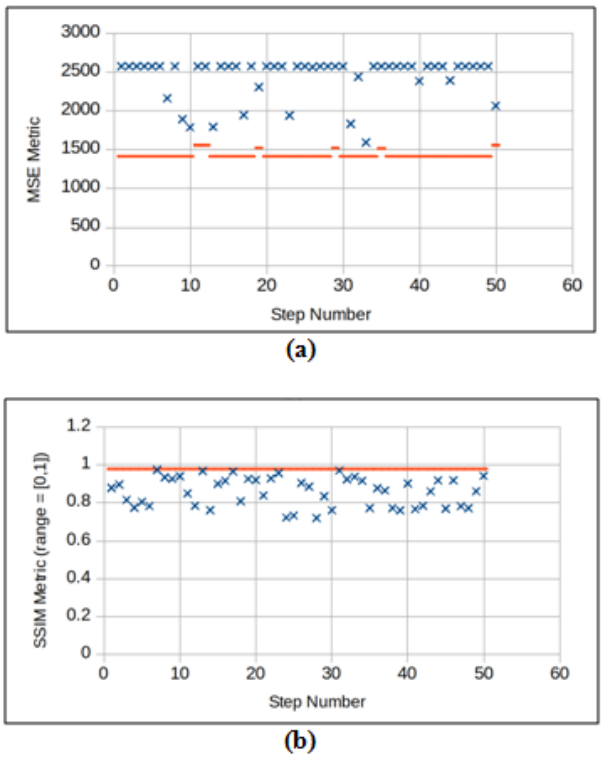


Figure 23: False-negative results for detecting death events.

of the images in the database in order to test how well they could detect imminent death scenarios requiring reflexes. Results showed that both metrics were unable to detect any of the 50 deaths that occurred during game-play. Figure 23 shows the values of the two metrics (blue 'X's) in relation to their corresponding thresholds (red '-'s) for each of these 50 false-negative cases (MSE: (a), SSIM: (b)).

For the MSE metric, the errors represent the difference in corresponding pixel intensity values. The square of the mean of all errors is compared to the square of the maximum allowable error at any given pixel. This maximum allowable is computed as the mean of the maximum percentage of the intensity value at each pixel that is allowed as deviation, and is calculated separately for each image in the death-scenarios database. This percentage value is a parameter that can be varied. For the results in Figure 22, it was set to 0.45. The SSIM metric, on the other hand, ranges in value from 0 to 1, and represents percentage similarity between the two images compared. Hence, the threshold represents the minimum percentage similarity below which the images are considered dissimilar, and was set to 0.98.

Since none of the 50 actual death events were caught, the positive matches that did occur for the two metrics were all false-cases, as shown in Figure 24 (MSE: (a), SSIM: (b)). While increasing the MSE metric threshold (or, correspondingly, decreasing the SSIM metric threshold) could have helped to remove some of the false negatives and catch more death events, it would have come at the cost of a significant increase in false positive detections as well, since it was observed that there

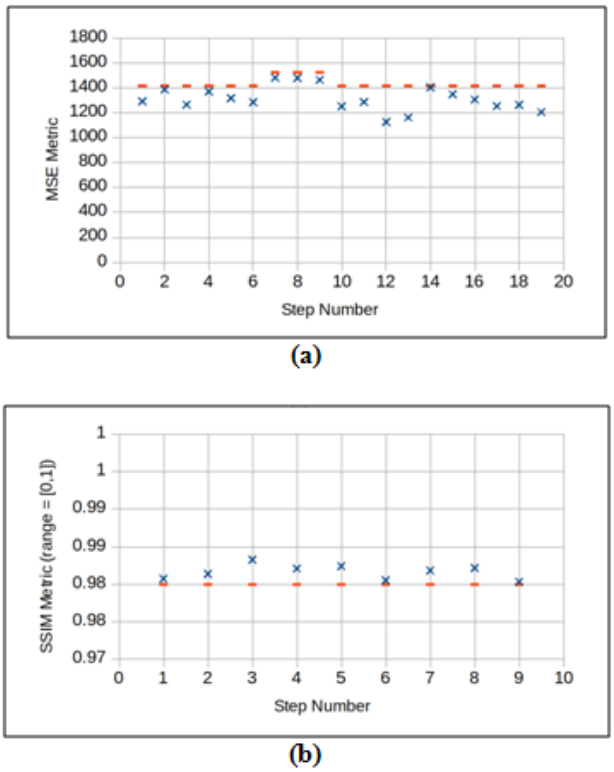


Figure 24: False-positive results for detecting death events.

were many successful negative matches that were already quite close to the threshold.

These preliminary tests showed that these simple pixel-error based methods perform quite poorly, and are not able to accurately and reliably detect the death scenarios, at least with only a 10-image database. To be sure, these tests were repeated with a larger database (100 death-scenario images). However, no significant improvement in detection success rate was observed, establishing that such comparison methods, although attractive in their computational simplicity, are not viable, and an alternative, more intelligent method of death-detection is certainly required.

Detection via Latent Representations

A second approach to detecting potential death events in Atari games explored the use of auto-encoder neural networks for image-matching with observations in the database of death scenarios. This approach was investigated using a 3-layer auto-encoder that converted a 1 x 7056 vectorized representation of a given observation (an 84x84 grayscale image) from an Atari game into a compressed, latent representation in the form of a 1 x 289 vector. Each of the 100 observations in the death-scenarios database were also compressed using the same auto-encoder, and "matching" with death-scenarios was accomplished by computing the MSE between these latent representations of observations and death scenarios.

Simulations conducted using this approach involved playing the Space Invaders Atari game until 50 death events occurred. At each death-event, the encoded representation of the observation obtained

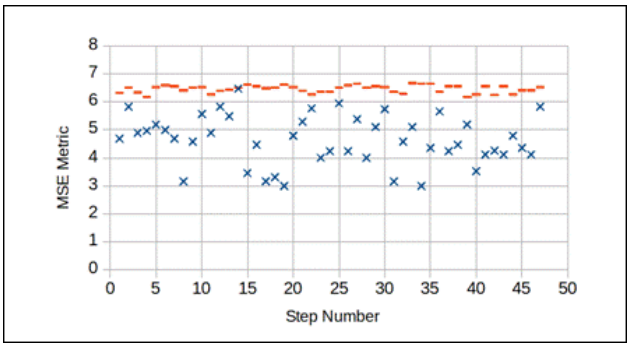


Figure 25: Successful death-event detections using the MSE metric on latent representations.

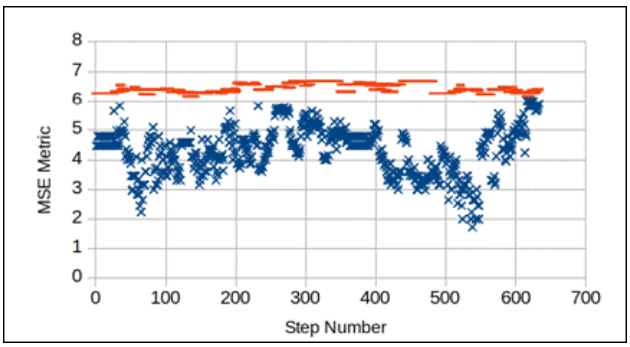


Figure 26: False-positive results for the MSE metric used on latent representations.

10-frames prior was compared with that of each element of the death-scenarios database to see if a match could be found. The same MSE metric maximum allowable deviation percentage parameter (as described earlier) of 0.45 was used. The results showed a significant improvement in success rate (94% in particular), as shown by the plot of the 47 successful matches in Figure 25.

Nevertheless, a significant number of false positives were also found to occur (Figure 26), resulting in an "over-reactive" agent that very often incorrectly predicts imminent deaths requiring a reflex. Still, the use of auto-encoders here shows some promise and is worth investigating further, within the context of a more intelligent method, as discussed below in the third approach to death-detection that is being explored.

Detection via Model-Based Prediction

A third approach to death-detection that was explored involved the use of the multi-scale memory model developed within our project. This model includes a fast-learning component, analogous to the human hippocampus, which uses memory of recent experiences, as well as a slow-learning component, analogous to the human neo-cortex, that gradually incorporates new information with previous experiences. Together they form a temporal prediction system that could potentially be used to identify imminent death scenarios in Atari game environments.

This memory model was inspired by, and combines the benefits of, earlier work done in [76] on developing World Models and involving simulated replays to help in single-task learning, and in [171] involving deep generative replay where multiple task-specific models are integrated during

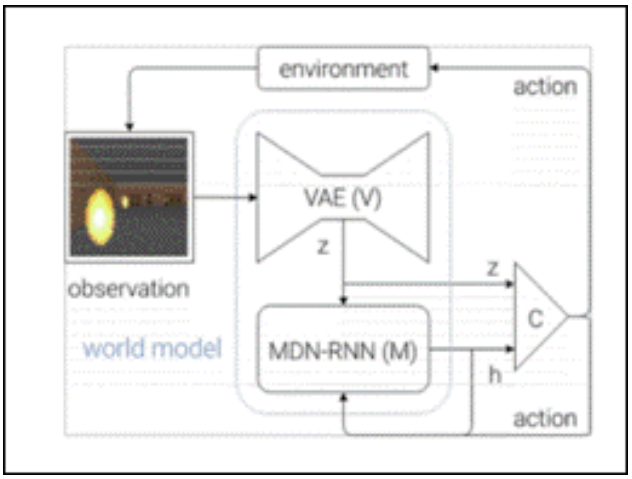


Figure 27: Schematic representation of the agent model.

offline "consolidation" to train the agent to best perform them all as each one is introduced, thereby facilitating continual learning as new tasks arise. Figure 27 shows a schematic flowchart diagram of the structure of this memory model as it relates to the work in [76] to train an optimal controller to act in a given game environment. Observations from the environment are once again transformed into a latent space (z) through an auto-encoder, and the memory model uses this latent representation to predict the observation one time step into the future.

Although Figure 27 only shows the hidden state (h) that is output to the controller (C), the Mixture Density Recurrent Neural Network (MDRNN) that constitutes the World Model is trained to model the dynamics of the game-play and can be used to output a prediction of the next state (z_{t+1}) given the action that will be performed at the current state. By feeding this predicted output back into the RNN, it is possible to predict multiple observations (frames) into the future.

Before conducting the simulations to test the death-detection performance of this memory-model-based prediction approach, some modifications to the Atari OpenAI Gym environments had to be made. In particular, the Atari games that were used for these tests register a death only after a fixed death-animation is played out. Moreover, all lives that the player starts with must be lost before the agent is considered to have died. This poses a problem for training the memory-model in predicting death events because it is imperative that the registration of a death occurs precisely when the agent actually dies, and not several frames after (death animations can be long), in order for the memory model to predict deaths early enough to allow time for a reflex action to be implemented. In addition, it would take significantly longer to train the model properly if deaths are only registered after all lives are lost, which could require significantly long game-plays to achieve. As such, the game environments must be made to register death after each loss of life.

The original memory model was developed and trained to help in developing a controller that allowed an agent to learn how to behave in a game environment, particularly when the environment could change significantly, requiring learning how to perform new tasks. Therefore, it was necessary to modify and re-train the memory model, and to modify the game environment that was used as

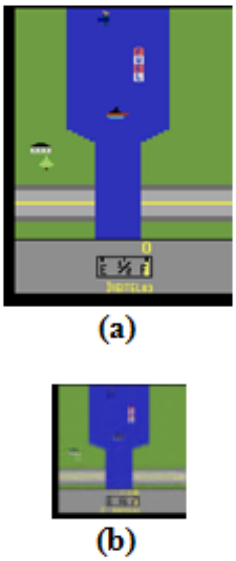


Figure 28: Comparison of (a) true and (b) predicted observations with a well-trained memory model.

well. In particular, the Atari game environment had to be placed within an external "single-life-episode" wrapper object that allowed deaths to be registered after each life-loss. Secondly, the training algorithm for the memory model was modified to appropriately change the value of the "done" variable that is output with each step of the simulated rollout that the model performs. When generating the training data, the true point of death is identified by tracing back the appropriate number of time steps from when the death was registered to just before the death animation (a fixed number of frames) is played out. The value of the "done" variable is then modified, post-data generation, accordingly.

With these modifications to the game environment and the training algorithm, the model was re-trained on the River Raid Atari game using parameters found to work well from earlier training sessions. However, upon carrying out the test simulations, it was found that the memory model was not able to make accurate predictions. Prior to the above-described modifications, Figure 28 below shows the predicted output from the memory model compared to the corresponding observation from the true environment (note that the predicted output is down-sampled compared to the true observation). Hence, this established the level of accuracy that one can reasonably expect.

The re-trained model, however, was found to produce poor results, as shown by an example output in Figure 29. Without the ability to accurately predict observations during a simulated rollout performed by the memory model, it is not possible to test its performance in death-detection. Hence, further effort was required to determine why the memory model performed poorly and how it can be improved.

With a working model, the procedure envisioned is to use it to perform an N-step simulated rollout with each step taken in the true environment. The model would first be initialized with the current observation from the true environment. The predicted next frame can then be fed back into the model to obtain the subsequent observation prediction. With the game environment and training

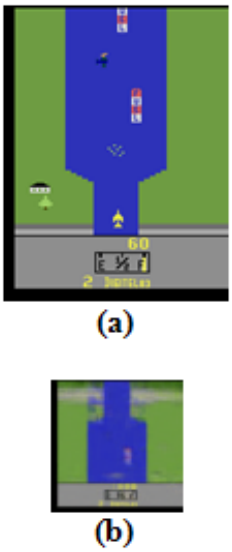


Figure 29: Comparison of (a) true and (b) predicted observations with the current memory model.

algorithm modified appropriately the simulated rollout should be able to predict a death event if it were to occur within the next N frames using the actions given by the agent being used. Given the fact that uncertainty in the predictions will grow with each recursive application of this memory model, one cannot expect to make accurate long-term predictions. However, for the purposes of predicting only in the short-term future (i.e., around $N = 10$ frames into the future), using the model in this manner seems feasible.

If a death is predicted to occur in any of these N simulated frames, it would be compared to what actually unfolds from carrying on with the true environment rollout. In this way, data on death-prediction performance in terms of true/false-positives and true/false-negatives can then be obtained to compare with the previous approaches.

Reflex Search via Random Sampling

A key question that needed to be answered first before moving forward with developing the concept of reflexes was whether we can expect such reflexes to have a significant impact on an agent's performance, both during training and after. In order to get an indication of the potential impact of reflexes, and to provide a sort of "proof-of-concept", a set of simulation experiments were devised.

Two Atari game-playing agents were created, one that received no training (i.e., a "random" agent whose parameters were those obtained through random initialization) and another trained with 2,000,000 time steps of game-play (i.e., a "trained" agent). Each agent was then allowed to play the game it was trained on for some M steps under two scenarios: with the use of reflexes, and without.

Usage of reflexes was incorporated into the simulations in the following m anner. At each time step of game-play of the main environment, another instance of the game environment was created and brought to the same state as the main one. This secondary environment took the place of a prediction module and was used to "play" the game forward N steps into the future (using actions

obtained from querying the agent playing the game). If a death was found to occur during these N steps, an alternative action-sequence (i.e., the reflex) was found and implemented for the next N steps in the main environment.

To find the reflex to implement in these situations, a set of R, N-step action sequences were randomly generated and then played out in the secondary game environment while recording the cumulative reward and the outcome at the end of each sequence. The action-sequence that did not result in death and which gave the greatest increase in cumulative reward was selected as the reflex to implement in the main environment.

For each of the two scenarios described above (i.e., game-play experiment with and without reflexes), the final cumulative reward and the total number of deaths that occurred were recorded at the end of the T total time steps of game-play. This experiment was repeated a total of 10 times under each scenario, and the distribution of cumulative reward and number of deaths from these 10 experiments were compared between the two scenarios through the use of box plots. This allowed for the analysis of the degree of impact that the incorporation of reflexes can have on the performance of an autonomous agent. The following parameters were used for these experiments:

$N = 20$: each reflex was a 20-step action-sequence.

$R = 200$: a sample set of 200 randomly-generated reflexes were obtained from which to select the best reflex to implement each time a death-event was predicted.

$T = 1{,}000$: each experiment under both scenarios involved playing the game for a total of 1,000 time steps.

The experiments under each scenario were repeated a total of 10 times.

The experiments under the two scenarios for each of the two agents (i.e., random agent and trained agent) were repeated for four Atari 2600 games: Space Invaders, River Raid, Chopper Command, and James Bond. The box plots comparing the impact of reflexes between these two scenarios for the different agent types and Atari games are given in Figures 30 to 33 below.

As the box-plots in Figures 30 to 33 show, reflexes indeed have a significant positive impact on an autonomous agent's performance. They are most beneficial for the random agent with respect to cumulative reward, which comes as no surprise since a random agent would tend to die more frequently due to poor choices. This confirms our earlier hypothesis that reflexes would have most benefit during the early stages of an autonomous agent's training when it still does not know much about how to function effectively within its environment. Once it has learned its task well, the impact of reflexes decreases significantly. Nevertheless, they may still be useful when the agent is exposed to new environments or situations for which it needs to be trained from scratch. In such cases, the simple reflex maneuvers might still apply and be effective enough to keep the agent alive as it begins to learn new tasks.

It is also important to note that the amount of improvement varies with the four games tested, as can be seen in Figure 31, and while the bulk of the distributions do show some overlap (at least in two of the games), there is still improvement in median cumulative reward. However, it should

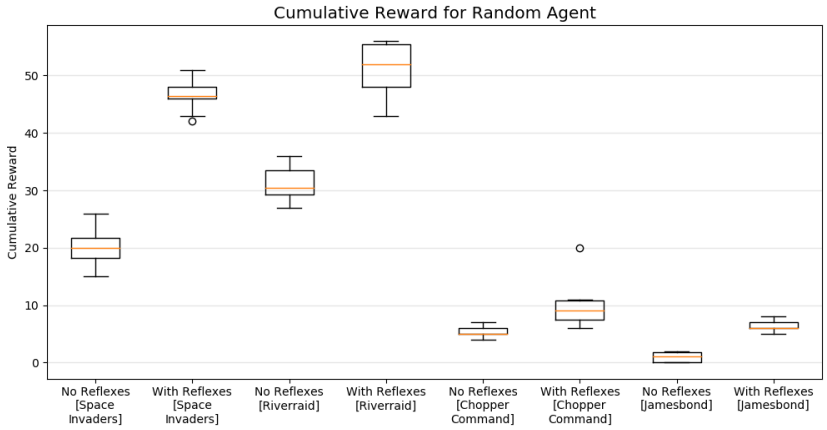


Figure 30: Impact of reflexes on cumulative reward for a random agent.

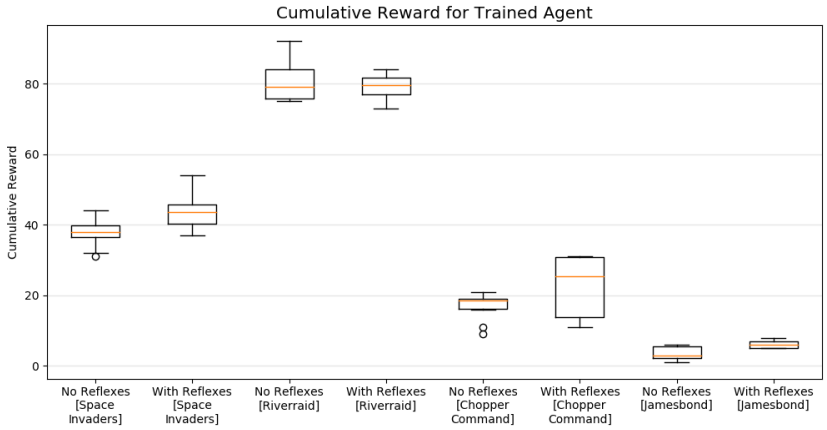


Figure 31: Impact of reflexes on cumulative reward for a trained agent.

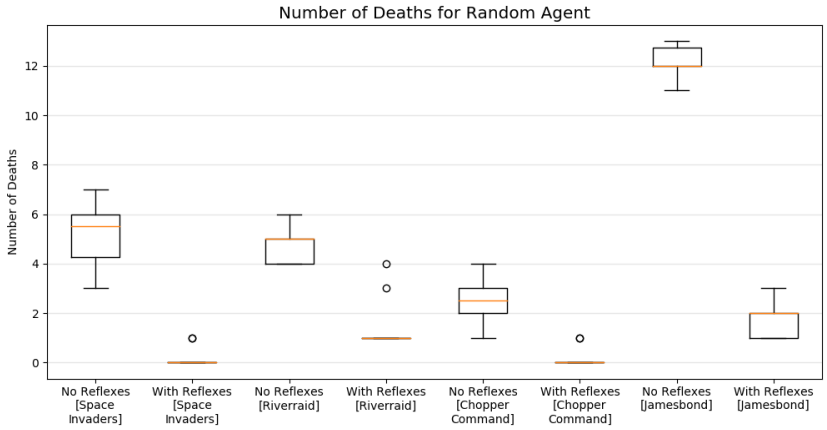


Figure 32: Impact of reflexes on the number of deaths for a random agent.

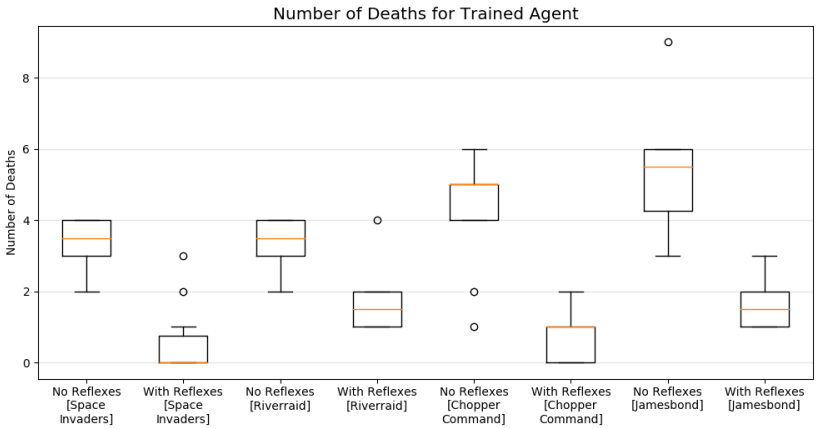


Figure 33: Impact of reflexes on the number of deaths for a trained agent.

also be noted that the agents for the four games were trained to the same number of time steps, but not necessarily the same level of performance or proficiency among the four games. This will also impact the relative benefit that each agent sees when reflexes are used.

Emergency Prediction

Emergency prediction was done using a system based on our enhanced version of the World Model. It involved using a VAE combined with a MDRNN, modeled after the work done on World Models [171]. This model was able to capture the dynamics of a given game environment in order to predict the next state into the future for a given action. By feeding the predicted state back into the model, multi-step future predictions can be made, albeit with decreasing accuracy. Nevertheless, for very short time-span predictions, this can still be a viable approach to testing the impact of a given policy into the near future.

Earlier issues in accuracy of predicted observation images (Figure 29) were successfully addressed by increasing the dimensionality of the latent representation to a 1×64 vector, and increasing the number of training rollout examples to 1,000,000. This allowed for the output observation frames to be visually comparable to the expected output. However, accurately associating a predicted observation with the correct value of the Boolean output variable that indicates whether or not a loss of life has occurred was found to be poor. In other words, while observation frames could be predicted reasonably well, the model could not accurately predict whether a death-event had occurred. Investigations into why this was occurring, and modifications to help improve prediction accuracy, were conducted.

In these investigations, focus was placed on the prediction of the terminal ("done") values of an OpenAI Gym environment, for which three associated measures were defined: (1) Accuracy: representing the proportion of all terminal values (i.e., done=0 and done=1 cases) that were predicted correctly; (2) Precision: representing the proportion of the positively predicted terminal values (i.e., done=1) that were correct; and (3) Recall: representing the percentage of the actual positive terminal values (i.e., done = 1) that were predicted correctly by the trained model.

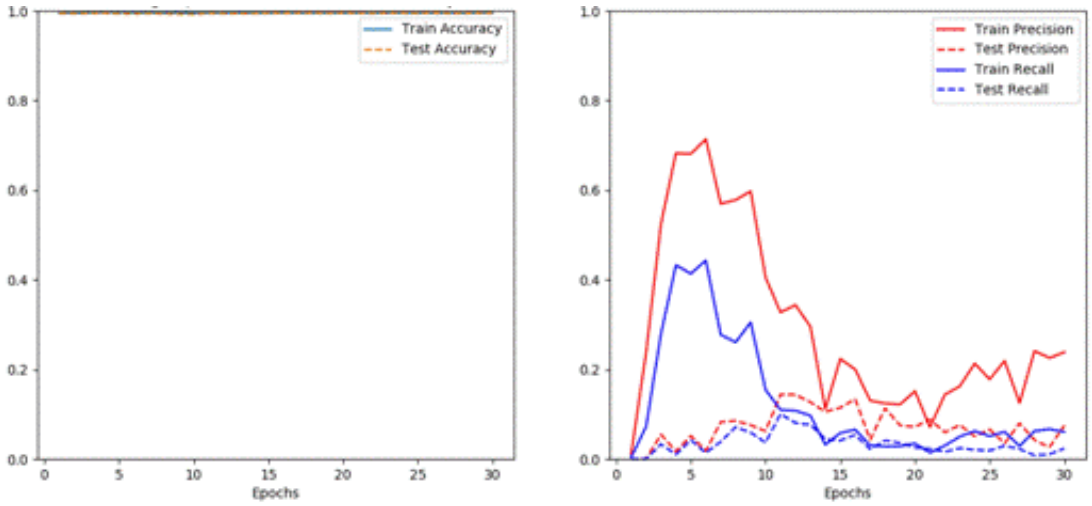


Figure 34: Prediction performance of the baseline World Model implementation.

Figure 34 shows the performance of the World Model in [171] implemented without any modifications. Note horizontal axes represent epochs of training; vertical axes represent percentage accuracy/precision/recall. Clearly, in terms of accuracy (left plot) as defined above, the model seems to perform well. However, if we focus in on the accuracy of the terminal value predictions by computing the precision and recall (right plot), the performance is quite poor. This contradiction arises from the fact that during any given rollout of game-play used for training and testing a given model, there are significantly more non-death events than death-events, resulting in very few examples of positive terminal values to learn from. As a result, not only does the model perform poorly when it comes to predicting done=1 cases, but it also gives the illusion of high accuracy, regardless, due to the fact that it performs so well in predicting the large number of done=0 cases in the test-data, which overshadows those few instances of done=1 cases that arise that it predicts incorrectly.

In order to rectify this issue, it was clear that the model had to be guided towards paying closer attention to the few done=1 cases that occurred in order to learn them better. To accomplish this, the Binary Cross-Entropy (BCE) component of the loss-function used during training had to be modified.

The baseline loss function of the World Model in [171] consists of three parts: (a) the GMM loss, pertaining to how well the next observation is predicted; (b) the MSE loss, pertaining to the prediction of the reward value in the next frame; and (c) the BCE loss, pertaining to the prediction of the terminal value (i.e., the "done" value) in the next frame. This third component of the loss was modified in three different ways to study the resulting impact on terminal value prediction: (1) Adding different weights to the positive and negative terminal value loss components of the BCE loss function; (2) Adjusting the terminal values leading up to the actual death event in the training data to increase the number of positive terminal value observations to learn from; and (3) "Softening" the penalties for incorrect terminal value predictions close to the actual death-event.

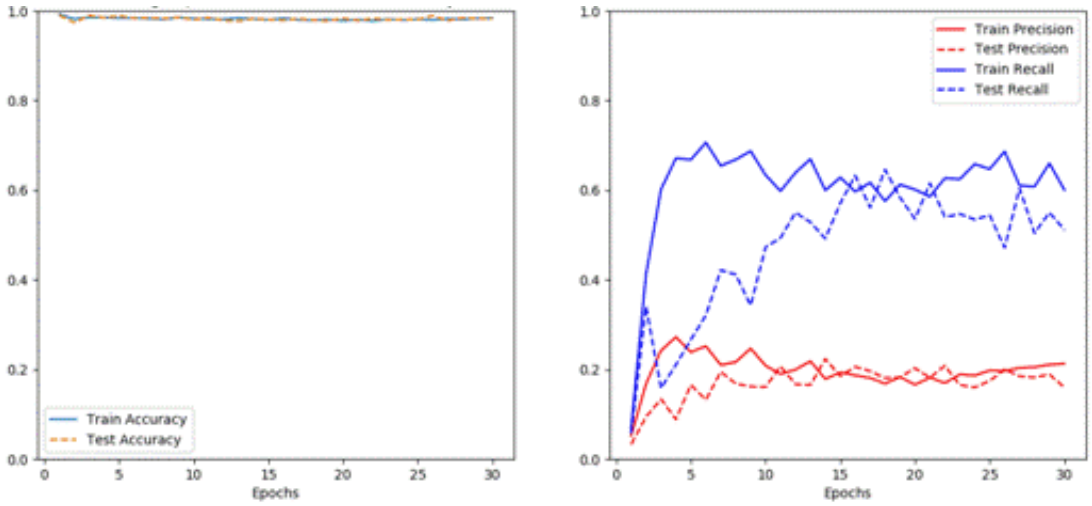


Figure 35: Model performance under BCE weights modification.

Modifying BCE Weights

Here, the BCE loss function was modified such that the weight associated with incorrect prediction of positive terminal values was greater compared to that for negative terminal values. Through testing it was found that a significant difference in these two weights was required to effect any significant change in the precision and recall performance of the m odel. In particular, with the positive terminal value prediction loss weighted at 0.93, and the negative terminal value prediction loss weighted at 0.07, Figure 35 shows the resulting impact on model performance. Note horizontal axes represent epochs of training; vertical axes represent percentage accuracy/precision/recall.

This modification had the greatest impact on the recall performance metric, resulting in just over 50% of the actual positive terminal values predicted correctly during testing. On the other hand, although it improved, precision remained below 20%, indicating that a large proportion of predicted done=1 values were incorrect (i.e., many false positives).

Adjusting Terminal Values

In this second investigation, the terminal values corresponding to observations leading up to the actual death event were purposefully made positive. The intuition here was that by doing so, the model would learn to associate environmental states over a short time-span leading up to the death event with positive terminal values as well. This would lead to a slightly premature emergency situation detection, which is an acceptable compromise and, in fact, could even be beneficial by providing a little more time for the agent to react. Figure 36 shows the impact of this modification for terminal values adjusted over a time-span of 40 frames prior to a death-event. Note horizontal axes represent epochs of training; vertical axes represent percentage accuracy/precision/recall.

Compared to simply modifying the BCE weights, it can be seen, then, that this adjustment has a much more significant impact, with precision now rising to about 80% and recall close to 90%. Thus, by increasing the number of positive terminal frames to learn from in a “safe” way (i.e., associate only frames prior to the death-event with positive terminal values) the model is better able

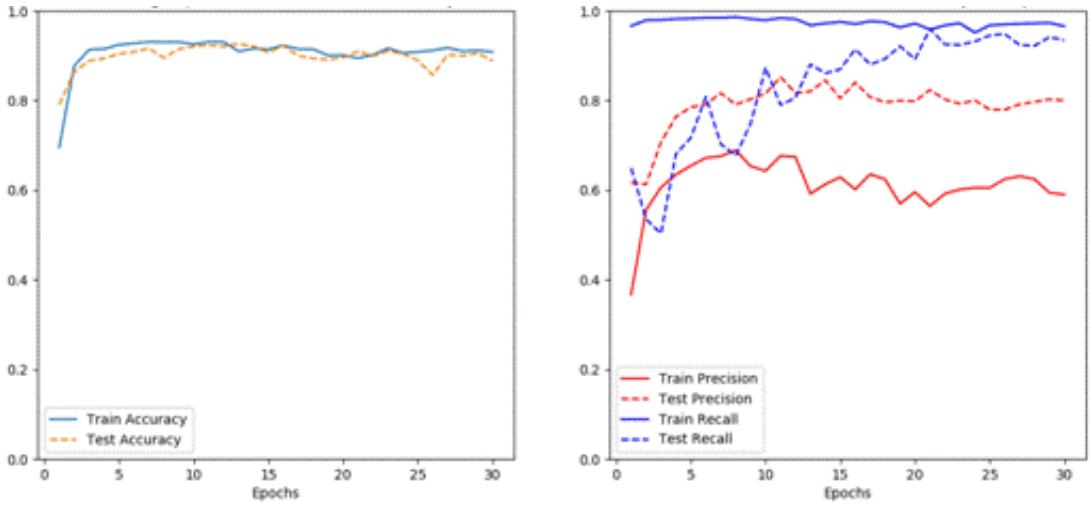


Figure 36: Model performance under terminal value adjustment approach.

to predict when an emergency situation will occur, with greater number of the death events actually caught and fewer false positives.

Using Soft Precision

A third approach that was also investigated was to consider lowering the strictness with which terminal value prediction correctness is adjudicated. This can be done through a gradual decrease of the loss-weight for incorrect positive terminal value predictions around the true death-event frame. Namely, a linear scaling of this loss weight, from 1 to 0, over some N steps leading up to the death-event frame would ensure that the model is not penalized as heavily for predicting a death for frames prior but close to the actual death-event. The intuition, and thus the expected net impact on the agent's learning, is similar to that for the second investigation above. Essentially, the agent may predict emergency situations a little before they actually happen, which is a good compromise if it improves overall performance. Figure 37 shows the results for a linearly scaled loss-weight, starting at five steps prior to an actual d eath-event. Note horizontal axes represent epochs of training; vertical axes represent percentage accuracy/precision/recall.

Clearly, applying this modification also had a significant benefit in terms of precision and recall for the agent. Indeed, the corresponding percentages are not too different from those in Figure 36. With recall being close to 90%, more of the actual death events are caught in the predictions, thus improving the reliability of the death-predictor. Nevertheless, with precision remaining around 80%, it still indicates that a significant number of false positives tend to occur with this model's predictions.

Hence, applying a combination of all these three modifications, or even just the first and third (i.e., using a weighted BCE with soft precision) allows us to improve the model's accuracy in predicting death-events. These results, however, pertain to the model's performance in a single-step prediction. For the purpose of reflexes, we are interested in how well the model performs when used to predict multiple steps into the future.

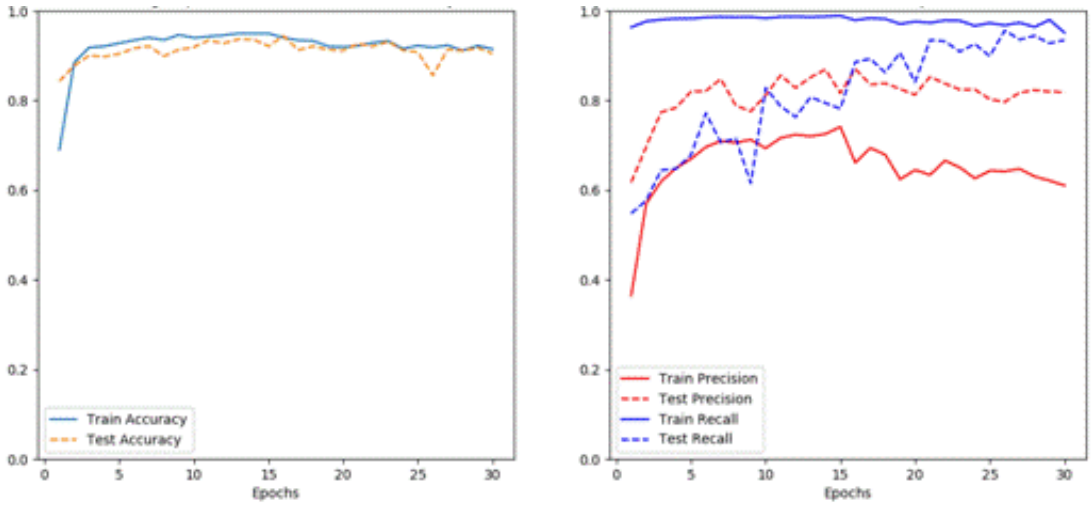


Figure 37: Model performance with soft precision.

Overall Improved Prediction Model

Another set of simulations were run wherein the World Model, implemented and trained with all three modifications presented above, was used to make predictions over a range of future time steps. Using a weighted BCE with positive terminal value adjustment and linearly-scaled soft-precision applied to the five steps prior to a death-event during training, the resulting model was used to predict future death-events over a time-horizon ranging from 1 to 40 time steps into the future.

The Space Invaders OpenAI Gym Atari game was used as the environment on which to test/train the model. The VAE component of the World Model was trained separately from the MDRNN part. The PyTorch libraries were used, and training was conducted using the Stochastic Gradient Descent optimization algorithm (through the "Adam" optimizer provided by PyTorch) in the RL loop. Figures 38, 39, and 40 show the performance of the model in terms of accuracy, recall, and precision, respectively, compared with the baseline. Note horizontal axis in these figures represents number of time steps predicted into the future; vertical axis represents corresponding performance metric.

The plots in Figures 38 to 40 show the mean (solid line) and standard deviation (shaded area) of the corresponding metric over five repetitions of the simulation tests. With the model tasked with predicting multiple time steps into the future, both recall and precision understandably drop (with accuracy remaining high due to being dominated by the relatively large number of correctly predicted done=0 cases). Nevertheless, the targeted 50% accuracy in death-prediction can indeed be achieved for a 10-step future prediction setting.

Reflex Retrieval

A reflex retrieval component, while not mandatory, serves as a way to possibly expedite the search for a good reflex during deployment. As a preliminary approach to begin investigating this idea, a fast nearest-neighbor search using latent representations was employed to try to match new emergency situations with existing ones within a given database. An investigation was also conducted into the

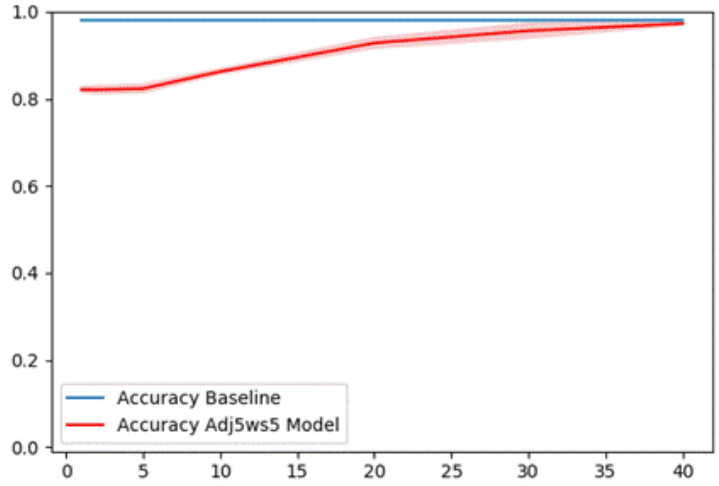


Figure 38: Multi-step future death-event prediction accuracy of the modified World Model.

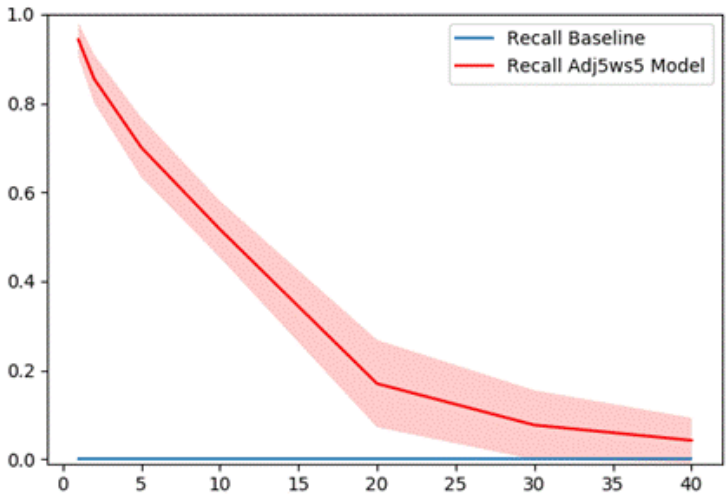


Figure 39: Multi-step future death-event prediction recall of the modified World Model.

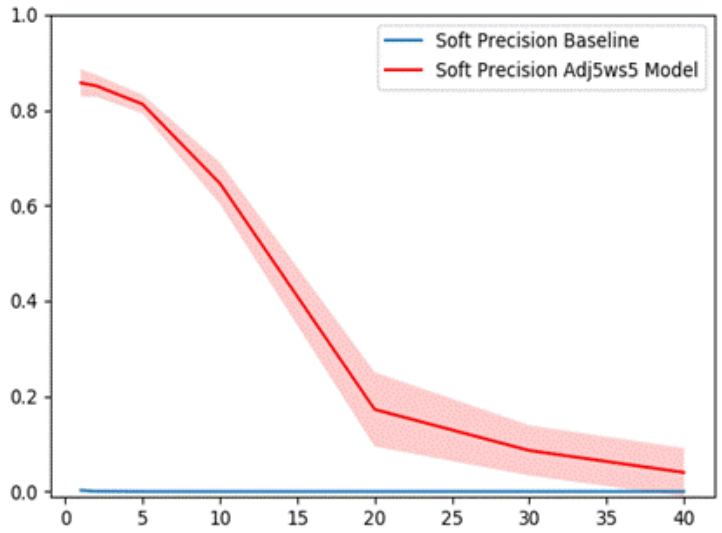


Figure 40: Multi-step future death-event prediction precision of the modified World Model.

development of a CNN classifier for the purposes of reflex retrieval.

Fast Nearest-Neighbor Search Approach

To implement the nearest-neighbor search approach, the convolutional VAE developed in the work on World Models done by [171] was used. The input to this VAE is a 64x64 grayscale observation image from an Atari game, and the output is a corresponding latent vector representation. During training, the model is designed to learn the parameters (i.e., mean, μ, and standard deviation, σ) of a probability distribution for each component of the latent vector. Then, for a given input observation image, each element of the corresponding latent vector is a random sample taken from the corresponding distribution. This allows the model to be less sensitive to strange or outlier latent vectors that may be passed to it from the MDRNN part when it is first being trained, thereby helping to make the overall World Model more robust. After training, when the model is put into use for our purposes, the mean value of the distribution of each component is used so as to ensure a repeatable and consistent mapping from input to output. Conducting reflex retrieval, then, involves first creating (and maintaining/updating) a database of previously encountered emergency situations represented as latent vectors. Then, when a new emergency situation is encountered during deployment, it is first converted to a latent representation using this same VAE, and the closest latent vector from the database (within some threshold) is taken to be the matching emergency situation whose solution could be used for this newly-encountered scenario.

To test this approach, two datasets of reflexes were constructed, one to serve as a database within which to search, and the other to serve as a test-set with which to conduct the nearest-neighbor search. The database consisted of 20,000 5-image observation histories each taken 20 time steps before an imminent death event, obtained from the previously-mentioned sampling-based reflex search method. The test-set consisted of 2,000 such observation histories representing new emergency events for which a match would be attempted to be found from the database.

Firstly, a ground-truth set of results were obtained by computing the pixel-based differences between

each image-set in the test-set and each image-set in the database. In particular, the Frobenius matrix norm of the image-set difference between each pair of test-set image-set and database image-set was computed. The pair with the minimum such norm value that fell under a specified threshold value was taken to be a match. If no pair gave a norm value falling under the threshold, it was considered as simply a test-set element that had no match. Hence, this ground-truth search established what percentage of the test-set actually had a matching emergency event in the database.

Secondly, the Python Fast Library for Approximate Nearest Neighbors (PyFLANN) library of fast nearest-neighbor search algorithm implementations was used to conduct and test the nearest-neighbor search approach for this application. To do so, each item in both the test-set and the database was converted into a latent vector representation using the VAE model described above. The reflexes and, thus, the VAE, were based on the Space Invaders Atari game. The k-means nearest-neighbor search algorithm was then conducted on these latent vector test- and data-sets to obtain a list of the nearest neighbor for each element in the test-set, as well as the corresponding distance value. These distance values were then compared to a second threshold to determine if the corresponding element from the test-set and the database could be considered a matching pair. The two thresholds here (one for the ground-truth search and the other for the nearest-neighbor search) were selected in such a way as to ensure the same percentage of matches occurred between the two.

To gauge how well the nearest-neighbor search performed, the matching accuracy obtained by using this method was determined. Here, accuracy refers to how many of the matched elements in the test-set were "correct" matches, where the "correct" match was that resulting from the ground-truth search. Two variations of the nearest-neighbor search approach were conducted, one using a 32-element latent vector representation and the other using a 64-element vector. Test results, however, showed only about a 10% matching accuracy for both cases. Moreover, there was poor correlation between the image distances calculated for the ground truth data and the latent vector distances computed by the nearest-neighbor algorithm, Table 4.

Table 4: Results summary of the nearest-neighbor search approach.

Latent Dimension	**Percent Matched via Image Difference**	**Percent Matched via Nearest Neighbor**	**Nearest Neighbor Matching Accuracy**	**Pearson's Correlation Coefficient**	**Spearman's Correlation Coefficient**
32	41.4	23.9	10.9	0.3965	0.4639
64	41.4	32.7	10.4	0.3348	0.4420

However, since the test-set is a separately generated set of death-event observation-stacks from those in the database, there has been no control over the degree of similarity between the two. Hence, what the results really show is that the nearest-neighbor search approach has a difficult time generalizing from the data contained in the database. Thus, when presented with new death events that are not exact matches with those found in the database, the ability to reliably produce good matches is poor.

Subsequent tests were therefore conducted using a subset of the database as the test-set to see how

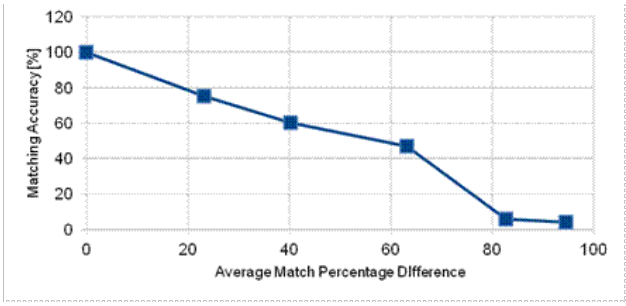


Figure 41: Reflex retrieval of the fast nearest-neighbor search approach for the Space Invaders Atari game.

many matches one can get with the nearest-neighbor approach and how accurate those matches are. In these tests, each of the five observation images of a given emergency situation were converted to a latent representation individually, and the resulting latent vectors were concatenated to form a single 160-element latent vector representation of the emergency situation. The "nearness" of a given emergency situation to anyone in the database can then be measured as the Frobenius norm of the difference between their corresponding 160-element latent vector representations. One may thus search for an exact match in the database, or one may try to find a "close" match by searching for an emergency situation for which the difference in latent representations is below some pre-defined threshold.

To implement this approach, the PyFLANN library of fast nearest-neighbor search algorithm implementations was once again used, and matching accuracy tests were performed on emergency situations obtained from the OpenAI Gym implementation of the Space Invaders Atari game environment. A total of 2,000 emergency situations and corresponding good reflex action-sequences were obtained through game rollouts of Space Invaders and the sampling-based reflex generation method described previously. Each item in this database was converted to its latent representation and this database was separated into a test-set and a dataset.

The test-set was selected in such a way as to control the average percentage difference between each latent vector emergency situation in the test-set and its closest match in the dataset. Hence, a series of tests were conducted, each involving a gradually increasing average percentage difference, starting from 0% and moving up to about 90%. A difference of 0% indicates a test of performance in exact-matching, while those of higher percentages represent tests of performance in finding a nearest match. By removing the test-set elements from the database for all non-exact-match tests, we can establish the ground truth results for matching by finding those elements in the remaining database (i.e., the dataset) that show the smallest magnitude of image-difference between the emergency-situation frames in the dataset and those in the test-set. Then, by performing the nearest-neighbor search algorithm on the latent representations of each element in the test-set with those in the dataset, and comparing the results with the ground-truths, we can test the matching accuracy of this approach. The plot in Figure 41 shows how the matching accuracy of the nearest-neighbor search approach varies with the average difference between the test-set elements and their closest found matches.

With a more controlled experiment such as this we see a marked improvement in this approach's

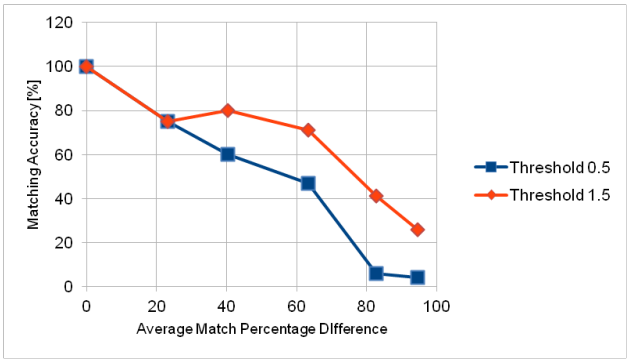


Figure 42: Reflex retrieval difference for two different latent vector difference thresholds.

performance for exact and near matches. The results in Figure 41 show that the nearest-neighbor search approach performed reasonably well on this game environment. While matching accuracy does drop rather fast with increasing average difference of the nearest match, one can observe an approximate 80% matching accuracy for a 20% average difference in the closest match. This indicates some ability to generalize and reliably find good "close" matches even if no exact matches exist.

Thresholds for maximum image and latent-vector differences at or below which two emergency situations were considered to be a match had to be established to perform these tests. If the threshold on the latent-vector difference is increased, one would expect a greater chance of finding a match from the dataset. However, this would also tend to increase the variability of the matches found, thereby decreasing the chances of finding the "correct" close match and negatively impacting reflex retrieval performance. The threshold used to obtain the results in Figure 41 was 0.5. Figure 42 shows how increasing this threshold to 1.5 can impact matching accuracy. Hence, for this particular environment and test- and data-sets used we see an improvement in accuracy, indicating that 1.5 may be closer to an optimal threshold, beyond which there may be negligible gains or, perhaps, a decrease in performance. Nevertheless any performance gains obtained may not justify the added cost of having to search for an optimal threshold, especially if the nearest-neighbor search retrieval method is not intended to be used beyond some percentage difference threshold anyway. In the case of Figure 42, for example, if reflex retrieval were to only be used for percentage differences less than or equal to 20%, the improvement obtained by refining the threshold may not be sufficiently significant.

CNN Classifiers for Reflex Retrieval

A preliminary investigation was also carried out to determine if a CNN-based classification neural network could be trained to detect when a given emergency situation matches one in a given database. The approach used was to detect a matching emergency situation by considering how well the corresponding reflexive action-sequence matches those corresponding to the emergency situations in the database. Thus, rather than trying to match the input observations directly, we can try to match outputs.

To do so, we construct a neural network that can be trained to find the reflex action-sequence

corresponding to a given emergency situation in the database. It can be trained to learn the current database well, so that when an emergency situation is encountered during deployment, the network outputs what it believes is the corresponding action-sequence, as well as the probability with which it believes this action-sequence is correct. Hence, a low-probability output would indicate no good matches, whereas a high-probability output would indicate a good match.

To build such a network, we considered breaking the classifier down into two main components: (1) a CNN auto-encoder for data-reduction, and (2) a linear classification network to associate this reduced output to a reflexive action sequence. This system was implemented and tested in the context of OpenAI Gym Atari games, wherein each action in the sequence is a discrete action that can fall into one of several classes defined by the discrete set of actions that make up the action-space of the game.

The CNN-based auto-encoder is the VAE used in the work done in [171] on learning policies for Atari games. However, modifications were made to the dimensions of the input and output.

In particular, input images were 3-channel, 84×84 images, while the latent representation had a dimension of 1×64. The role of the VAE was to reduce the image-based emergency situation representation into a l-dimensional latent vector, which could then be input to a linear network trained to classify the actions in the reflex action sequence from this latent vector, rather than operating on the raw images. The VAE thus takes over the heavy image-processing computation overhead of analyzing the images that comprise the emergency situation to distill them down to the essential information. The linear classifier then has a significantly lower dimensional input to work with, which, in turn, simplifies the network greatly.

The linear classifier (LC) network, used to map latent emergency situation vectors to a reflex action-sequence, is shown schematically in Figure 43 and consists of three linear, fully-connected layers in addition to the input and output layers. The ReLU activation function is used after the second and third hidden layers. All hidden layers have dimensions $1\times1{,}000$. Emergency situations are represented by a stack of five images obtained from the 5-step history before the time step at which the emergency situation is predicted. Each image is converted to a 1×64 latent vector individually using the trained VAE. These vectors are then concatenated to form a single vector of shape 1×320. This concatenated latent vector then forms the input layer of the LC network.

For the tests conducted, a 10-timestep reflex action sequence was considered. Thus the LC network must take the 1×320 latent representation of the emergency situation and map it to a 10-step action-sequence. Since all OpenAI Gym Atari games have an action space consisting of integers in the range [0, 17], each action in the reflex action-sequence can be represented by a 1×18 vector of binary values. The value at the index corresponding to the action to be taken would be a '1', while all other values in that vector would be a '0'. Thus, for a 10-step action-sequence, we would have 10 output binary vectors, each of size 1×18, and each containing a '1' only at the index corresponding to the reflex action at that time step. This binary vector can therefore be thought of as containing probabilities, where a '1' indicates the correct action.

Hence, in the LC network, the final hidden layer is fully-connected to 10 separate output layers, where each output layer is a 1×18 vector containing probabilities for each possible action at the

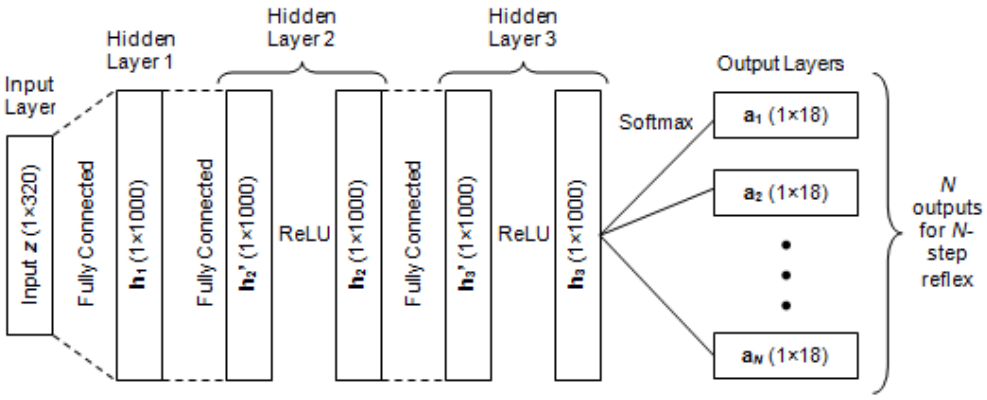


Figure 43: Schematic of the linear classifier network used to trigger reflexes.

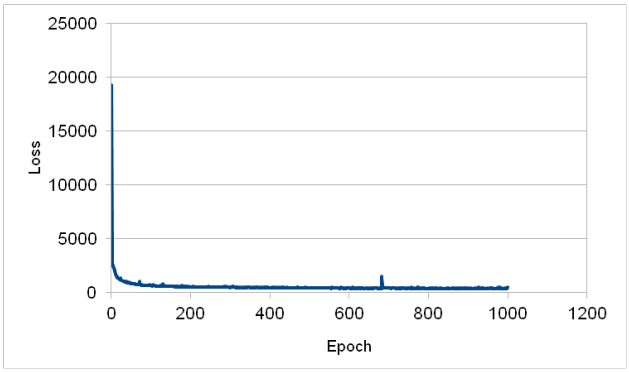


Figure 44: Training loss for the CNN-based VAE.

corresponding time step. The action with the highest probability would be the action to take for that time step. In order to ensure that the output vectors contain values in the range [0, 1], the softmax activation function is used on all the 10 output layer vectors.

To train the VAE, a total of 10,000 images, generated by playing rollouts of the River Raid Atari game, were used. These images were fed into the network in batches of 64, and training was conducted over a span of 5,000 epochs. The PyTorch set of python libraries were used to build and train the network. This same VAE has been used in our prior tests when developing other components of the reflex module. As such its ability to successfully encode and decode Atari game images has already been verified. For reference, Figure 44 shows a plot of the training loss over the first 1,000 out of the total 5,000 epochs of training.

As part of preliminary tests of the CNN classifier, the LC network was trained using an archive of 30 different emergency situations encountered through many different rollouts of the River Raid Atari game played by a random agent. Using a random agent ensured that unsafe actions would be more likely to be performed, thereby increasing the chances of encountering situations in which a 10-step future prediction showed an imminent death-event under the agent's current policy. An effective 10-step reflexive action-sequence was found to circumvent each emergency situation as well.

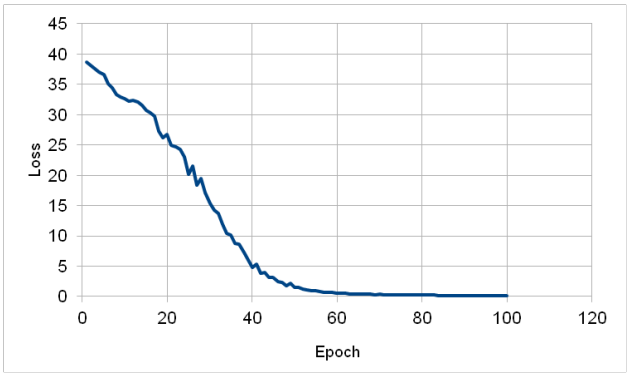


Figure 45: Training loss for the LC network.

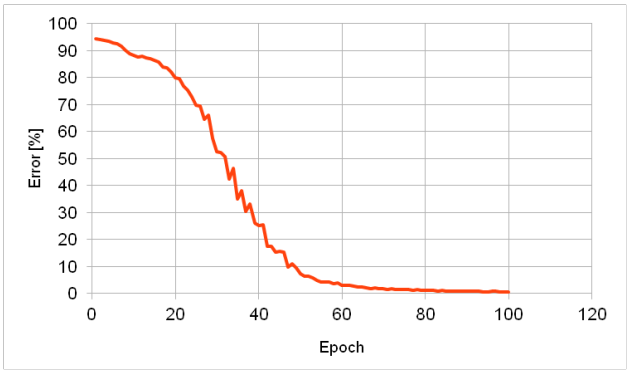


Figure 46: Average absolute error per predicted output.

These emergency situations, each represented by a 5-timestep observation history, were converted to their corresponding 1×320 latent vectors using the trained VAE. They were then paired with their corresponding 10-step reflex action-sequence to produce the training data for the LC network. The LC network was trained using batches of 12, and over a span of 100 epochs. The training (cross-entropy) loss is plotted in Figure 45 and shows very promising results, as the significant drop in the loss value indicates that the network is able to learn the mapping from latent emergency situations to reflex action-sequences well.

Moreover, the average percent error per output vector, measured as the mean of the absolute differences between the probabilities output vectors and their corresponding binary action vectors, were computed. The corresponding plot is given in Figure 46, which shows that the actual mathematical difference between the binary vector labels and the probability vector outputs is reduced significantly as well. The resulting low percentage error at the end of the 100 epochs of training indicates that the LC network, combined with the CNN-based VAE, can successfully learn the mapping from emergency situation to reflex action-sequence for the elements in a given archive of priors. As part of future work, this CNN classifier will have to be tested further within the context of training an agent to see how well it performs in matching emergency situations that are different from but close to the priors stored in the archive.

Reflex Generation

To devise reflexes when an emergency situation is encountered during deployment, an online optimization approach is proposed. This allows for an adaptive reflex response, customized to the particular emergency situation encountered. This optimization entails an intelligent search in the reflex action-sequence solution space conducted online, with the fitness of each solution investigated being determined by using the predictive model to rollout the corresponding action-sequence. However, since further research still needs to be done to develop a working predictive model, an ideal predictor was used in all the tests that were performed.

The approach to reflex generation proposed here is to apply Bayesian Optimization (BO) to find a customized reflexive response to a given emergency situation as it arises. This approach involves using Gaussian Processes to model the underlying fitness function that describes the performance of a given reflex using successive, but intelligent, selections of potential reflexive actions and testing them out using a predictive World Model. An important assumption here is that such a World Model is available that allows us to make accurate predictions of the state of the environment, given actions, multiple time steps into the future. This model is queried repeatedly to ascertain the impact of a given reflex and to compute its corresponding fitness.

The BO approach, starting with a few random reflex samples to seed the search, gradually builds and maintains the mean and uncertainty information on the fitness function for the emergency situation in question. With each subsequent sample taken, the particular reflex to try is chosen based on a user-specified parameter that balances exploration (i.e., sample a solution point in a probable high-fitness region with more uncertainty) and exploitation (i.e., sample a solution point in a probable high-fitness region with less uncertainty). The BO approach nominally starts with no knowledge of the fitness function (i.e., fitness function is '0' everywhere with some specified upper and lower bounds on uncertainty). A set of random samples must first be taken to begin to obtain some information about the fitness function. The number of samples needed will depend on the application, and must be empirically determined. Nevertheless, these initial points are required to "seed" the search, after which the BO approach can commence as normal, with successive reflex points selected to balance exploration and exploitation. However, once a set of sampled points have been obtained for a given emergency situation, these points and their fitness values can be saved for future use. In particular, if this emergency situation, or one similar to it, is encountered in the future, one can recall the set of high-performing solutions and use them to seed the search. In this way, it is more likely that the initial points tried are already close to the optimal solution, thereby allowing the BO procedure to find the optimal solution faster with fewer evaluations.

It should be noted here that the scenario one would most typically expect to see is one where an emergency situation occurs that is very similar to, but still differs in certain specifics, from that encountered before. Hence, even if one runs into a situation on which the BO search has already been performed and a good solution has been found, it will still never be exactly the same as the situation at hand, and the reflex must be customized in order to "fine-tune" it to the case being addressed. Of course, given that the scenario is similar to the one encountered before, the best solution can also be expected to be similar (if not the same) and this reduces the amount of computational effort that would be required if those prior solutions are used as starting points to seed the search.

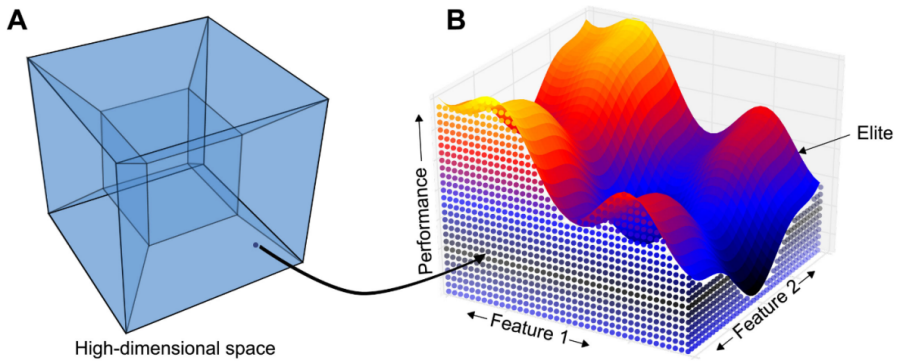


Figure 47: Illustration of the MAP-Elites algorithm.

Hence, good priors help the BO search to find a good solution faster, but are not required. Maintaining an archive of good priors, then, while not absolutely necessary, can be beneficial for online computation during deployment in time-critical applications. This is where the MAP-Elites algorithm can be used. We next present the work that was done on developing a multi-task version of the original MAP-Elites algorithm to address the sub-problem of generating an archive of priors that could be used to help the reflex generation optimization searches that are performed during deployment. And further next, we present the BO method for conducting the online reflex-generation search.

The Archive of Priors

Here, a multi-task version of the standard MAP-Elites [172] algorithm was devised. It is used to first find high-performing solutions for many different emergency situations together, offline, in simulation, through an evolutionary approach, thus producing an archive of "priors". During deployment, these priors are used as good starting points to seed a search method that rapidly finds customized reflex action-sequences online that are fine-tuned to the specific scenario at hand through an intelligent search of the solution space.

In the standard MAP-Elites algorithm [172], a high-dimensional solution space of a complex problem is reduced to a relatively lower dimensional "feature" space defined by dimensions of variation chosen by the user (Figure 47). The MAP-Elites algorithm then finds highest-performing solutions over this entire reduced-dimensionality feature-space. The advantage of this is that rather than obtaining a single optimal solution, the user is provided with a map of good solutions to illuminate the fitness potential for different combinations of the features of interest. As such, this type of algorithm is referred to as an "illumination algorithm".

As illustrated in Figure 47, by converting a high-dimensional solution space into a fitness potential map over a grid defined by user-specified features of variation, it explicitly reveals to the researcher the tradeoff between fitness (i.e., solution performance) and different values of each feature. For example, consider the task of designing a robot, with its speed being the performance criterion. A researcher might select the robot's height and weight as being two features of interest. The MAP-Elites algorithm then finds a set of high-performing solutions for each combination of these

features. Note that this does require discretization of the feature space, for which a CVT is used in the interest of scalability with the number of features (see [167] for more details). The resulting archive of high-performing solutions allows the researcher to understand how different settings for the height and weight of the robot impacts the speed of the resulting design and what directions of variation result in improving or deteriorating performance.

The mechanism by which the MAP-Elites algorithm works to find these high-performing solutions is evolutionary computation. In order to generate a new solution with each iteration of the MAP-Elites algorithm (that addresses an as-yet unexplored cell in the feature map or potentially improves upon a previous solution), standard mutation and crossover operations of evolutionary algorithms are used on existing solutions (also referred to as "genomes" or "genotypes" as per evolutionary computation terminology). The values that any given genome is comprised of (e.g., a sequence of actions describing a game-play policy) are referred to as a "behavior". Moreover, because MAP-Elites produces a set of different high-performing solutions for different feature combinations of a given task, it falls under the category of Quality Diversity (QD) optimization algorithms.

When we consider a problem such as finding an effective short-term action-sequence (i.e., a reflex) to circumvent an imminent catastrophic event in an Atari game, for example, it is clear that we are dealing with a problem involving multiple different tasks that would each need a good solution in order to provide an effective safety module. Here, each potential catastrophic event is a unique emergency situation that requires an action-sequence that can successfully circumvent this imminent dangerous event. If we can rapidly find effective reflexes to handle a wide assortment of emergency situations through a QD approach, namely, MAP-Elites, we will essentially create a repertoire of reflexes that can be drawn upon if and when needed. This, therefore, requires a multi-task version of the MAP-Elites algorithm.

In the multi-task MAP-Elites algorithm that was devised, we tested the key underlying principle that we hypothesize through analogy with nature, which is that high-performing solutions for different tasks will share many key characteristics. This is analogous to what we find among different species in nature, where organisms residing in different niches will still have significant similarities in their DNA [173]. As a result, when performing a search to build a repertoire of good reflexes for different emergency situations, a high-performing solution for one task is likely to be a good initial solution from which to search in order to solve another task.

Even in this multi-task version of the algorithm, the basic principles of MAP-Elites continue to apply, namely:

- the task descriptor space is divided into a large number cells (or "niches"), that are organized spatially in an archive (also called "map") – thus, each task is a niche;
- each niche contains the best known solution (i.e., the "elite") for the corresponding task; and
- new candidate solutions are generated by performing conventional mutation and cross-over operations from evolutionary algorithms on two randomly selected elite solutions.

Based on the aforementioned underlying hypothesis, the basic workflow of the multi-task MAP-Elites (MTME) algorithm is as follows: (1) Randomly select a certain proportion of tasks from the

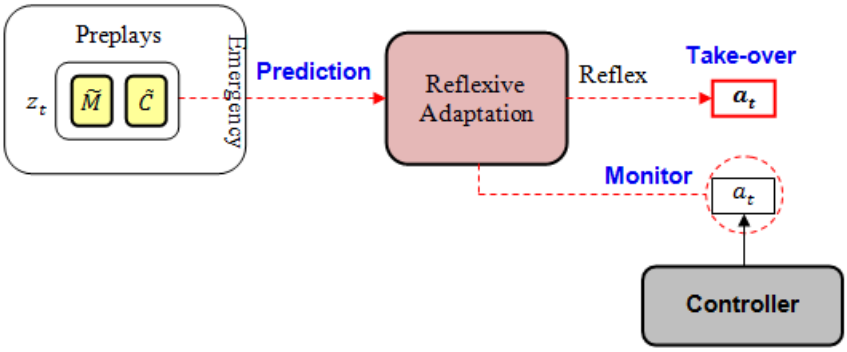


Figure 48: The reflex module monitors and takes over the controller's actions as required.

task descriptor space and seed them with a random solution (i.e., a random genome); (2) Compute the fitness for each solution generated and store them in an archive. Each solution is the current elite of its corresponding niche; (3) Randomly select two elite solutions to serve as parents for evolution; (4) Apply mutation and cross-over operations on the parents to create a solution; (5) Choose a task from the map to potentially assign this solution to; (6) Call the fitness function to evaluate the performance of this solution on the selected task; (7) Compare the resulting fitness with that of the current elite (if one exists) for this task stored in the archive; (8) Assign this new solution to this task if no elite currently exists for it, or, replace the current elite for this task with the new candidate solution if it evaluated to a higher fitness, and then update the archive accordingly. If the current elite for this task has the higher fitness, discard the candidate solution that was just generated; (9) Repeat steps 3 to 9 until a pre-defined number of maximum evaluations (i.e., the evaluation budget) is reached.

By employing this algorithm on a set of emergency situations generated through many rollouts of different Atari games, one can generate a database of such situations along with corresponding good reflex action-sequences. This database then becomes the archive of priors that can be used with BO during deployment to find customized reflexes to actual emergency situations encountered.

The BO Approach

In the larger scheme of things, the reflex module works to monitor the actions output by the controller of the agent being trained, and takes over when necessary. As illustrated in Figure 48, the reflex module monitors the controller output and takes-over with an appropriate reflexive response when dangerous actions are predicted. Predictions of the impact of the controller's actions, as well as any reflexes sampled during the search procedure, are conducted through preplays using the latest saved copy of the predictive World Model, $\tilde{M}$. Controller actions given predicted environment states are obtained by querying the latest saved copy of the system controller, $\tilde{C}$.

Figure 49 provides a closer look at how the BO approach works to find a good reflex online that is adapted to address a given emergency situation when it arises. From factory training, we obtain an archive of emergency situations and corresponding reflexive actions found from simulation (for example, through playing many rollouts of Atari games, or running many driving scenarios in the CARLA autonomous driving simulator). This archive contains a database of emergency scenarios

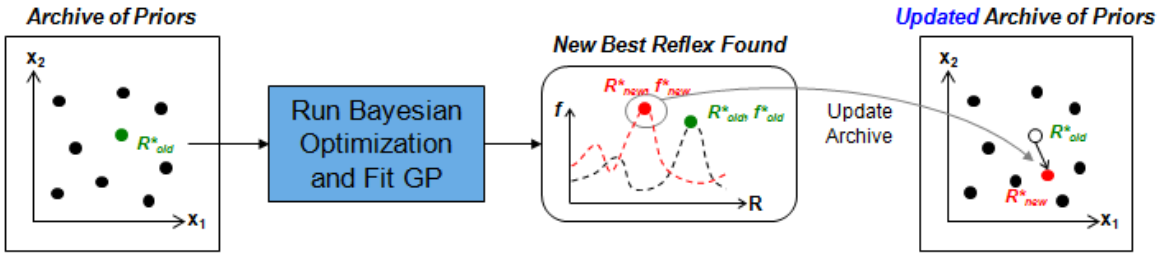


Figure 49: Overall workflow of the reflexive adaptation procedure.

as well as the best reflex, R^*, for each of them. The reflex, for example, may be represented by a set of two intermediate waypoints, (x_1, x_2), that define a safe short-term trajectory that the agent must implement to circumvent the emergency (i.e., $R^* = \{x_1, x_2\}$). Each element in this archive also contains the fitness value corresponding to each best reflex found so far.

Subsequently, during deployment, if an emergency situation is predicted, the first step is to find the closest match within the archive of priors, and to recall the corresponding best reflex, R^*_{old}, to use as a starting point for the BO search procedure. Since simulation will never be able to capture reality exactly, one can expect that there will always be some variation between the matched scenario from the archive of priors and the details of the actual emergency situation being dealt with. Hence, it is necessary to run an online optimization, starting the search with this good solution obtained from simulation, which can be expected to be close to the true optimal reflex solution for the real-life scenario. Then, through the rapid and intelligent search procedure provided by the BO approach, seeded with this good solution, an adapted reflex, fine-tuned to this scenario, can be found and implemented to circumvent the emergency.

Under normal circumstances, the prior found from the archive should provide a solution that is close to the optimal for the scenario encountered. However, when there is a change in task, it is likely that this good solution in the archive will now perform sub-optimally. In particular, with the task having changed (e.g., changing the vehicle from a car to a motorcycle) the fitness function for the reflexive actions will change, and the fitness value, f^*_{old}, corresponding to the reflex found from the archive, R^*_{old}, will no longer be accurate. The new fitness function will now have a new optimal reflex, R^*_{new}, with a different corresponding fitness value f^*_{new}. When the optimization search procedure commences, and evaluates the old reflex solution, it will suddenly discover this mismatch in the fitness value. The very nature of the BO search procedure involves balancing exploration of the solution space (i.e., trying solutions in relatively unexplored, uncertain regions to see if they show potential for a better solution) with exploitation of high-performing solutions found so far (i.e., trying solutions near explored regions with less uncertainty that have shown to give the best solutions so far). In so doing, the search will naturally move away from the now low-performing R^*_{old} and gradually find the new optimal solution, R^*_{new}, or at least get very close to it after exhausting its evaluation budget.

Once this new good reflex has been found, the archive of priors must be updated accordingly. Thus, adaptation to new tasks happens in two ways. Firstly, the system recognizes the sub-optimality of

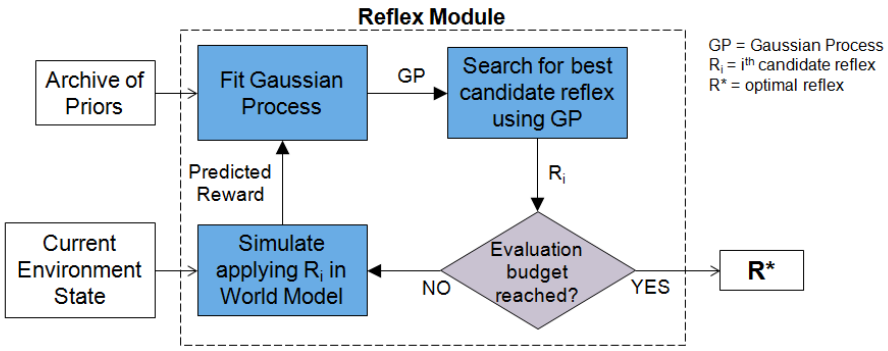


Figure 50: Workflow of the BO search procedure used during deployment.

R^*_{old} and runs a search online to find R^*_{new}, thereby adapting the reflexive action to the emergency situation within the context of the new task. Secondly, the reflex solution and corresponding fitness value are changed to the new solution just found for this emergency situation in the archive of priors. Thus, if this situation is encountered again in the future under this new task, an applicable high-performing solution will be available to seed the search close to the true optimal, thereby allowing the system to even find a good reflex faster. Hence, knowledge can be updated during deployment to give good priors for future scenarios.

The workflow of the BO procedure is shown in Figure 50. Firstly, the archive of priors must be checked to determine if a similar (or same) emergency situation has been dealt with in the past. If one is found, the corresponding best reflex solution found is used to help seed the search. The search procedure involves the use of Gaussian Process regression to build an estimate of the underlying fitness function for reflexes in the given emergency scenario. The form of the fitness function will depend on the particular application. For example, in Atari games, the fitness function could be the cumulative reward obtained by implementing a given reflex. In the context of an autonomous driving simulator, it could be a sum of penalties assigned to the reflex based on the type and duration of traffic violations or dangerous actions taken during the course of the reflex maneuver. In any case, the Gaussian Process regression takes the fitness value computed for each reflex tested during the search and uses them to update an initial model of the fitness function along with uncertainty information.

To start with, a fitness function of '0' could be assumed, with the same Gaussian noise (with mean, 0, and some assumed variance) at all points along the function's domain. With each reflex that is tried, a corresponding point on the function is found. The fitness values of all reflexes tried so far is used to update the known and uncertain regions on the function model. The more points tried, the better the model represents the true function. However, the idea here is to try to sample points close to the optimal of the function. The fewer tries one takes to get closer to the optimal of the function, the faster the optimal solution can be approximated. Thus, it is important to carefully select each successive reflex solution. In the BO approach, an acquisition function is chosen and optimized to find the best point to try next. This optimization is based on balancing exploitation (i.e., sample a point in a region with less uncertainty and close to high-performing points) and exploration (i.e., sample a point in a largely unexplored region of the fitness function with higher uncertainty).

With each point tried, the Gaussian Process updates the fitness function model fit, and then a new point (i.e., the next best reflex candidate, R_i, to try) is selected to be tested. In order to test it to get the corresponding fitness value, the predictive World Model must be used. This model is initialized to the environment state at the point where the emergency event prediction N steps into the future was made, and then is used recursively to play out the reflex action sequence of the candidate being tested. This simulation of the reflex candidate allows one to predict what will happen in the environment if that reflex were to be implemented, and the fitness value can be computed accordingly. This fitness value is used to update the Gaussian Process model, and the process repeats. Once a pre-determined evaluation budget is reached, the best reflex candidate found is selected as the reflex to implement in the actual environment.

1.3.7 Probabilistic Program Neurogenesis

We focused on adding new neurons to fully connected, feed-forward neural networks, however, our approach is equally applicable to recurrent and convolutional networks. Our method is initialized with a neural network that has been previously trained on one or more tasks. The neurogenesis process begins when a new task is presented. Here, we do not address the need to identify the arrival of a new task, but in practice, assuming that the input distribution for the new task is different, one can identify changes in the statistics of the neuron activations in the penultimate layer of the network to infer that a new task has arrived. An example of neurogenesis is shown in Figure 51. The gray blocks are new neurons. The weights on the black connections were learned from a previous task and are fixed, while the green connections are new and have trainable weights. First, a new output (classification/prediction) layer is added to the network. Next, new neurons are added and connections are established. When new neurons are added to a layer it effectively creates a new layer that emits new connections (green) to the new neurons in the nearest layer above, which may be the output layer. Additionally, if new neurons are added to the first hidden layer, then new connections are established from the input layer to these neurons. Restrictions on the connectivity patterns of new connections ensure that the performance of the network on old tasks is not altered. This is because the activations of new neurons do not affect the activations of old neurons.

Estimation of Distribution Algorithm

EDAs are a class of evolutionary computation algorithms [174]. Unlike most EC algorithms, EDAs explicitly encode a probability distribution over the search space from which new individuals are generated. Rather than using mutation and crossover operators, an EDA adapts the parameters of a probabilistic model during the optimization process so that new individuals are increasingly likely to be selected from good regions of the search space. In our approach the probabilistic model is a probabilistic program, which is described below.

Figure 52 shows a diagram of our EDA. The uniqueness of our approach lies in the probabilistic program that learns to generate good performing individuals. The process begins by generating a new population of individuals using the program. Each individual is a neural network and is trained on a pre-defined task, such as a classification problem. Next, the fitness of each individual is computed. We use a fitness function that captures the trade-off between classification accuracy and

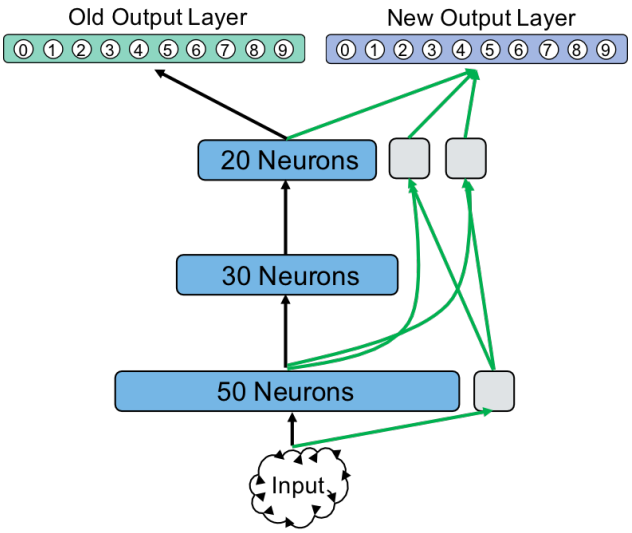


Figure 51: Illustration of neurogenesis used in our approach.

model complexity. The fitness function is given by

$$Fitness = k \cdot accuracy - complexity, \tag{41}$$

where $accuracy \in [0,1]$ and is the accuracy on the new task, *complexity* is the model complexity given by the number of new neurons divided by the number of neurons in the parent network, and $k \in R^+$. The selection process then proceeds in two separate steps. First, one individual is selected using tournament selection. This provides some degree of exploration since the individual with the highest fitness may not be s elected. Second, we use a "hall-of-fame" that always and only contains the best performing individual found over the course of the entire optimization process. If the fitness of the best performing individual in the current population exceeds the best fitness found thus far, then it becomes the individual in the hall-of-fame. The last step is to update the parameters of the probabilistic program. This is done using the newly selected individual and the individual in the hall-of-fame, which may be the same. Updating the parameters tends to adjust the probability distribution represented by the probabilistic program in such a way that it is more likely to sample from good regions of the search space. At this stage the EDA either generates a new population using the updated program or terminates. The termination criterion is that the elapsed wall-clock time has exceeded 21 min, though other criteria are possible, such as those based on a maximum number of iterations or fitness level.

Probabilistic Program Learner

Our approach uses a probabilistic program as a generative model to create new individuals. It has trainable parameters that are adjusted through a learning process, so that over the course of the optimization, individuals are sampled with increasing frequency from the best regions of the search space. We refer to this component as the *Probabilistic Program Learner* (PPL). Figure 53 shows a diagrammatic representation of the PPL. The purple boxes are random variables that specify the characteristics of the individuals, such as the number of new neurons to add. The light blue boxes

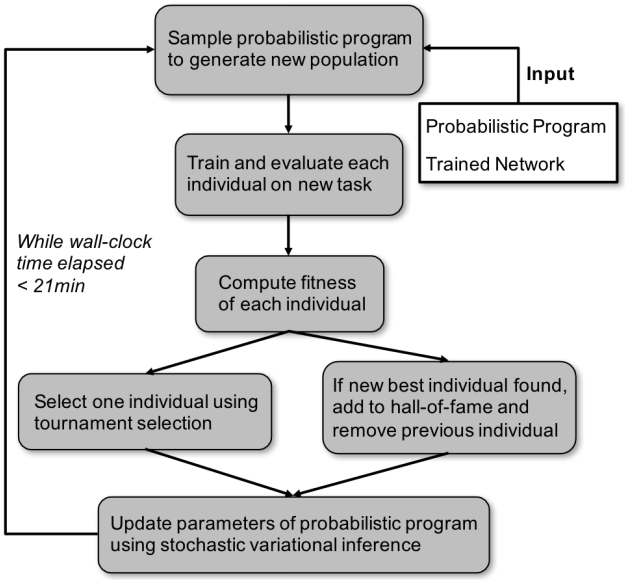


Figure 52: Our EDA used to execute neurogenesis.

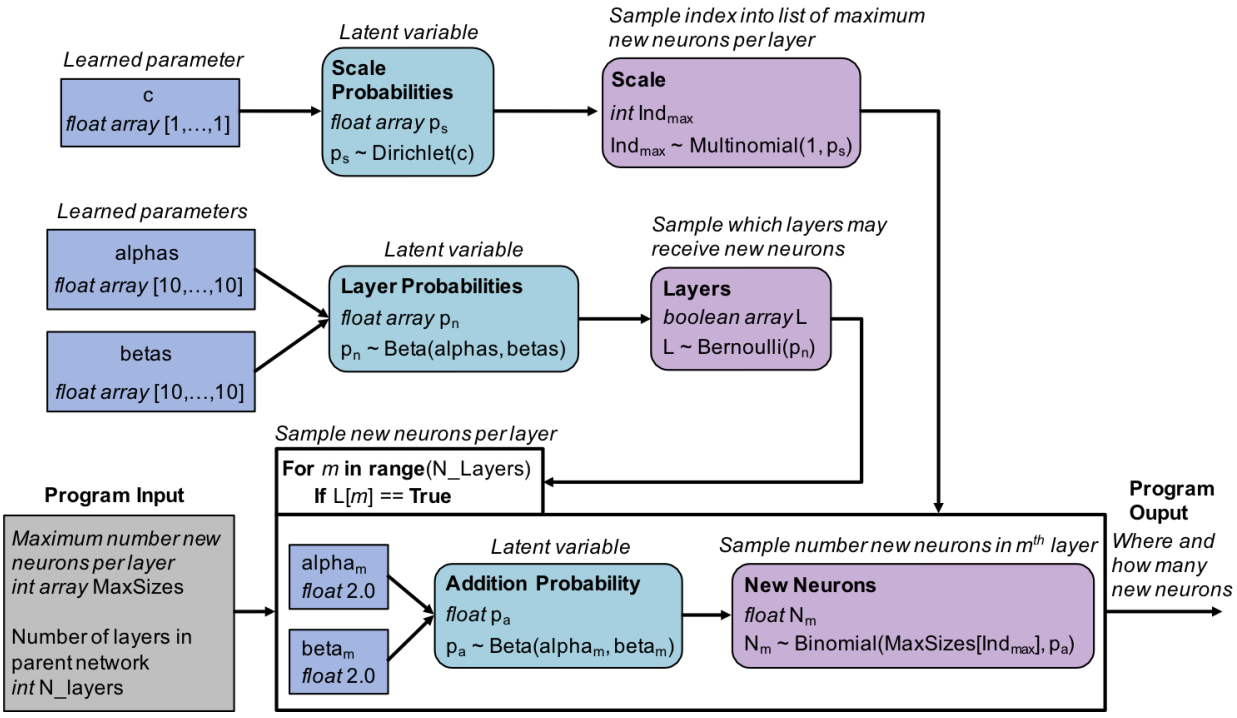


Figure 53: Illustration of the probabilistic program learner used in our approach.

are latent (unobserved) random variables that control the shapes of the probability distributions governing the aforementioned characteristics. These boxes each contain a descriptive name, the data type of the random variable, and the form of the probability distribution from which it is drawn. The dark blue boxes represent learnable parameters, hence the word "learnable" in the acronym PPL. They show the names of the parameters, their data types, and initial values. The program input (gray box) specifies a list of the maximum numbers of new neurons that may be added to any layer and the number of layers in the parent network, both of which are fixed throughout the optimization process.

We now describe the stochastic process of sampling from the PPL to generate a new individual. The sub-process shown at the top of Figure 53 begins by sampling a parameter $p_s \sim Dirichlet(c)$, where c is a learnable parameter. Next, an index is drawn from $Ind_{max} \sim Multinomial(1, p_s)$. This index is used later in the program to access the array MaxSizes. The next sub-process begins by sampling a list of parameters $p_n \sim Beta(alphas, betas)$, where $alphas$ and $betas$ are learnable parameters. The parameters p_n are the individual probabilities of selecting each of the layers in the parent network to receive new neurons. Next, the layers that will receive new neurons are selected

according to $L \sim Bernoulli(p_n)$, where L is a Boolean array. The last sub-process selects how many new neurons to add to each layer $m \in N$. The control logic allows new nodes to be added only to those layers selected in the second sub-process (i.e., $L[m] == True$). The probability of adding a single new neuron to layer m is drawn from $p_a \sim Beta(alpha_m, beta_m)$, where $alpha_m$ and $beta_m$ are learnable parameters. The total number of new neurons to add to the m^{th} layer is sampled from a binomial distribution according to $N_m \sim Binomial(MaxSizes[Ind_{max}], p_a)$, where $MaxSizes[Ind_{max}]$ is the maximum number of new nodes that can be added to any layer. Once the loop is completed, the program output specifies where and how many new neurons to add, which is sufficient to build a new individual.

Learning

In each generation of the optimization process, after selection has occurred, the learnable parameters of the probabilistic program are updated based on the attributes of the selected individuals. We refer to these attributes as observable variables and denote them by x. For each individual, the observables are the maximum number of new neurons per layer (Ind_{max}), the layers that are permitted to receive new neurons (L), and the number of new neurons in the m^{th} layer (N_m). The random variables that are not observable are referred to as latent variables and are denoted by z. The latent variables in our model are p_s, p_n, and p_a, which are described above. For each individual, the learning algorithm updates the parameters (c, $alphas$, $betas$, $alpha_m$, and $beta_m$) so that the probabilistic program is more likely to generate similar individuals in subsequent generations. We used stochastic variational inference as the learning algorithm [175].

Experimental Setup

We applied our approach to a variant of the MNIST optical character recognition problem [147]. Our setup is shown in Figure 54. First, a "parent" neural network was trained to classify gray-scale images of handwritten digits from 0 to 9 (old task). The training dataset consisted of 1,000 randomly selected images with approximately 100 images per class. The parent networks used in our experiments achieved an average testing accuracy of 87.4%. The images were flattened into a 748-dimensional vector. The parent network consisted of three hidden layers with hyperbolic

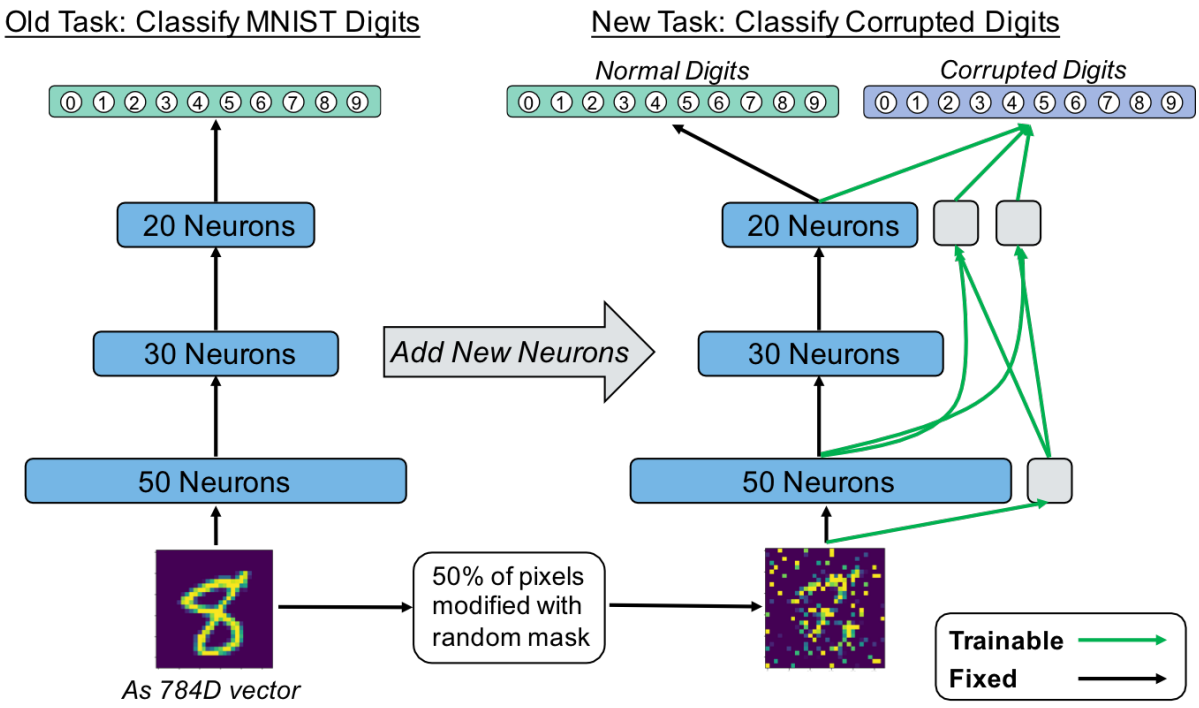


Figure 54: Illustration of our experimental setup for neurogenesis.

tangent (tanh) activation functions. There were 50, 30, and 20 neurons in the first, second, and third hidden layers, respectively.

A new task was created by randomly generating a permutation mask and applying it to each of the images in the dataset. The training dataset for the new task consisted of 1,000 randomly selected, permuted images with approximately 100 images per class. The permutation mask was created by randomly selecting two non-intersecting sets of pixel indices, and then swapping the corresponding pixels in each image. In our experiments, 50% of the pixels in each image were modified. The resulting new task was similar enough to the old task that some information from the parent network was still valid, but different enough that adding new neurons significantly improved performance on the new task. If no new neurons were added, the parent network would experience catastrophic forgetting (loss of performance on the old task) due to the change in the input distribution. We set the maximum number of new neurons that could be added to any layer at 50. This leads to a total of $50^3 = 125,000$ unique individuals.

We set the parameter k in Equation 41 to 10.0. We found that smaller values of k resulted in the complexity term dominating the fitness, which resulted in a fairly simple fitness landscape with the global optimum being achieved by adding only 1-3 neurons at *any* layer. In contrast, setting $k = 10.0$ provided better balance between accuracy and complexity, and consequently, a more challenging optimization problem with many good, but suboptimal, local minima. For this setting, the global optimum is achieved by adding 16 new neurons to the first hidden layer and no new neurons to the second and third hidden layers. However, good, but sub-optimal, local minima can

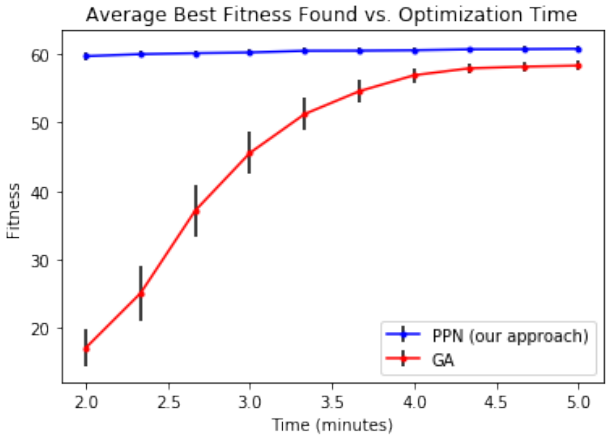


Figure 55: Comparison of the average best fitness found over the first 5 minutes of optimization time.

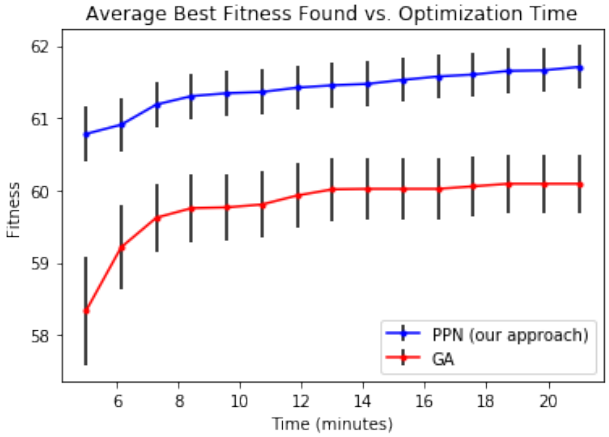


Figure 56: Comparison of the average best fitness found over the next 5 to 21 minutes of optimization time.

be achieved by adding new neurons to only the second or third hidden layers.

We used a genetic algorithm (GA) as basis for comparison with our approach. Genetic algorithms [176] are a good fit to this problem due to the discrete nature of the search space. For the GA, an individual was encoded as a vector of length 3, where the values of the components indicated the number of new neurons to add in each of the three hidden layers. The maximum number of new neurons that could be added to any layer was 50. We used a population size of 30 and tournament selection with a tournament size of 3. Among the selected population, an individual was chosen for crossover with another randomly chosen individual with probability 0.5 and was chosen for mutation with probability 0.2. Once selected for mutation, each entry in the individual was mutated uniformly at random with probability 0.3 to a value in the interval [0, 50]. The relatively high mutation rate was found to prevent pre-mature convergence to poor solutions. We used two-point crossover with the crossover points being selected uniformly at random.

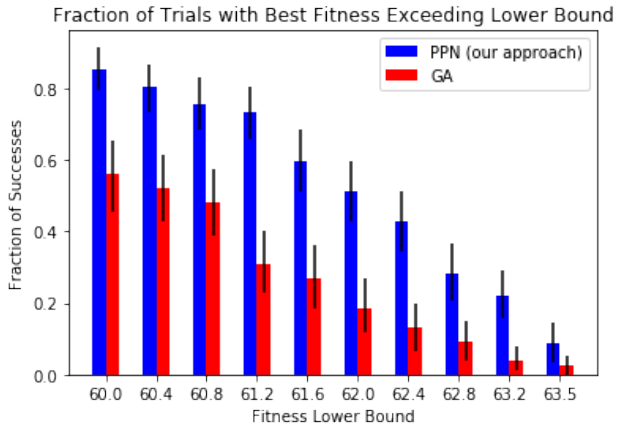


Figure 57: Comparison of our approach with the GA.

1.3.8 Meta-Learned Instinct Network

Following the instinct network architecture introduced in [102], the agent's neural network is divided into two modules (Figure 58). The first module (policy network) has to learn the task, while the second module (instinct network) should learn to modify the main policy's actions when those would likely lead to safety violations.

The instinct network is pre-trained to detect hazard zones and to engage instinctual actions to avoid them. Unlike related work [113], this architecture is suitable for continuous action spaces. The final policy output is determined by:

1. Following the standard way of continuous action exploration in RL [37], the actions of the policy network a^P are noisy; the policy network outputs a mean action a_μ that is given to a distribution (usually the normal distribution) from which the output action is sampled: $a_i^P \sim \mathcal{N}(a_\mu^n, \sigma^n)$, where σ is part of the policy parameters θ^p, and n denotes n^{th} action dimension.

2. The instinct network is aware of the action $\vec{a}_i^P$ as well as the state observation s_i at step i, creating the instinct state observation $s_i^I := \langle s_i, a_i^P \rangle$. This is in contrast to our previous Meta-Learned Instinct Network approach [102], in which the instinct co-evolved to expect what kind of behavior the policy performs around hazards and therefore did not need $\vec{a}_i^P$ as input. In our IR^{2}L approach, the instinct needs to work with a random policy on a task where hazards could be distributed differently than during pre-training; the instinct needs to know what the policy wants to execute so it can modulate it accordingly.

3. The instinct network outputs two instinctual actions: modulation value $m_i \in [0,1]$ and the instinctual action $\vec{a}_i^I \in \mathscr{A}$.

4. The modulation value m_i is multiplied with the policy action vector $\vec{a}_i^P$ giving $\vec{a}_i^{P^*}$.

5. The instinct network action $\vec{a}_i^I$ is multiplied with $1 - m_i$ giving $\vec{a}^{I^*}$. If the policy action is getting suppressed by having m_i close to 0, the instinct action can pass through and vice versa.

The final action $\vec{a}^F$ is the sum of $\vec{a}_i^{I^*}$ and $\vec{a}_i^{P^*}$.

Iterative Training Procedure
Substituting the expansive meta-learning loop in [102], we present a more efficient sequential method where the instinct learns a transferable skill by being exposed to only one task. We perform pre-training in two phases: (a) policy-only pre-training, and (b) instinct-only pre-training (Figure 59).

First, we train a policy on a task without any hazards and transfer the policy to the second phase consisting of the same type of task with added hazards. The policy is frozen, but an instinct is introduced that has to learn to prevent the policy from colliding with hazards. We consider the second phase to be a situation in which the agent can afford to commit many hazard violations during training (e.g., training in a simulator). The hypothesis is that after this pre-training phase, safe learning from a random policy in safety-critical domains should be possible by combining this policy with the hazard-avoiding skills of the instinct. We assume that the hazard observations are invariant between all tasks. In more detail, the pre-training phases include:

Phase 1: Policy-only pre-training without hazards
We train a policy network without an instinct component to solve a task without any hazards in the environment. The purpose of this phase is to have a policy that can perform the task well but not safely. When transferred to the instinct pre-training phase, it should help the instinct to collect relevant hazard experiences.

Phase 2: Instinct pre-training
The second pre-training phase introduces hazards to the task used in the first pre-training phase. The policy from the first pre-training phase is expected to do frequent hazard collisions and thus provide abundant experiences for an instinct to learn to avoid. During this phase, the policy is fixed to the policy found in the first pre-training step while the instinct is randomly initialized. Unlike during later training of the policy during transfer to test tasks ("Training" column in Figure 59) in which a random policy executes stochastic actions and the instinct deterministic ones, here the instinct executes stochastic actions and the policy executes deterministic ones. The goal of the second phase is to train an instinct that stops an agent from colliding with hazards while still giving the policy enough flexibility to reach its goal. To achieve this task, we designed the following instinct reward function:

$$r_i(s_i, \vec{m}_i, r_t, h) = (1 - h(s_i)H)m_i r_t(s_i)D, \tag{42}$$

where s_i is the current state observation, m_i is the instinct suppression value, r_t is the reward given for getting closer to the goal. $h(s_i) = 1$ if the agent collided with a hazard in the i-th step, otherwise $h(s_i) = 0$. The first term $1 - h\,(s_i)H$ models the safety, and is 1 if the agent is s afe. If the agent collided with a hazard, the term becomes smaller for $H > 0$. If the hyperparameter $H > 1$ the safety term will become negative and the instinct reward will be negative overall. The hyperparameter H controls how harsh the instinct will be punished for not preventing collisions with hazards and, as a result, how conservative it will be in the end.

As a reminder, $0 < m_i < 1$, where $m_i = 1$ means that only the policy is controlling the agent, and $m_i = 0$ means that the instinct took complete control. The smaller m_i is, the smaller is the instinct

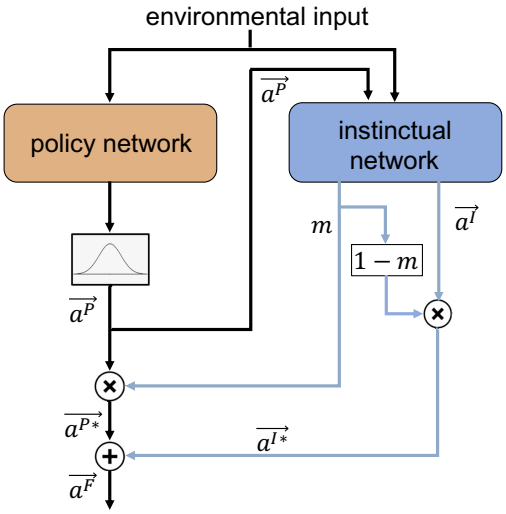


Figure 58: The topology of the policy network with the instinct module.

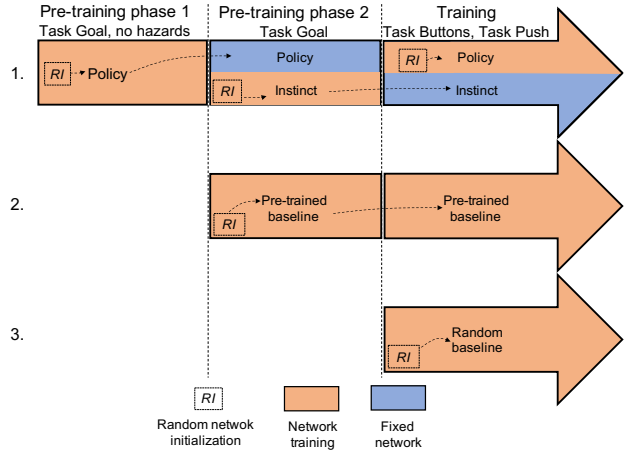


Figure 59: Pre-training and training procedures for IR^{2}L and two baselines.

reward, discouraging the instinct from activating when not necessary. Since the instinct works in combination with a policy that is efficient in reaching the goal, the goal-reaching reward r_t is there to maximize the reward when the instinct is not active. We would like to discourage the instinct from doing anything if the agent operates within safe parts of the environment, activating only if the agent is in danger of colliding with a hazard. Hyperparameter D is there to amplify the contribution of r_t. We performed a hyperparameter search on H and D and found that $H = 100$ and $D = 15$ work best for our experiments. The instinct was trained for 300 epochs and in each epoch, 215 trajectories were sampled.

Task Environment

We tested our approach on the OpenAI Safety Gym environment developed exactly for studying reinforcement learning for safe exploration [97]. The framework provides a variety of tasks, agents, and obstacles that can be easily rearranged and modified and is built upon the MuJoCo physics

engine for robotics simulations.

The environment allows three different task types: “goal”, “push”, and “buttons” (Figures 62 to 64). In the “goal” task type, the agent (red body) has to reach a green cylinder randomly spawned in the environment. In the “buttons” task type there are several buttons (orange spheres) where the agent needs to learn to press the correct button. The most complex“push” task type requires the agent to push a yellow box in the green cylinder. The agent is challenged by a variety of obstacles that generate cost if the agent collides with them. We focused only on static “hazard zones” (blue circles) that the agent can cross but that generate a cost for every step the agent spends within one of them.

The agent is equipped with a set of LIght Detection And Ranging (LIDAR) sensors for detecting environment elements. There is a separate set of LIDARs for each element (colored crowns above the agent in Figure 62). Each LIDAR set has 16 LIDARs distributed around the agent at equal angles. They are represented as a vector with each LIDAR observation in the range [0, 1], 0 when the target is not visible and 1 when the observed object is adjacent to the agent. Furthermore, the agent has access to its orientation relative to the north and the central point of the map.

The agent is a floating dot that only moves on a two-dimensional surface. The available actions are represented by a two-dimensional vector where the first dimension represents backward/forward movement, while the second dimension represents left/right turning movement. In the original Safety Gym setup, the agent sees the goal object and the correct button with the same LIDAR set. Thus, a policy trained to follow the green cylinder in the “goal” task type, would perfectly transfer to the “buttons” task type. To make it more challenging for the agent, we modified the original Safety Gym code to separate the LIDAR set into two different sets, one for “buttons” and one for “goal” task types. We also modified the framework to allow different LIDAR ranges for hazard detection and for task elements (box, cylinder, button). If the original range was too short, the agent could not see goals too far away. If the LIDAR range was too long, the agent would get over-saturated with hazards LIDAR inputs in case there were a lot of hazards present in the environment. We modified the original source code to allow for short LIDAR ranges when detecting hazards, and long-range when detecting other elements. The episode length of all implemented tasks is equal to 1,000 steps ($s_T = s_{1,000}$).

“Goal” Task

Here the agent has to reach the green cylinder randomly placed on the map (Figure 62). The agent is spawned in the center of the map at the start of each episode. If the agent reaches the goal before the episode finishes, the goal is placed at a different location on the map. There are other elements on the map (yellow box and orange buttons) but they are irrelevant to this task, so the agent needs to learn to ignore them. The hazards are distributed in a static 5×5 grid, with the central hazard missing, equaling to 24 hazards in total. Since this task is used only in the instinct pre-training phase, we wanted to maximize the opportunities where the agent can collide with the hazard while still being able to learn to reach the goal.

The reward is defined as the negative difference between the agent-goal distance in this step and in the previous step. If the agent came closer to the goal, the reward is positive. Otherwise, the reward

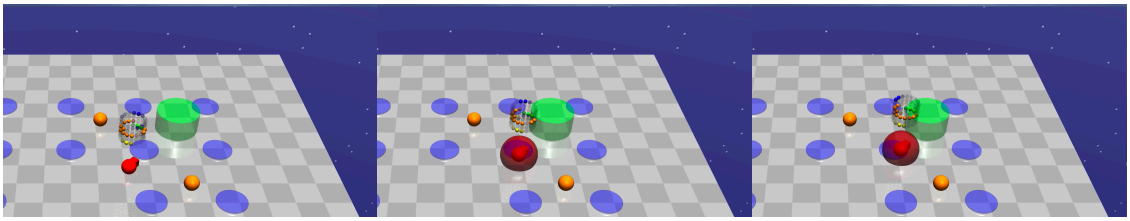


Figure 60: A sequence of frames showing the pre-trained policy going over a hazard in the "goal" task.

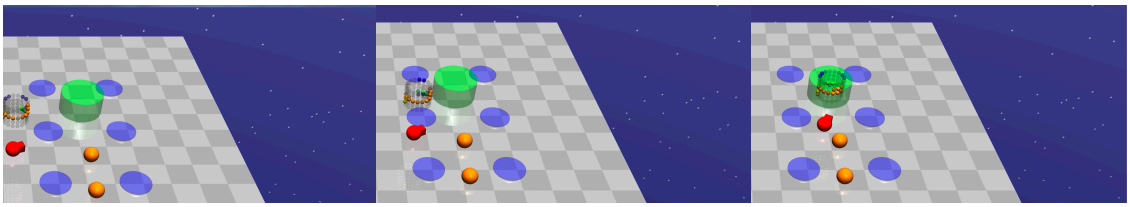


Figure 61: Coordination of the policy and instinct networks to avoid hazards in the "goal" task.

is negative. For each step that the agent spends in a hazard zone, the environment hazard function $h(s)$ returns 1, otherwise, it returns 0. The behavior of the policy trained without hazards and the same policy after we added the pre-trained instinct network can be seen in Figures 60 and 61.

"Buttons" Task

The agent is spawned in the center and has to press the correct button out of four randomly positioned buttons (Figure 63). All four buttons are detectable through one LIDAR set, and the correct button is visible through a separate LIDAR set. The buttons are fixed throughout the episode and the next correct button is randomly chosen as soon as the agent presses the current correct one.

There are eight randomly positioned hazards at the beginning of an episode. This is the first task used to test the hazard avoidance capabilities of the instinct network. We would want the agent to learn to press the correct buttons while avoiding the hazards during the training phase. The reward is the measure of how much closer the agent is to the correct button since the last step. The $h(s)$ function communicates whether the agent is stepping over a hazard.

"Push" Task

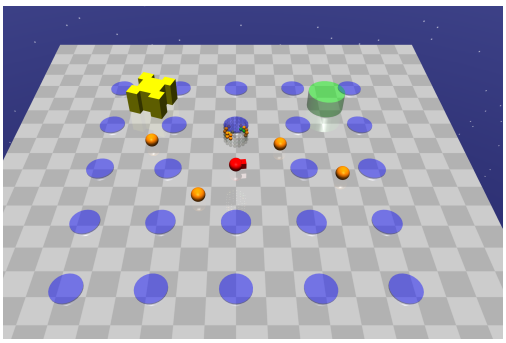


Figure 62: Depiction of the "goal" task in the Safety Gym environment.

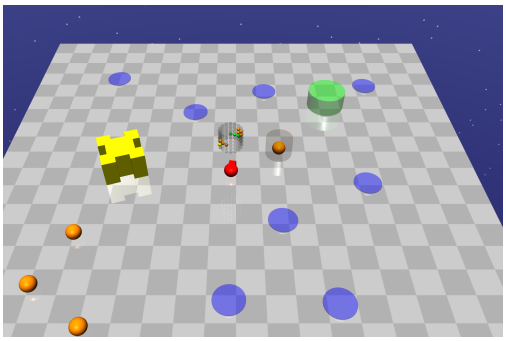


Figure 63: Depiction of "buttons" task in the Safety Gym environment.

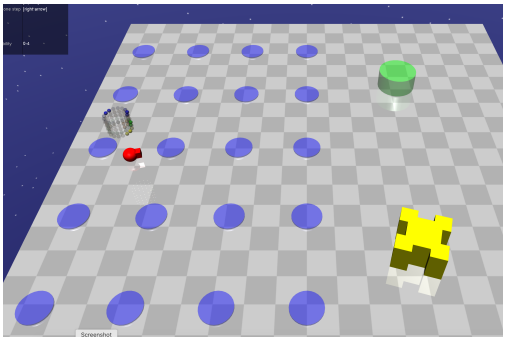


Figure 64: Depiction of "push" task in the Safety Gym environment.

The goal of the agent is to push the yellow box into the goal (green cylinder) (Figure 64). The agent gets a reward for getting close to the yellow box and another reward for closing the distance between the box and the goal. The two rewards are summed, resulting in the final step reward. The hazards are spawned in a 5×4 grid on the left-hand side, while the box and the goal are located on the right-hand side of the map. The agent is always spawned just at the left of the hazards grid.

The hazards are fixed in a grid layout while the goal and the box are randomly spawned in their area. The reasoning for this layout was that in an environment where hazards, box, goal, and the agent are uniformly distributed across the map, it is extremely difficult for the agent to push the box around the hazards. For that reason, we made it easier for the agent to solve the task by decoupling the obstacles from the task. The agent needs to clear the hazards grid to solve the task. The environment hazard function $h(s)$ communicates hazard violations.

Training Details

Network Implementation

For the policy and instinct networks, we use an advantage actor-critic system [177], where actor and critic are two separate, fully connected neural networks with three hidden layers of 512 neurons each and tanh activation functions. The policy gradient in the advantage actor-critic system can be described as:

$$\hat{g}(\theta) = E_\theta[\nabla_\theta \log f_\theta(s,a) A_\theta(s)], \tag{43}$$

where $A_\theta(\cdot,\cdot)$ is the advantage calculated from the critic and $f_\theta(\cdot,\cdot)$ is the output of the actor-network. The critic-network is updated to minimize the temporal difference between predicted

expected return $A_\theta(s_t)$ at state s_t, and the reward $R(s_t)$ updated return estimate: $A_\theta(s_t) - (R(s_t) + \gamma A_\theta(s_{t+1}))$, where γ is the reward discount hyperparameter [178, 179].

The actor outputs a mean action for a Gaussian distribution $\mathcal{N}(\vec{a}_\mu, \vec{\sigma})$, from which an action is sampled [37]. The critic outputs the predicted value (predicted future cumulative reward). The final layer of the plastic policy's actor-network has two outputs scaled to $[-0.1, 0.1]$, reflecting the Safety Gym Point agent's action space. The instinct actor-network outputs three outputs; two are the instinct actions reflecting the agent action space and the last one is the instinct modulation signal m. The modulation signal is then scaled to fit the $[0, 1]$ range. We clip the modulation signal to $[0, 1]$ range in case sampling from a Gaussian distribution $\mathcal{N}(\vec{a}^I_\mu, \vec{\sigma}^I)$ takes it outside the range.

RL Training

The weights of all networks implemented are initialized with Kaiming uniform initialization [180]. Gaussian action noise parameter σ is initialized to 0.6 and the learning rate is set to 0.001. In each training epoch, the sample buffer collects 216,000 state-action-reward samples which are equal to 216 trajectories/episodes. An existing PPO implementation from [181] served as a starter code for our implementation of the method. There is a set of PPO hyperparameters needed for training: γ discount factor (0.99), PPO clip parameter (0.2), PPO epoch number (4), value loss coefficient (0.5), and entropy term coefficient (0.01). This configuration was used for both pre-training phases and experiment task adaptation.

1.3.9 Plastic Neuromodulated Network

Problem Formulation

In the meta-RL setting, tasks are sampled from a task distribution $p(\mathcal{T})$. Each task $\mathcal{T}_i$ is a MDP, which is a tuple M_i=$\{\mathcal{S}, \mathcal{A}, q, r, q_0\}$ consisting of a state space $\mathcal{S}$, an action space $\mathcal{A}$, a state transition distribution $q(s_{t+1}|s_t, a_t)$, a reward function $r(s_t, a_t, s_{t+1})$, and an initial state distribution $q_0(s_0)$. When presented with a task $\mathcal{T}_i$, an agent (with a policy π) is required to quickly adapt to the task from few interactions. Therefore, the goal of the agent for each task is to maximize the expected reward in the shortest time possible:

$$\mathcal{J}(\pi) = \mathbf{E}_{q_0,q,\pi}\left[\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t, s_{t+1})\right], \tag{44}$$

where H is a finite horizon and $\gamma \in [0, 1]$ is the discount factor.

CAVIA

The CAVIA meta-learning framework [122] is an extension of the Model-Agnostic Meta-Learning algorithm [82] that is interpretable and less prone to meta-overfitting. The key idea in CAVIA is the introduction of context parameters in a policy network. Therefore, the policy $\pi_{\theta,\phi}$ contains the standard network parameters θ and the context parameters ϕ. During the adaptation phase for each task (the gradient updates in the inner loop), only the context parameters are updated, while the network parameters undergo updates during the outer loop. There are different ways to provide the policy network with the context parameters. In [122], the parameters were concatenated to the input.

In the meta-RL framework, an agent is trained for a number of iterations. For each iteration, N tasks represented as $\mathbf{T}$ are sampled from the task distribution $\mathscr{T}$. For each task i, a batch of trajectories τ_i^{train} is obtained using the policy $\pi_{\theta,\phi}$ with the context parameters set to an initial condition ϕ_0. The obtained trajectories for task i are used to perform a one-step inner loop gradient update of the context parameters to new values ϕ_i, shown in the equation below:

$$\phi_i = \phi_0 - \alpha \nabla_\phi \mathscr{J}_{\mathscr{T}_i}(\tau_i^{train}, \pi_{\theta,\phi_0}), \tag{45}$$

where $\mathscr{J}_{\mathscr{T}_i}(\tau_i, \pi_{\theta,\phi})$ is the objective function for task i. After the one-step gradient update of the policy, another batch of trajectories τ_i^{test} is collected using the updated task-specific policy π_{θ,ϕ_i}.

After completing the above procedure for all tasks sampled from $\mathscr{T}$, a meta-gradient step (also referred to as the outer loop update) is performed, updating θ to maximize the average performance of the policy across the task batch.

$$\theta = \theta - \beta \nabla_\theta \frac{1}{N} \sum_{\tau_i \in \mathbf{T}} \mathscr{J}_{\mathscr{T}_i}(\tau_i^{test}, \pi_{\theta,\phi_i}). \tag{46}$$

PEARL

PEARL [123] is an off-policy meta-RL algorithm that is based on the soft actor-critic architecture [182]. The algorithm derives the context of the task to which an agent is exposed through probabilistic sampling. Given a task, the agent maintains a prior belief of the task, and as the agent interacts with the environment along trajectories $\mathbf{c}$, it updates the posterior distribution with the goal of identifying the specific task context. The context variables $\mathbf{z}$ are concatenated to the input of the actor and critic neural networks of the setup. To estimate this posterior $p(\mathbf{z}|\mathbf{c})$, an additional neural component called an inference network $q_\phi(\mathbf{z}|\mathbf{c})$ is trained using the trajectories $\mathbf{c}$ collected for tasks sampled from the task distribution $\mathscr{T}$. The objective functions for the actor, critic, and inference neural components are described below,

$$\mathscr{L}_{actor} = \mathbb{E}_{\substack{\mathbf{s}\sim\mathscr{B},\mathbf{a}\sim\pi_\theta \\ \mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{c})}} \left[D_{\mathrm{KL}} \left(\pi_\theta(\mathbf{a}|\mathbf{s},\bar{\mathbf{z}}) \middle\| \frac{\exp(Q_\theta(\mathbf{s},\mathbf{a},\bar{\mathbf{z}}))}{\mathscr{Z}_\theta(\mathbf{s})} \right) \right] \tag{47}$$

$$\mathscr{L}_{critic} = \mathbb{E}_{\substack{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\sim\mathscr{B} \\ \mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{c})}} [Q_\theta(\mathbf{s},\mathbf{a},\mathbf{z}) - (r + \bar{V}(\mathbf{s}',\bar{\mathbf{z}}))]^2 \tag{48}$$

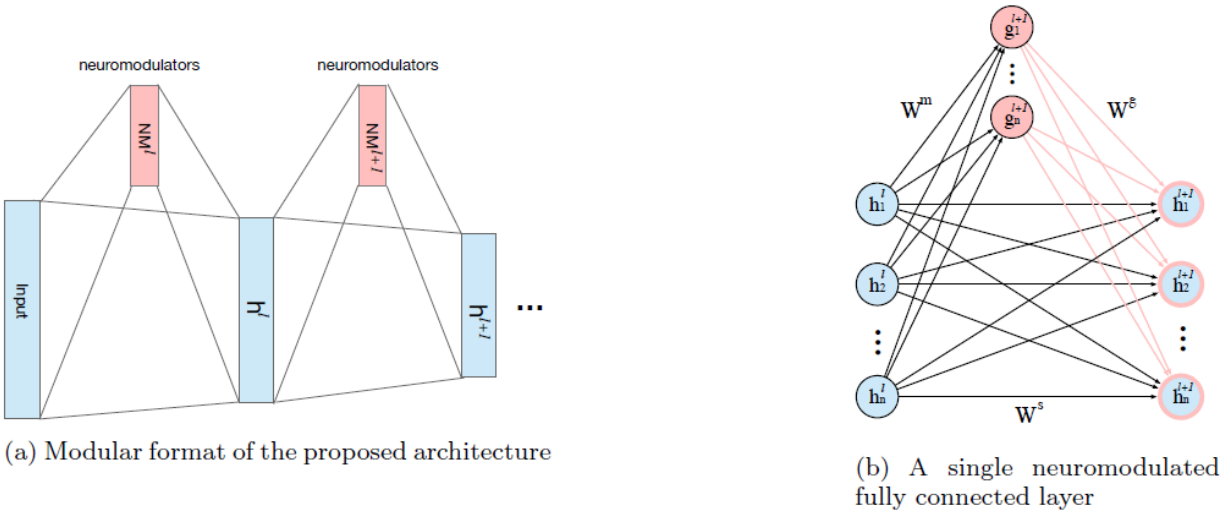
$$\mathscr{L}_{inference} = \mathbb{E}_{\mathscr{T}} [\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{c}^{\mathscr{T}})} [\mathscr{L}_{critic} + \beta D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{c}^{\mathscr{T}}) || p(\mathbf{z}))]] \tag{49}$$

where $\bar{V}$ is a target network and $\bar{\mathbf{z}}$ means that gradients are not being computed through it, $p(\mathbf{z})$ is a unit Gaussian prior over Z, $\mathscr{B}$ is the replay buffer, and β is a weighting hyperparameter.

Neuromodulated Network

Here we introduce the extension of the policy network with neuromodulation. A graphical representation of the network is shown in Figure 65(a). The neuromodulated policy network is a stack of neuromodulated fully connected layers.

A neuromodulated fully connected layer contains two neural components: standard neurons and neuromodulators (Figure 65(b)). The standard neurons serve as the output of the layer (i.e., the



(a) Modular format of the proposed architecture

(b) A single neuromodulated fully connected layer

Figure 65: Overview of the proposed computational framework.

layer's representations) and are connected to the preceding layer via standard fully connected weights W^s. The neuromodulators serve as a means to alter the output of the standard neurons. They receive input via standard fully connected weights W^g from the preceding layer in order to generate their neural activity, which is then projected to the standard neurons via another set of fully connected weights W^m. The function of the projected neuromodulatory activity defines the representation altering mechanism. For example, it could gate the plasticity of W^s, gate neural activation of $\mathbf{h}$ or do something else based on the designer's specification. While different types of neuromodulators can be used [125], in this particular work, we employ an activity-gating neuromodulator. Such neuromodulator multiplies the activity of the target (standard) neurons before a non-linearity is applied to the layer. Formally, the structure can be described with three parameter matrices: W^s defines weights connecting the input to the standard neurons, W^g defines weights connecting the input to the neuromodulators and W^m defines weights connecting the neuromodulators to the standard neurons. The step-wise computation of a forward pass through the neuromodulatory structure is given below:

$$\mathbf{h}^s = W^s \cdot \mathbf{x} \tag{50}$$

$$\mathbf{g} = ReLU(W^g \cdot \mathbf{x}) \tag{51}$$

$$\mathbf{h}^m = tanh(W^m \cdot \mathbf{g}) \tag{52}$$

$$\mathbf{h} = ReLU(\mathbf{h}^s \otimes \mathbf{h}^m) \tag{53}$$

where $\mathbf{x}$ is the layer's input, $\mathbf{h}^s$ is the weighted sum of input of the standard neurons, $\mathbf{g}$ is the activity of the neuromodulators derived from the weighted sum of input, $\mathbf{h}^m$ is the neuromodulatory activity projected onto the standard neurons, and $\mathbf{h}$ is the output of the layer. The key modulating process takes place in the element-wise multiplication of the $\mathbf{h}^s$ and $\mathbf{h}^m$.

The tanh non-linearity is employed to enable positive and negative neuromodulatory signals, and thus gives the network the ability to affect both the magnitude and the sign of target activation values. When ReLU is used as the non-linearity for the layer's output h, $\mathbf{h}^m$ has the intrinsic ability

to dynamically turn on or off certain outputs in $\mathbf{h}$. A simpler version of the proposed model can be achieved by only considering the sign, and not the magnitude, of the neuromodulatory signal, using the following variation of Equation 53:

$$\mathbf{h} = ReLU(\mathbf{h}^s \otimes \mathbf{sign}(\mathbf{h}^m)) \tag{54}$$

This variation is shown to be suited for discrete control problems.

1.4 Results and Discussion

1.4.1 MOdulated Hebbian Network

Here we report the analysis of how (i) learning mechanisms in MOHN compare to those of REINFORCE and DQN, (ii) MOHN and the new loss function enhance the features from DQN to solve the POMDP problems, and (iii) MOHQA compares against DQN, QRDQN+LSTM, REINFORCE, A2C, and AMRL in the CT-graph and Malmo benchmarks.

Comparison of MOHN Learning to DQN and REINFORCE
Here MOHN's learning mechanisms, (i) Hebbian learning, (ii) eligibility traces, and (iii) rare correlations, are contrasted against two other classical learning methods, (i) TD learning in the form of DQN and (ii) policy gradient in the form of REINFORCE. For this comparison, we train a one-layer network with each of the techniques and we remove the feature extraction part. Then, we feed a high-level one-hot feature space to one layer networks for the sole purpose of analyzing the learning of the head modules. To be exact, observations are represented by one-dimensional vector of size k on two different scenarios: (i) with one element equal to 1 and all others equal to 0 to show the importance of eligibility traces and high learning rates and (ii) with one element equal to 1 and all others assigned a random value using 10% uniformly distributed random noise to show the importance of rare correlations. The singular element represents the unique observation. Moreover, the training is performed in an imitation learning-like fashion, where an agent is led to the goal directly in each episode by an all-knowing oracle.

Importance of eligibility traces and high learning rates for quick learning
Figure 66 shows the decision made at each state at episodes 2, 5 and 10 for DQN, REINFORCE, and MOHN. The agents were guided to the goal manually for each episode to see how quickly they will learn the correct path. The correct sequence of actions is to take action 0 in wait state, action 1 in the first decision point and action 2 in a second decision point. As can be seen, MOHN is the fastest, while DQN is struggling to reach good decisions in a few episodes. This ability is vital in sparse reward problems as a reward is encountered rarely and needs to be utilized efficiently.

Importance of rare correlations for coping with noisy inputs
Figure 67 shows the same comparison between three methods, this time with added noise 10% to the inputs. In this scenario, only MOHN was able to learn correct decisions in 10 episodes. This demonstrates another key mechanism of MOHN: utilization of rare correlations. This experiment shows that MOHN exploits rare correlations to make associations between key observations, actions and rewards.

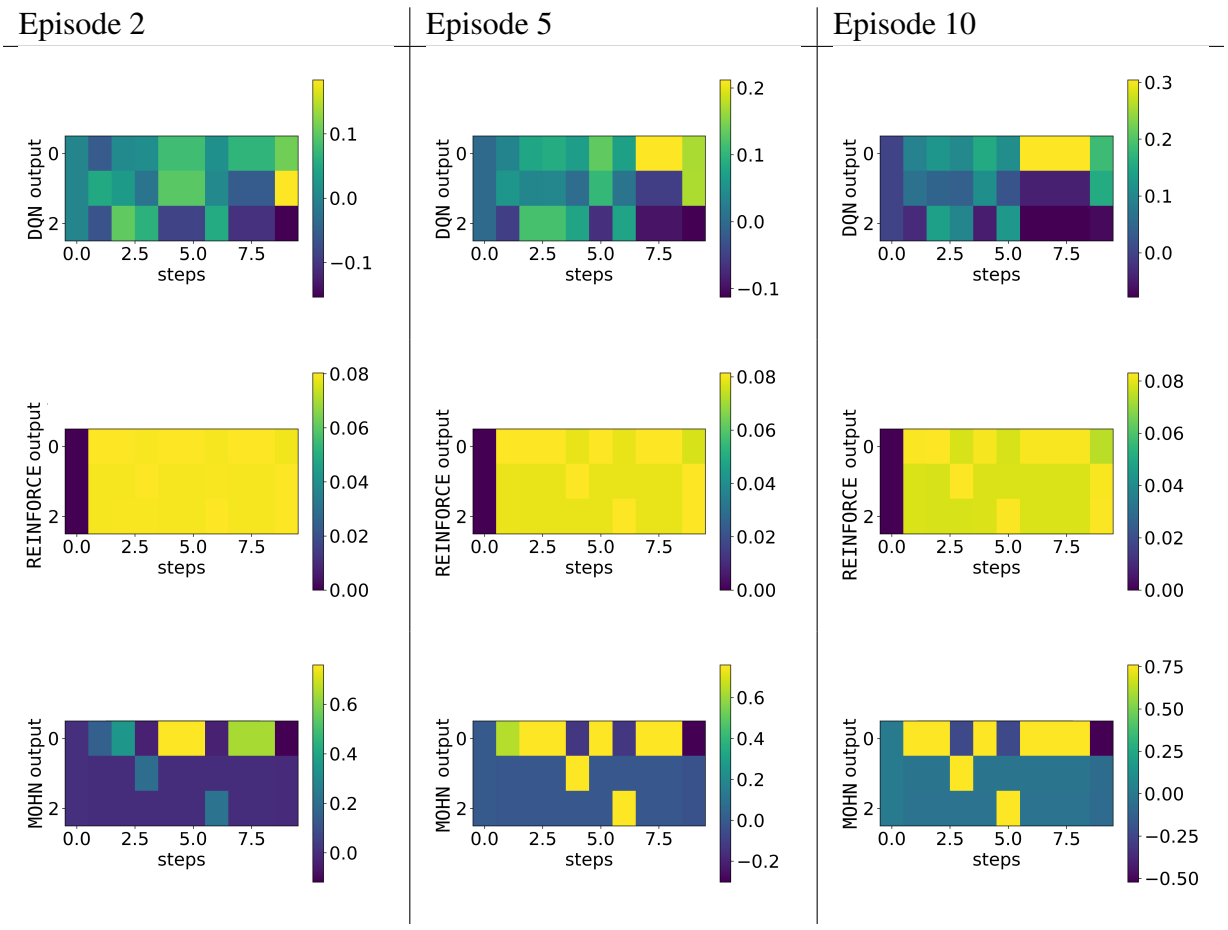


Figure 66: Decision made by various algorithms at different episodes.

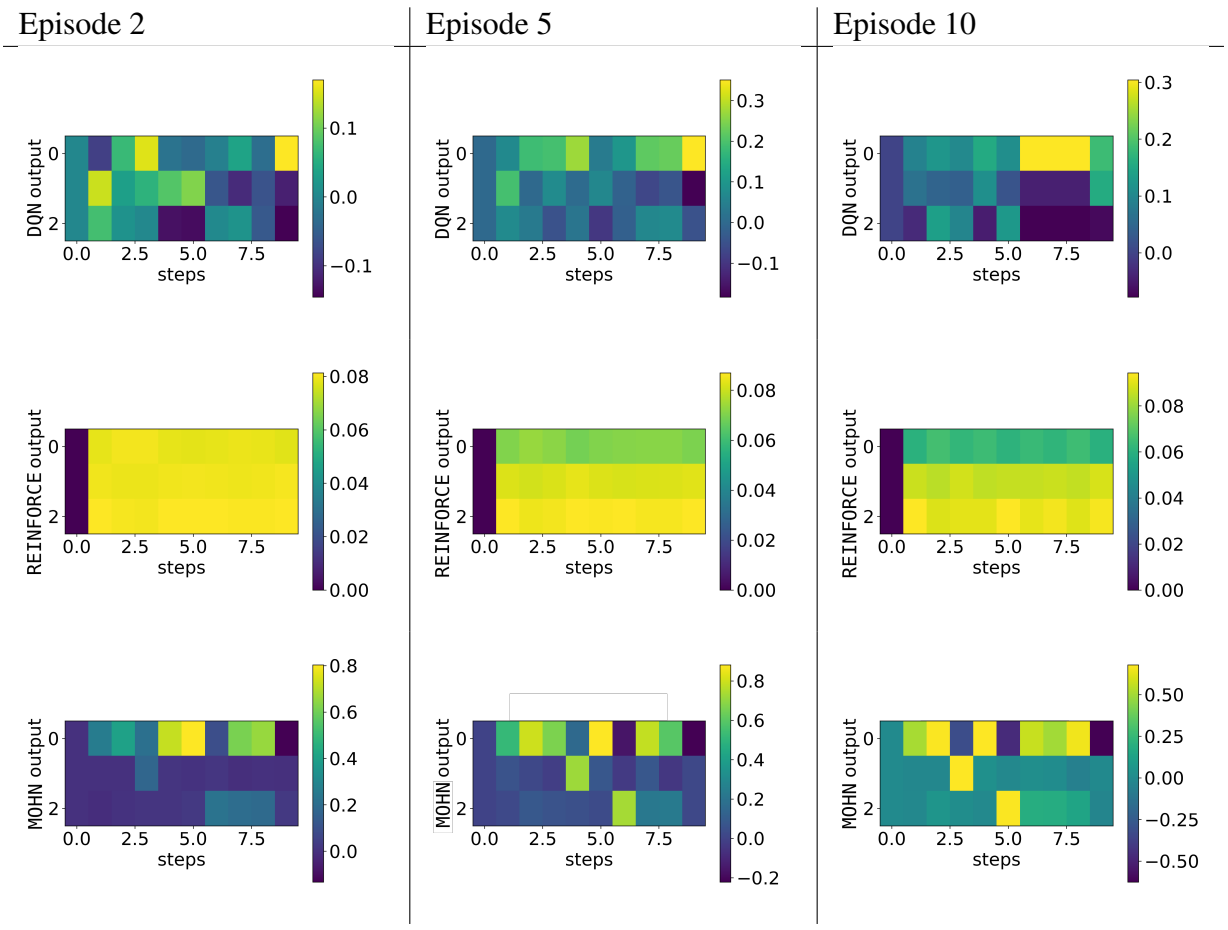


Figure 67: Decisions made by various algorithms at different episodes in a problem with noise.

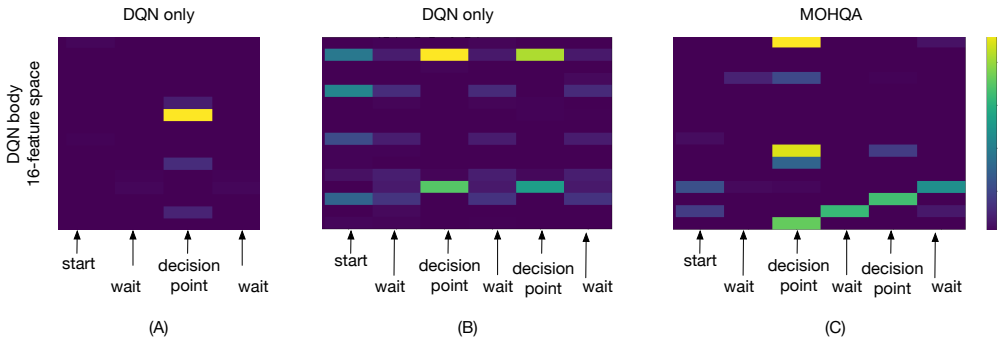


Figure 68: Features learned by DQN without and with MOHQA.

Analysis of Feature Learning

To better understand the need for a new loss function $L(\Phi)$ (Equation 26) and how it affects feature learning (v_b from Figure 8), three experiments are performed (Figure 68): (A) in a one-decision-point CT-graph, (B) in a two-decision-point CT-graph with confounding observations, both using the standard DQN's loss function, and finally (C) a two-decision-point CT-graph with confounding observations and the newly proposed loss function. Each of the snapshots of learned feature space were taken after the agent started consistently scoring maximum reward.

Figure 68(A) shows the learned feature space output throughout one episode. As expected, DQN learns similar high-level features from different observations if those require the same action. Wait states that require the wait-action a^0 are distinguishable from decision points that require act-actions.

A similar situation is observed in a longer CT-graph with two decision points and confounding observations Figure 68(B). In this case, the two decision points had unique observations, thus making the problem observable, but only at decision points. DQN learns two distinct features spaces one for wait states and one for decision points. This is reasonable because these two state types require either action a^0 (wait state) or actions a^1 or a^2 in the decision point. However, the network is unable to distinguish between two decision points due to issues with propagating TD error through confounding wait states. This confusion between the first and the second decision point highlights a problem DQN faces when trying to solve confounding POMDPs: if DQN cannot learn the path to the reward, it cannot also learn the separate features that would enable correct decisions.

Finally, in the third experiment, same environment is used as in Figure 68(B), but the newly proposed loss function is utilized instead of the standard DQN's one. In Figure 68(C) the feature space clearly shows a difference between the first and second decision p oint. MOHN, by suggesting optimal actions to DQN, was able to also lead DQN to learn different features for different decision points, which DQN alone could not achieve. In this last test, the output values reveal the inner working of MOHQA: DQN suggests the wait-action at wait states, and expresses equal preferences for both act actions a^1 and a^2. MOHN contributes by biasing the decision towards the act-action (either a^1 or a^2) that is associated with the future reward.

Computational Speed and Memory Comparison

To analyze the computational effects of adding MOHN to DQN, MOHQA and DQN are compared

in terms of computational speed and GPU memory usage. Both DQN and MOHQA are run for 100,000 episodes on the same machine three times to minimize any variability in performance. The results of these runs are summarized in Tables 5 and 6.

It can be seen that MOHQA is about 66% slower than the original DQN implementations. This is mostly due to operations in MOHN. In general backpropagation has been extensively studied and improved over time and it has efficient libraries written for it. On the other hand the learning mechanism in MOHN is not currently optimized. In particular, some operations such as finding maximum/minimum correlations are very computationally expensive. Comparing memory, a small increase of a GPU memory usage can be attributed to the extra layer and associated eligibility traces storage.

Table 5: Computational cost of DQN.

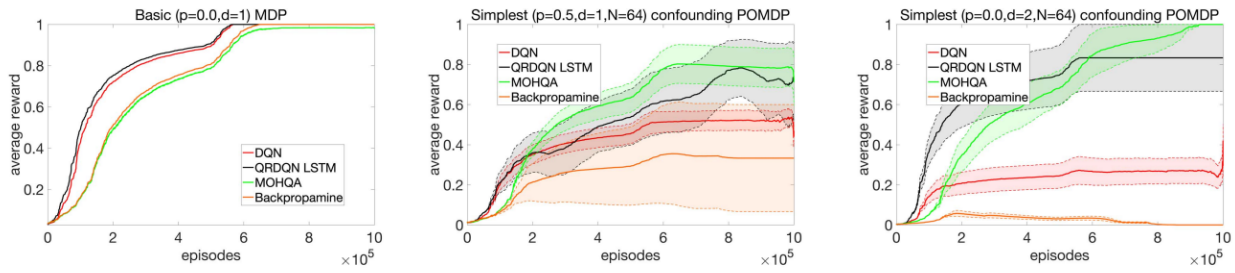
Time (s)	Steps	Time per step (ms)	GPU memory (per step)
419.51	278477	1.51	971
459.85	309650	1.49	970
463.97	309135	1.50	971

Table 6: Computational cost of MOHQA.

Time (s)	Steps	Time per step (ms)	GPU memory (per step)
669.86	277243	2.42	1027
627.70	257465	2.44	1027
645.41	262397	2.46	1026

Comparisons in the CT-Graph

Simulations were performed for a range of CT-graphs and compared with TD-learning approach DQN; three memory based approaches: classical QRDQN+LSTM, Backpropamine and AMRL; policy based approach REINFORCE; and hybrid of policy and TD-learning A2C. It is worth noting that AMRL is the most modern approach to compare against and it was developed to cope with similar type of POMDPs requiring memory. Backpropamine was implemented similarly to MOHN only on a final layer of the network. This was done for two reasons. First, memory modules are normally implemented only on a final layer [12]. Second, in their original implementation [30, 31] the test in RL settings was done on the high-level features. Six simulations with different seeds were run and averaged for each algorithm on each of the CT-graph experiments. Memory in QRDQN+LSTM and Backpropamine was truncated at the end of each episode, while in AMRL truncation happened at the end of the iteration. We tuned all hyper parameters manually to the best performance of each baseline. In general we split benchmarks in the CT-graph into two (rather arbitrary) categories: simple and complex, where simple are ones which most approaches can solve, while complex are ones where algorithms start to struggle.



(a) (b) (c)

Figure 69: Performance on the simpler versions of the CT-graph.

Simple CT-Graph Problems

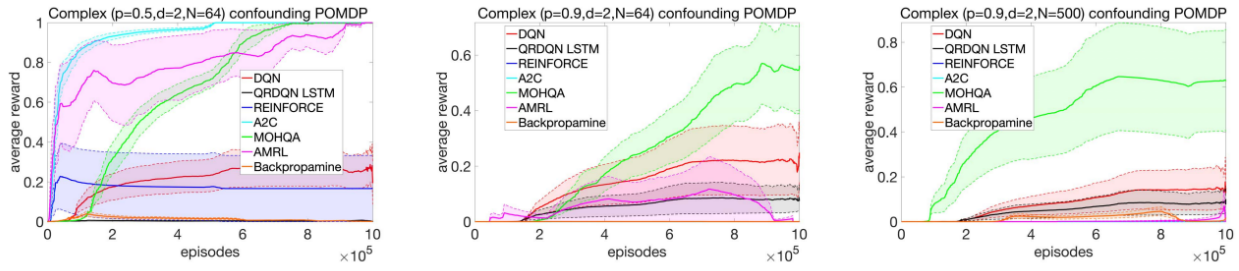
We run the simplest MDP version of the one-decision-point CT-graph (Figure 69(a)) to verify the correctness of implementations. In Figure 69(b), the confounding observations are introduced to the CT-graph by removing uniqueness from the wait states. MOHQA and other approaches are able to solve the problem while DQN starts to struggle as it cannot learn correct actions. Figure 69(c) shows the results on the simplest of two decision point CT-graphs with delay probability of ($p = 0$) and depth of ($d = 2$). This figure shows further advantage of MOHQA over DQN. Note that the history of observations is still relatively simple as histories repeat quite often, thus the good performance of QRDQN+LSTMs.

Complex CT-Graph

For the complex CT-graph problems, we compared MOHQA against A2C, REINFORCE and AMRL. All tests are performed with depth ($d = 2$) and three different configurations: (i) $p = 0.5$, $N = 64$ (Figure 70(a)), (ii) $p = 0.9$, $N = 64$ (Figure 70(b)), and (iii) $p = 0.9$, $N = 500$ (Figure 70(c)), where N is the cardinality of the set of confounding observations and p is the delay probability. The two-decision-point CT-graph (Figure 70(a)) shows very similar trends to the simple CT-graph problems, with MOHQA still improving on performance of DQN, achieving similar performance to memory based AMRL. QRDQN+LSTM starts to struggle due to significantly increased number of possible histories of observations. Figure 70(b) shows deterioration of performance of A2C, QRDQN+LSTM, REINFORCE and DQN, to the point where MOHQA and AMRL show significant advantage of performance. Finally, in the most complex CT-graph (Figure 70(c)), MOHQA outperforms all other tested algorithms. Wide confidence intervals are a consequence of the 'all or nothing' reward structure. This means that agents either tend to know how to solve it and score 1 or they don't know and they score zero. Thus, a run that scored optimally in five out of six seeds reports an average normalized fitness of 0.8 with a standard deviation of ± 0.14.

Comparison in Malmo Benchmark

The Malmo benchmark was configured as a maze with two decision points (Figure 4). An agent requires 30 steps to get from the home location to the end of the maze. Six simulations with different seeds were run and averaged. Figure 71 shows that MOHQA was able to achieve a higher average score than all other baselines. The result shows that the advantage achieved by MOHQA in the previous benchmarks is transferable to different domains.



(a) (b) (c)

Figure 70: Performance on a more complex CT-graph.

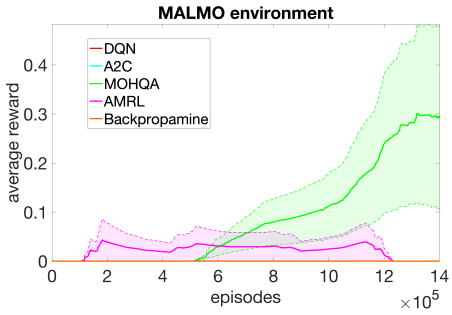


Figure 71: Comparison of results in Malmo benchmark.

The confounding POMDP has proven to be a difficult problem for DQN, QRDQN+LSTM, REINFORCE, A2C and AMRL. The addition of the MOHN module to the standard DQN allowed MOHQA to achieve higher average scores than standard DQN. Such an advantage is achieved by the ability of MOHN to distinguish important decision states from confounding wait states and bridge the temporal gap between state-action pairs and rewards. REINFORCE and MOHQA appear to have very similar weight update scheme, yet MOHQA shows very clear advantage over REINFORCE. We hypothesize that this is a result of a more effective extraction of cause-effect links with delayed rewards because we use non-symmetrical Hebbian updates and rare correlated traces, effectively generating hypotheses about causal temporal links in the complex POMDP dynamics, while ignoring inherent noise in the feature space.

The ability to find important decision points and bridge state-action-reward gaps was achieved using three key mechanisms: (i) a novel implementation of STDP-inspired eligibility traces, (ii) a novel use of modulated Hebbian learning in a deep RL, and (iii) a new deep RL architecture that integrates DQN with a Hebbian-based structure. STDP-inspired eligibility traces bridge state-action-reward gaps, allowing MOHQA to solve problems where TD-learning fails. Modulated Hebbian learning allows large weight updates without gradient explosions. This means learning happens quickly just from few scoring examples, which is particularly useful when the agent finds a reward rarely. The new deep RL architecture is used to guide DQN to learn to follow a good policy so that it provides good features to distinguish states. Each of the mechanisms is vital in achieving good performance.

An interesting consideration is that MOHN also appears to be instrumental for guiding DQN to

learn useful features. Due to the confounding observations that are provided during wait states, the baseline algorithms struggle to learn useful features of the decision points, which are instrumental to inform an optimal policy. Thus, learning the appropriate features depends on the actions which in turn depends on the features. While this chicken and egg problem is typical in deep RL, MOHQA appears to facilitate the process of guiding the learning of useful features by discovering cause-effect relationships and offering guidance to DQN.

It is worth noting that memory does not help to solve problems in which the history of observations does not repeat (see performance of QRDQN+LSTM and AMRL). Firstly, recall that just like in real life, the exact history repeats itself very rarely in the CT-graph. Secondly, the CT-graph and Malmo have significant gaps between key states and rewards at all times. Coping with those two problems require a significant number of samples, which very quickly become computationally infeasible. Even in memory approaches designed to find key states such as FRMQN [21], key states are learned on small problem sizes.

MOHQA is a first proof-of-concept and was tested on a limited set of sparse reward problems and compared with a limited number of benchmark algorithms. Further tests with other sparse reward problems, e.g., the Morris water maze and some Atari games, will be essential to test the full potential of the approach. Yet, the present work suggests that a fundamentally simple confounding POMDP casts insights on the challenging problem of learning simultaneously a feature space and a policy.

Our new architecture, while proving effective and posing a new learning paradigm, has some limitations. MOHQA is more complex than a standard DQN, and requires tuning of additional hyper parameters. However, to the best of our knowledge, this is the first successful attempt to combine a MOdulated Hebbian Network with a DQN in a new RL architecture. An interesting future research direction is to implement MOHQA with different deep RL algorithms such as A2C. This should extend the capabilities and improve stability to allow solving even more complex confounding POMDPs.

MOHQA could also be used to help solve other problems such as autonomous driving and robotic delivery. In autonomous driving, roads have different lengths between junctions. Also, they are full of irrelevant features (cars, pedestrians, different building etc.) and the key decisions happen at the junctions. The reward is only given after reaching its destination. This is similar to the type of problems MOHQA has been able to solve. Other robotics tasks in stimulus-rich environments, such as tasks in houses or public places, are also full of confounding and changeable observations that are problematic for RL algorithms that have been tested only in video-game environments. Here we suggest and demonstrate that better algorithms are required to make RL more applicable to real-world problems.

1.4.2 Sliced Cramer Preservation

Numerical Experiments
Permuted MNIST
We first test our algorithm on the benchmark permuted MNIST task and compare the performance

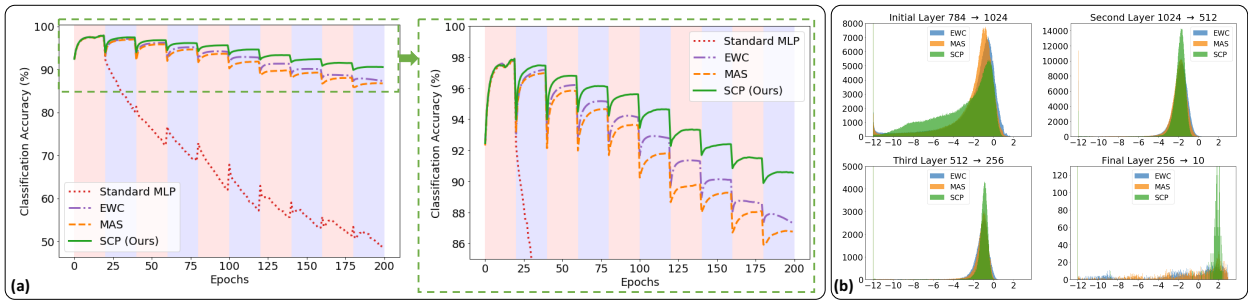


Figure 72: Comparison among various selective plasticity algorithms.

with online-EWC and our implementation of the online-MAS algorithm. For this experiment, we used a single head model that learns ten tasks, where each task contains a permuted version of the original MNIST dataset. We note that the reported results are a function of the architecture of the underlying network. Meaning that, while all three methods perform well on this task for larger networks, the true competitiveness of our method emerges for smaller networks where "over-estimation" of the synaptic importances significantly hinders learning of the subsequent tasks and leads to intransigence. For this experiment, we used a fully-connected network (i.e., a multi-layer perceptron) with the following architecture, $784 \rightarrow 1024 \rightarrow 512 \rightarrow 256 \rightarrow 10$ neurons, and for all optimizations we used the Adam optimizer with learning rate, $lr = 1e-4$. For Sliced Cramer Preservation (SCP), we used $L = 100$ slices. We repeated each experiment 10 times (with different permutations), and reported the average accuracy over all tasks in Figure 72 (a). We can see that SCP is capable of utilizing the capacity of the network better, which points to the fact that regularizing the distribution as opposed to the samples provides a less restricted regularization for the network and still enables the network to freely move individual samples so long as the overall latent distribution of the data is consistent.

The regularization coefficients for each algorithm was cross-validated on the following grid, $\lambda \in \{1e+i \mid i \in [-3,-2,...,9]\}$ and the optimal value was used to report the results in Figure 72 (a). Moreover, in Figure 72 (b) we show the histogram of the logarithm of the product of the regularization coefficients with importance parameters for each method (i.e., λ times the diagonal values of $F_{\boldsymbol{\theta}}$, Ω, and Γ for EWC, MAS, and SCP, respectively). One can see that the optimal values of the regularization coefficients provide, more or less, scale-consistent synaptic importances for all methods and distinguishing factor between the methods is on the difference between these distributions. Another interesting observation is that the distribution of the synaptic importances are more similar for sample-based methods (i.e., EWC and MAS) compared to our distribution-based method.

Sequential Learning of Auto-Encoders

Next, we consider an experiment consisting of unsupervised/self-supervised sequential learning. To that end, we learn an auto-encoder on single digits of the MNIST dataset sequentially. The model is chosen to be a fully connected auto-encoder, with the following encoder $728 \rightarrow 1024 \rightarrow 1024 \rightarrow 1024 \rightarrow 256$, a mirrored decoder $256 \rightarrow 1024 \rightarrow 1024 \rightarrow 1024 \rightarrow 784$, and ReLU activations. For the loss function, we used cross-entropy plus the ℓ_1-norm of the reconstruction error. Similar to the previous experiment, we used the Adam optimizer [105] for training the network with $lr = 1e-4$.

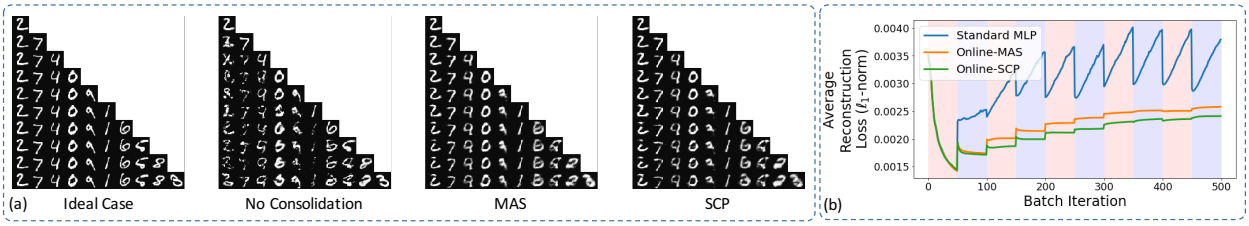


Figure 73: Comparison between MAS and SCP on sequential learning of auto-encoders.

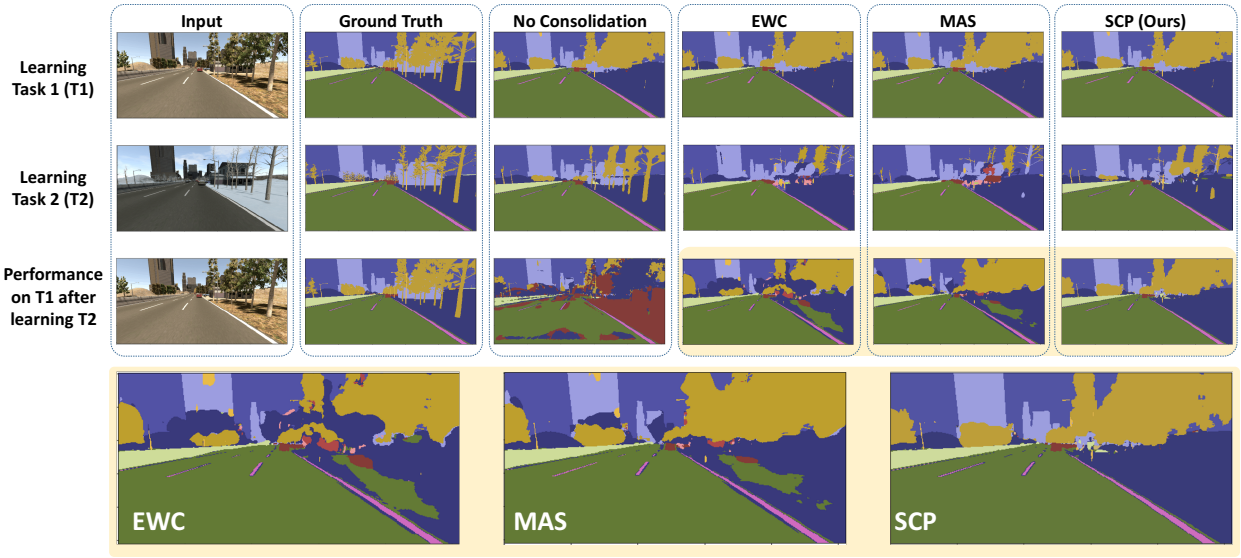


Figure 74: Comparison among various algorithms on semantic segmentation.

We perform 50 epochs of learning on each digit, before switching to the next digit. For consolidation, we used $L = 100$ slices for SCP. Finally, we permute the order of the digits and run our experiments 10 times, and compare 'No Consolidation,' with MAS, and SCP. Due to the unsupervised nature of the experiment, the EWC framework does not apply here.

Figure 73 demonstrate the results of this experiment. Panel (a) shows the reconstruction of a sample digit from one of our runs with the input digit sequence of $[2,7,4,0,9,1,6,5,8,3]$. It can be seen that both MAS and SCP can successfully retain the learned knowledge while acquiring new knowledge, while without consolidation the auto-encoder suffers from catastrophic forgetting. Panel (b) shows the average ℓ_1-norm of the reconstruction error for all seen tasks over the 10 runs. We can see that SCP and MAS are qualitatively on par, and SCP provides a modest yet statistically significant improvement over MAS for this task.

Semantic Segmentation of SYNTHIA Dataset

Lastly, we go beyond the benchmark yet less practical MNIST dataset and address catastrophic forgetting in a more interesting/critical application of autonomous vehicles. We specifically consider the problem of learning semantic segmentation of road scenes in a sequential manner, where the input distribution of the data changes over time. Semantic segmentation is the task of assigning a class label to every pixel of an input image. To that end, we use two sequences of the SYNTHIA dataset [183], namely 'SYNTHIA-SEQS-01-SUMMER' and 'SYNTHIA-SEQS-01-WINTER' as Task 1 and Task 2, respectively. There are 13 classes in the dataset namely: Miscellaneous, Sky, Building, Road, Sidewalk, Fence, Vegetation, Pole, Car, Sign, Pedestrian, Cyclist, and Lane Marking. We keep the last 100 frames of each sequence as the testing-set and train a deep convolutional U-Net architecture [184] on the tasks mentioned above. For the loss function, we used $(1 - Dice)$ [185], and each task was learned over 100 epochs. For the optimizer, we used the Adam optimizer [105] with learning rate, $lr = 1e-4$. For consolidation, we used $L = 100$ slices for SCP.

We learn the tasks sequentially (summer first and then winter), and a qualitative comparison of online-EWC, MAS, and SCP on a sample test frame is in Figure 74. The first row shows the performance on T1 after learning T1 (base model), the second row shows performance on T2 after learning on T1 and then T2 (indicator of intransigence), and the third row shows performance on T1 after learning on T1 and then T2 (indicator of the catastrophic forgetting). We can see that catastrophic forgetting happens when no synaptic consolidation is leveraged. Moreover, compared to online-EWC and MAS, SCP suffers less from intransigence as fewer artifacts are present in the second row. Lastly, SCP overcomes catastrophic-forgetting more successfully compared to EWC and MAS as it is apparent from the lack of artifacts in the last row. Finally, we provide a quantitative comparison between the methods in Figure 75, where we report the Dice score [185], averaged over ten runs, for each task and each method during the sequential training. As can be seen, SCP significantly outperforms online-EWC and MAS on this task both in overcoming catastrophic forgetting (left plot) and overcoming intransigence (right plot).

MNIST-to-SVHN Experiment

Here we include our results on the MNIST to Street View House Numbers (SVHN) experiment. In this experiment, Task 1 is learning the MNIST digits, and Task 2 is learning the SVHN digits. We show sample images from these two datasets in Figure 76a. And the model used is shown in Figure

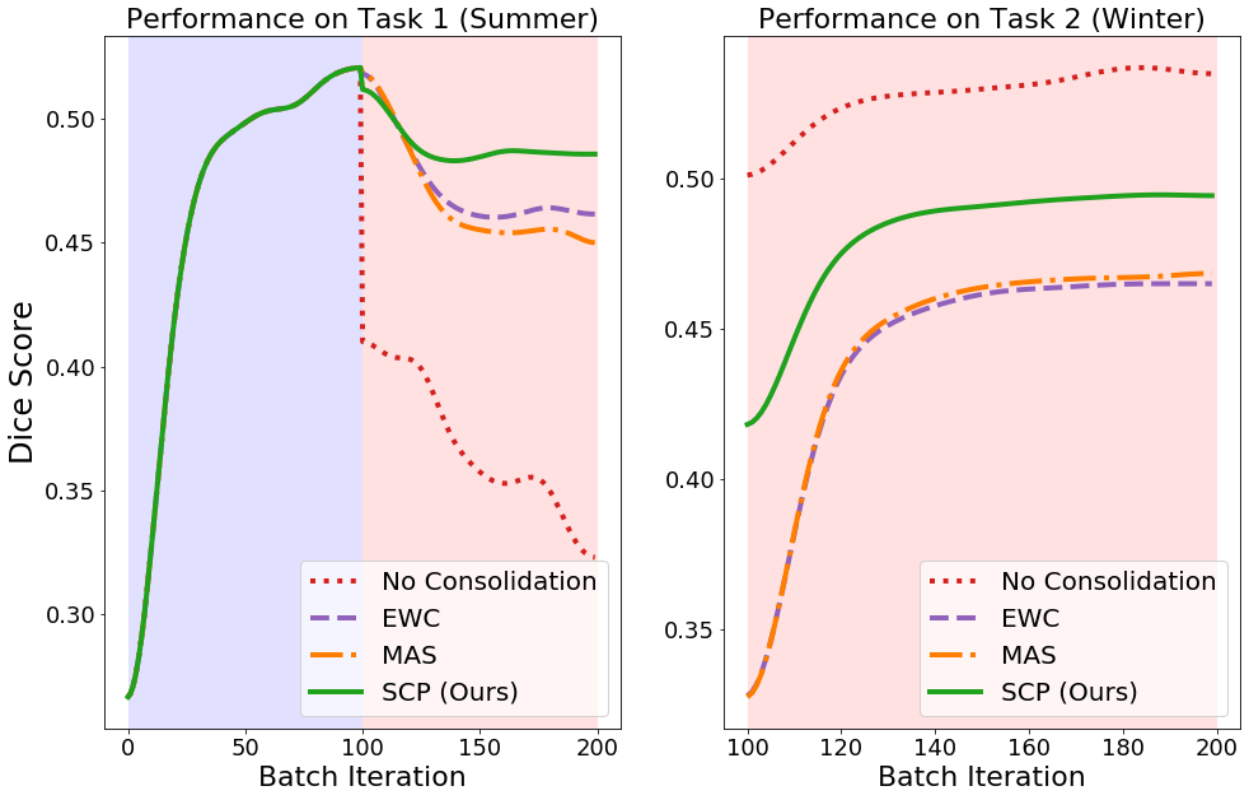


Figure 75: Testing Dice score of various selective plasticity algorithms.

77.

We performed incremental learning with no consolidation, Online-EWC, MAS, and SCP. The importance parameters were cross-validated on a coarse grid (due to the computational and time restrictions) of $\{1e+i|i \in [2,3,4,5,6]\}$, where the optimal parameters for the methods where $1e+3$ for SCP, $1e+5$ for MAS, and $1e+4$ for Online-EWC. We chose 20 epochs per task. Figure 76b shows the results of the experiment. Note that the first plot shows the testing performance of the methods on Task1 (i.e., MNIST dataset), where the blue shade indicates the first 20 epochs (training on the MNIST dataset), and the red shade indicates the second 20 epochs (training on the SVHN dataset). As can be seen, all methods can address catastrophic forgetting, while Online-EWC and SCP outperform MAS (However, a one should perform a finer grid search on the importance parameters for a definitive evaluation). The middle plot shows the testing performance of the model on the SVHN dataset (Task 2). The MAS and SCP methods outperform online-EWC, and SCP performs slightly better than MAS (i.e., overcomes intransigence better). The third plot shows the average testing performance of the methods on both tasks. Similarly, we can see that SCP outperforms MAS and Online-EWC. We repeated the experiments ten times, and the plots show the average performance.

CIFAR-100 Experiments

The CIFAR-100 [186] dataset contains images from 20 super-classes, where each super-class contains five s ub-classes. For this dataset, we considered an experiment in which the tasks are

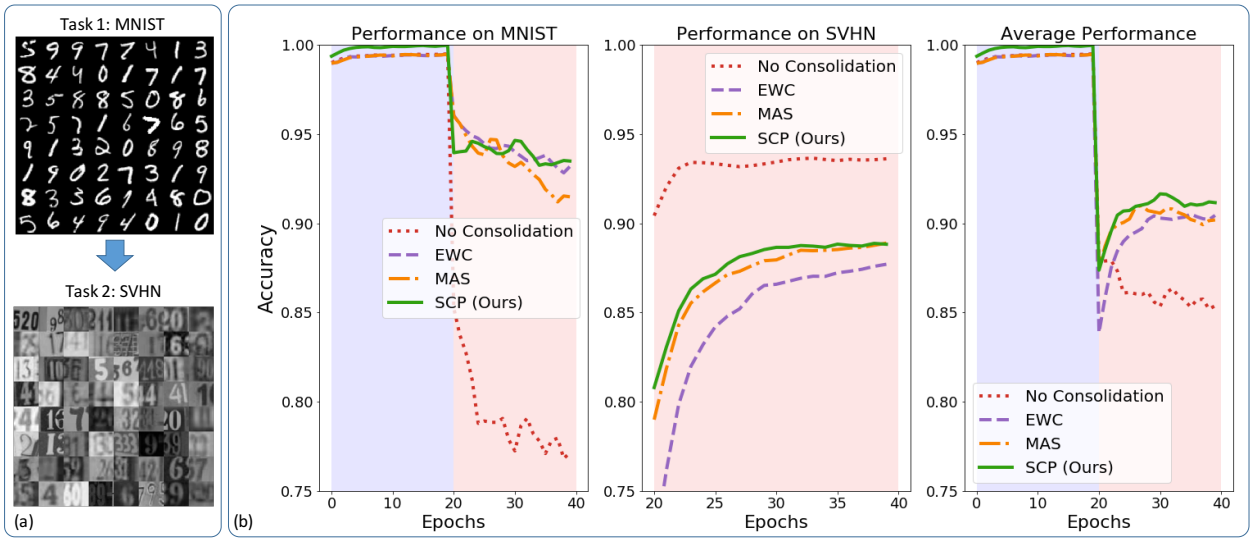


Figure 76: Comparison among various selective plasticity algorithms in the MNIST-to-SVHN experiment.

```
VGG_like(
  (activation): ReLU(inplace)
  (features): Sequential(
    (0): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): ReLU(inplace)
    (10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
    (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): ReLU(inplace)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
    (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (16): ReLU(inplace)
    (17): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
    (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (19): ReLU(inplace)
    (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc): Sequential(
    (0): Linear(in_features=12544, out_features=2048, bias=True)
    (1): BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace)
    (3): Dropout(p=0.2)
    (4): Linear(in_features=2048, out_features=1024, bias=True)
    (5): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU(inplace)
    (7): Dropout(p=0.2)
  )
  (classify): Sequential(
    (0): Linear(in_features=1024, out_features=10, bias=True)
  )
)
```

Figure 77: The model used in the MNIST-to-SVHN experiment.

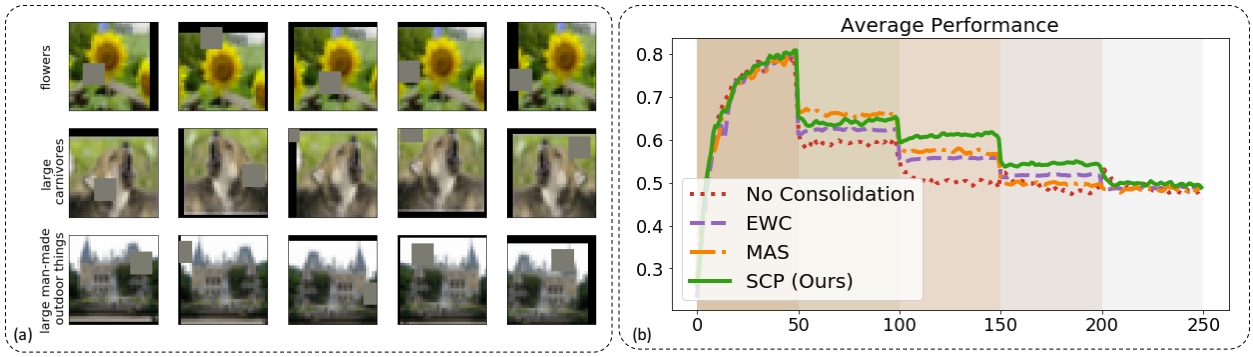


Figure 78: Comparison of various selective plasticity algorithms in the CIFAR-100 experiments.

Table 7: Prediction for 10,000 test pairs of noisy MNIST digits.

Goal Task	% Correct Digit Prediction	% Correct Goal Prediction
Even	92.03	99.50
Odd	91.15	99.75
Low	95.39	99.54
High	87.46	98.22

supervised classification of the super-classes. In short, we split the data into five sequential tasks, where each task contains a sub-class from all the 20 super-classes. We used a Wide Residual Network (Wide-ResNet) [187] network as our model and utilized the data augmentation suggested by [188]. Figure 78a shows sample super-classes together with data augmentation. For each method, we used a coarse grid search for the importance parameters $\{1e+i \mid i \in [2,3,4,5,6]\}$. Unfortunately, and due to the lack of time, we were able to only run the experiment for each method once. We report the average classification results in Figure 78b.

1.4.3 Neuromodulated Attention

Digit Prediction with c-EB and Noisy MNIST Pairs

The training process was carried out for 4,400 steps, including 256 noisy MNIST pairs modified from the original MNIST training set per step. The prediction performance of the fully trained model was tested on 10,000 pairs of noisy MNIST digits modified from the original MNIST test set [147]. Table 7 shows the digit and goal prediction results with c-EB driven by one of the four goal tasks (i.e., even, odd, low, or high value).

The goal was predicted along with the digit in the output layers for each forward pass. As shown in Table 7, the model predicted the goal correctly over 99% of the time, meaning that after the backward and forward pass the most active neuron predicting the goal matched the true goal. The model predicted the goal digit correctly over 90% of the time, meaning that the most active digit neuron after the backward and forward pass matched the expected digit based on the goal (Table 7).

This indicates that the goal tasks were successfully understood by the c-EB process to highlight related pixels. Although the statistics of the high-value goal task was slightly weaker than that of the other three goal tasks, the performance was still robust overall.

Goal-Driven Perception with Uncertainties

The robust digit and goal prediction results using c-EB assured that the network architecture could be applied to situations where goals uncertain and contexts are unknown. Therefore, the next step was to test the reliability and flexibility of our neuromodulation model for predicting goals in a noisy, dynamic environment.

Figure 79 shows typical runs of our neuromodulated system for three major validity settings; namely, 0.99 (a), 0.85 (b), and 0.70 (c). The major goal identity was randomly picked every 400 ± 30 trials for 10 switches in a run. The minor goal was the other goal in the same class of the major goal. For example, if the major goal "odd" had validity of 0.70, the minor goal "even" had validity of 0.30 until the next major goal switch. Within each sub-figure, the first subplot includes the true goals (labeled as "major goal" and "minor goal") and ACh-guessed goals (labeled as "guess"), and the second and third subplots show NE and ACh levels, respectively. A softmax function (Equation 33, with $\beta = 0.7$) was applied to ACh levels for goal guessing. Note that the ACh neuron corresponding to a major goal quickly increased driving attention to the most likely goal, as well as suppressing attention to distractors. In cases where the major goal validity was low, the ACh neuron corresponding to the minor goal was also activated, resulting in more exploration and a higher chance of guessing the minor goal. Interestingly, the prediction during exploration tended to remain in the same goal class. When there was a change in the goal identity, the NE neuron quickly recognized the change and responded with spike of activity. This caused the activities in the goal prediction network to reset, and a short period of exploration before the system found the new goal identity. Lower major goal validity led to longer exploration, especially after a goal identity switch, as well as more frequent NE bursts.

We also ran experiments where the goal validity could change during the run. Figure 80 shows the performance of a typical run with random switching among three major goal validity options, 0.99, 0.85, and 0.70. Similar to Figure 79, the system in this setting still focused more on the major goal. With lower major goal validity (i.e., when the major goal appeared less frequently, see 0.70 in Figure 80), the NE neuron fired phasically more frequently; meanwhile, the activity level of the major goal's ACh neuron oscillated more frequently with larger amplitude, giving higher potential for the minor goal's ACh neuron to fire at a low level. Because both the goal validity and goal identity changed during a run, the exploration period lasted longer with a lower major goal validity of 0.85 or 0.70. However, this also led to higher prediction accuracy of the minor goal.

Table 8 shows the performance of goal and digit prediction on the noisy MNIST pairs over 10 runs for each validity setting. The third and fourth columns refer to the percent of trials at which the goal digit prediction was correct. The fifth column refers to the percentage of incorrect goal guessing based on the ACh softmax distribution (Equation 33 with $\beta = 0.7$). The sixth column refers to the percent of incorrect digit predictions with c-EB driven by the guessed goal, when the ACh-guessed goal already matched the true goal. The seventh column refers to lag length of choosing the correct goals, which was computed as the number of trials between the first trial of a major goal switch and

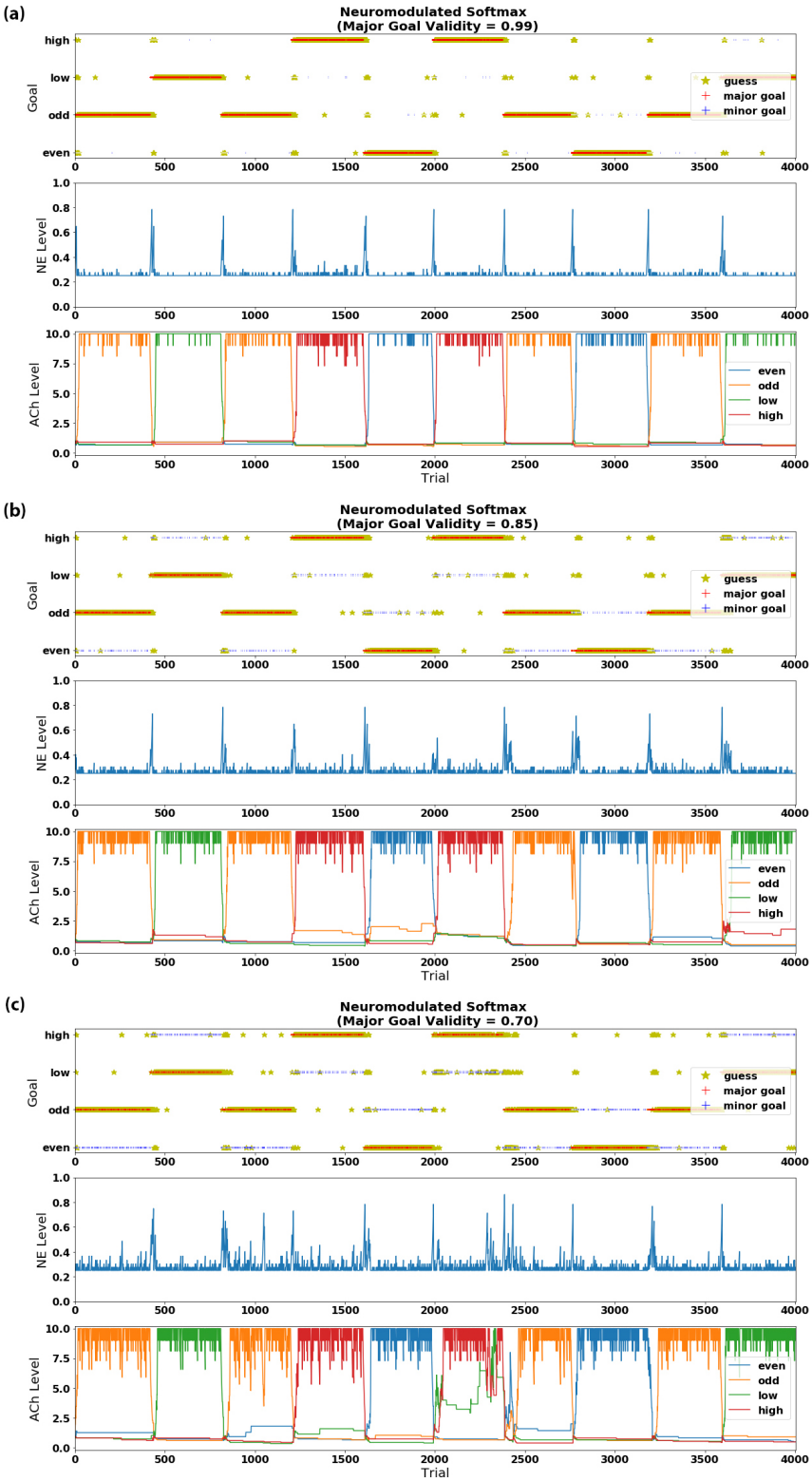


Figure 79: Visualization of goal-driven perception for different levels of the major goal validity.

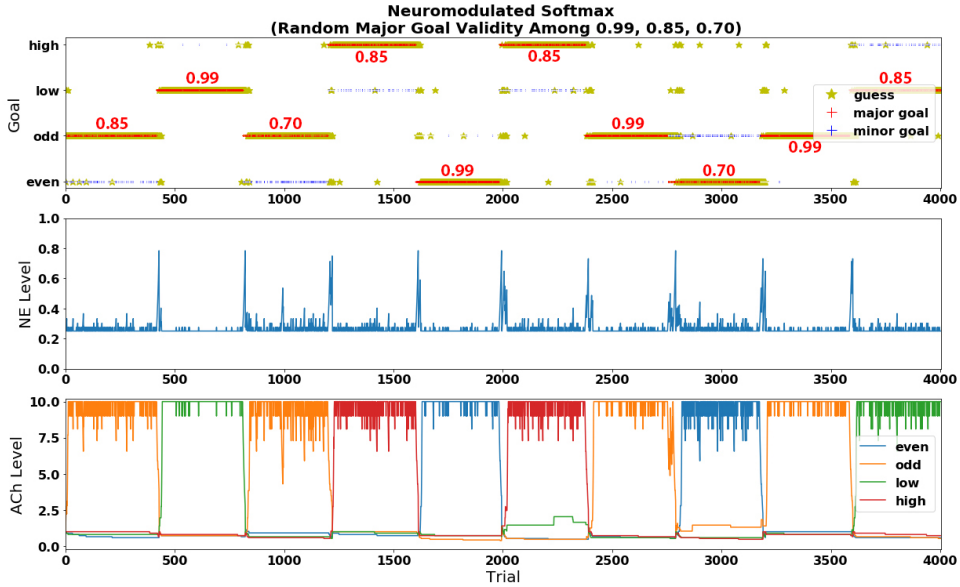


Figure 80: Visualization of goal-driven perception with the randomly switching major goal validity.

Table 8: Average goal-driven perception performance on noisy MNIST pairs for various goal validity settings.

Major Goal Validity	Minor Goal Validity	% Correct Major Goal	% Correct Minor Goal	% Incorrect ACh Softmax Goal Guessing	% Incorrect c-EB Digit Prediction	Lag Length (trials)
0.99	0.01	86.1	0.0	7.8	6.1	21
0.85	0.15	73.0	0.3	20.4	6.3	29
0.70	0.30	57.9	1.5	34.3	6.3	48
p_valid	1-p_valid	75.1	0.7	18.0	6.2	30

Table 9: Average goal-driven perception performance on noisy MNIST pairs for various ablations.

Ablated Neuron(s)	% Correct Major Goal	% Correct Minor Goal	% Incorrect ACh Softmax Goal Guessing	% Incorrect c-EB Digit Prediction	Lag Length (trials)
None	75.1	0.7	18.0	6.2	30
NE	70.8	1.1	21.6	6.5	54
ACh	19.8	2.9	70.9	6.4	400
NE & ACh	19.9	2.9	70.9	6.3	400

when the network started consistently making correct goal prediction 80% of the time over the last 10 trials.) The first three rows provide average statistics for runs at which a single goal validity was tested (see also Figure 79) and the last row corresponds with runs at which the goal validity could change randomly among three options during a run (Figure 80). p_valid means the major goal validity, and $(1 - p\_valid)$ means the minor goal validity. In each run, the major goal was randomly picked every 400±30 trials for 10 switches. The minor goal was selected from the same goal class.

Ablation Studies

We wanted to understand the effect of each neuromodulator on the network's ability to select goals. Therefore, we simulated ablation studies on ACh and/or NE neurons in a randomly changing goal validity experiment. These ablations had drastic effects on performance (Table 9 and Figure 81) compared to the complete network. In Table 9, the major goal was randomly picked among the four goal options every 400±30 trials for 10 switches in each run. For each major goal switch, the major goal validity was selected randomly among 0.99, 0.85, and 0.70. The minor goal was selected from the same goal class. The β value for the softmax function (Equation 33) was set to 0.7. With ablation of the NE neuron (Figure 81a), there was no scheme for network reset. The ACh neurons were still able to track the major goal switches. However as time elapsed, it took longer for the ACh activity level corresponding to the major goal to rise significantly and properly after goal switches, as measured by the lag length. With ablation of the ACh neurons (Figure 81b), the goal guessing became random. The firing rate of the NE neuron increased rapidly in the beginning and stayed at extremely high values afterwards. With ablation of both ACh and NE neurons (Figure 81c), there was no firing activity of either the NE or ACh neuron(s), and thus the goal guessing was random. These ablation studies demonstrate the necessity of having one system track the expected uncertainties (ACh) of goals and another respond appropriately when the goal distribution changes (NE).

Goal Selection Method Comparison

In the neuromodulated procedure of our model (Figure 12), the goal was selected by calculating the softmax distribution based on the activities of the four ACh neurons (Equation 33 with $\beta = 0.7$). The softmax function was important for raising the chance of choosing the minor goal when the major goal validity was low (e.g., 70%). We compared softmax to a WTA selection method, which still used the neuromodulatory head.

In addition, we compared the neuromodulatory head to another benchmark, which we call "random-

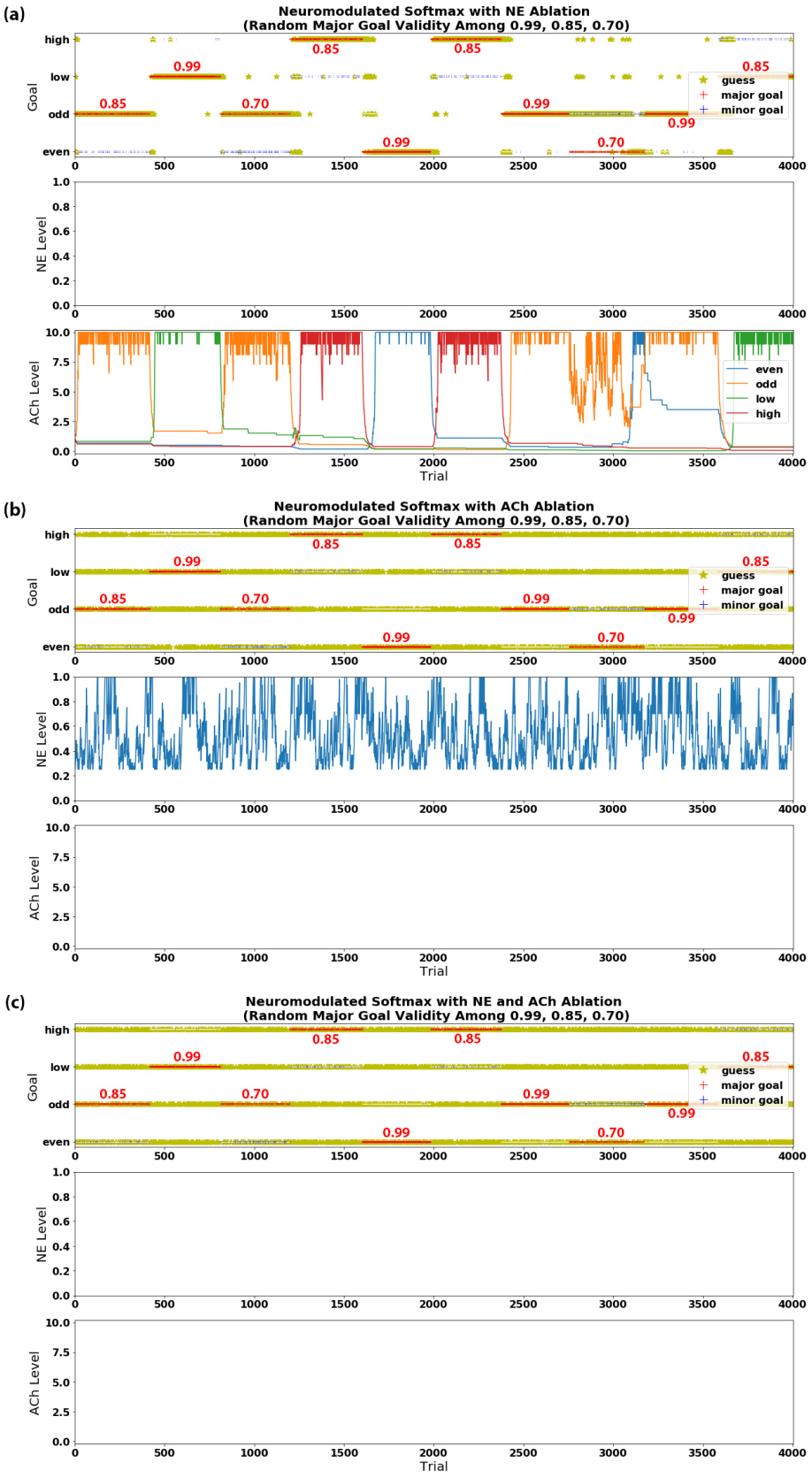


Figure 81: Visualization of goal-driven perception with the randomly switching major goal validity after NE and/or ACh ablation.

Table 10: Average goal-driven perception performance on noisy MNIST pairs among various benchmarks.

Goal-Guessing Method	% Correct Major Goal	% Correct Minor Goal	% Incorrect Goal Guessing	% Incorrect c-EB Digit Prediction	Lag Length (trials)
Neuromodulated Softmax	75.1	0.7	18.0	6.2	30
Neuromodulated WTA	76.4	0.8	16.5	6.3	24
"Random-Or-Fixed"	63.1	1.7	29.0	6.2	23

or-fixed". In the "random-or-fixed" benchmark, a predicted goal was randomly selected until it matched the true goal. Then the goal selection was fixed until a mismatch a ppeared. In other words, whether the guessed goal was random or stayed the same depended on whether there was a mismatch or match in the goal guessing process of the previous trial. Table 10 shows the performance comparison among neuromodulated softmax (shown in Figure 80), neuromodulated WTA (shown in Figure 82), and "random-or-fixed" (shown in Figure 83) on the noisy MNIST pairs. In each run, the major goal was randomly picked among the four goal options every 400 ± 30 trials for 10 switches. For each major goal switch, the major goal validity was selected randomly among 0.99, 0.85, and 0.70. The minor goal was selected from the same goal class. All three methods had similar lag lengths. Although the "random-or-fixed" method generated the highest percentage of minor goal matches, it was mostly caused by random guesses among all four goals, which also lowered the percentage of major goal matches. Therefore, the neuromodulation process was important for quickly following the desired goal class without hesitating over all four goals after each major goal switch. For neuromodulated softmax and neuromodulated WTA, their overall accuracy of major and minor goal guessing was quite similar. However, comparing Figure 80 for softmax with Figure 82 for WTA, we observed that the softmax function allowed higher chances of selecting the minor goal during the intervals at which the major goal validity was low, whereas the WTA function would like to select the major goal regardless of its true validity and could cause a much longer lag when the major goal validity dropped.

Goal-Driven Perception on Robot

To demonstrate that our model could generalize to a more practical application than MNIST digits, we tested our model on a HSR that had to guess an action with the neuromodulatory head, selectively attend to an object that corresponds to that action, and retrieve the object.

Figure 84 shows the performance of a typical run of the action-based goal-driven perception. Its neuromodulation process was similar to the algorithm described above for the noisy MNIST-pair experiment. Changes included using input images from three view angles of the HSR's camera, a fixed goal (action) validity of 1, and several parameter value adjustments (i.e., $\beta = 10$, $trial\_interval = 50$, $trial\_range = 0$, $ne_{correct} = 0.75$, $ne_{wrong} = 1.15$, $ch_{correct} = 1.35$, $ch_{wrong} = 0.95$). The average goal selection error for five runs was 23.8%, which was higher than the noisy MNIST-pair experiment (Table 8) because of shortened trial interval for each goal (action) switch. The average c-EB object prediction incorrectness was 30.6%. The average lag length was 13 trials. Uncertainties in each trial were addressed by possible object location switch, possible object removal and/or introduction, possible multi-instances of the same object(s), and slight view angle adjustment,

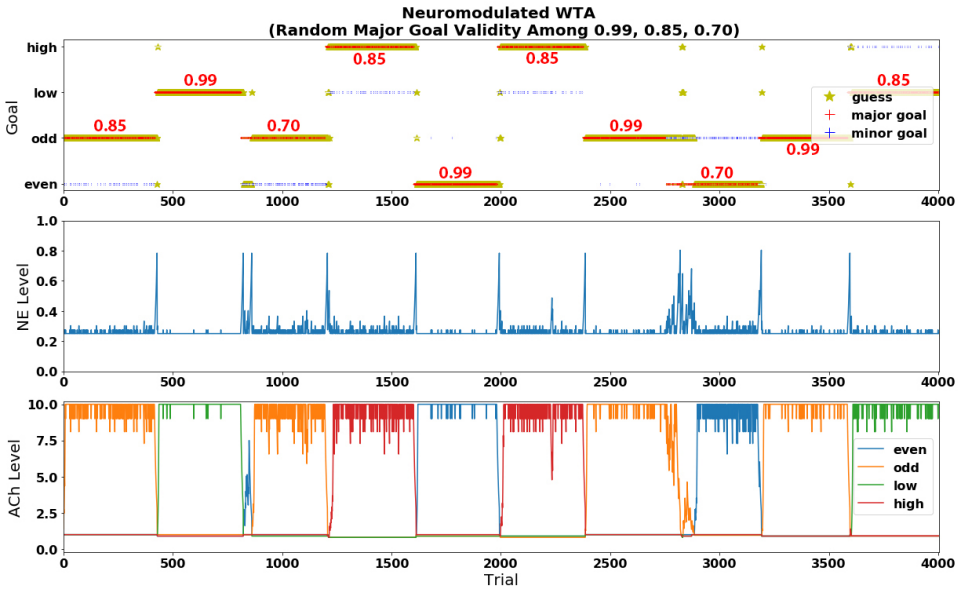


Figure 82: Visualization of goal-driven perception with the randomly switching major goal validity for neuromodulated WTA.

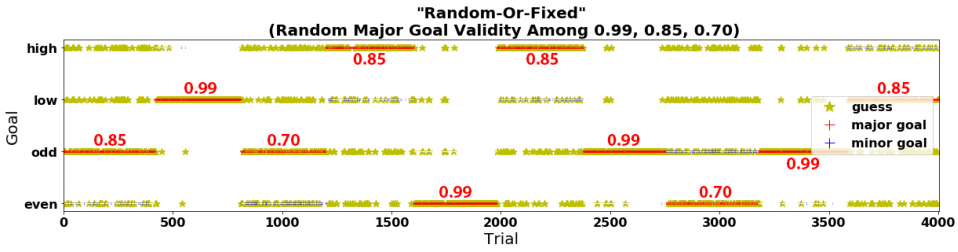


Figure 83: Visualization of goal-driven perception with the randomly switching major goal validity for the "random-or-fixed" benchmark.

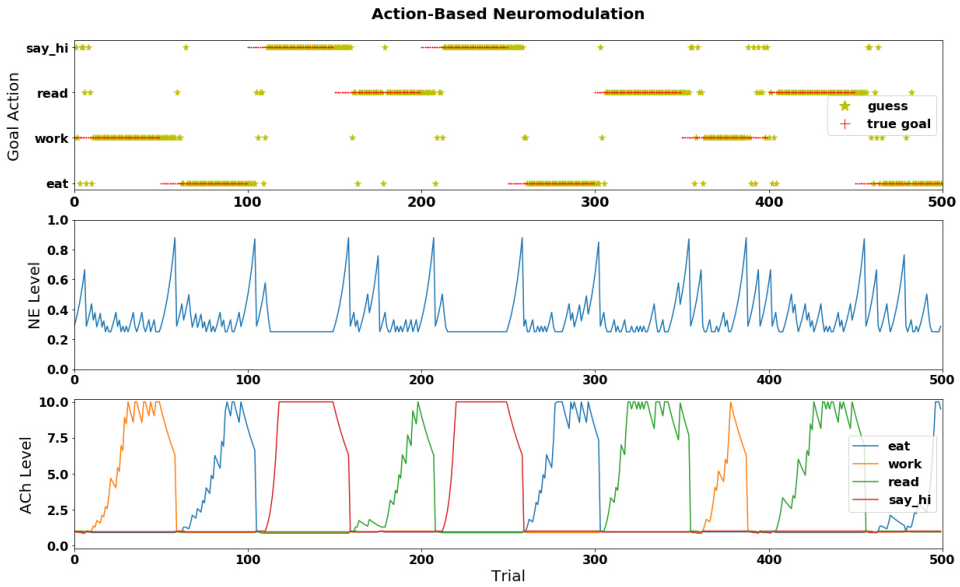


Figure 84: Visualization of action-based goal-driven perception in the robot experiment.

in addition to possible true action switch (i.e., every 50 trials, not given to the agent). A complete trial with HSR in the testing room can be seen in the YouTube video (https://youtu.be/DUy-0fDZEvY).

Main Findings

Here we showed that a neuromodulated goal-driven perception model, which combines ideas from neuroscience with goal-driven perception in artificial neural networks, could track context and flexibly shift attention to intended goals. Among many top-down attentional systems, we adapted c-EB [57] as part of our model because of its similarities to how the ACh neuromodulatory system both increments attention to a goal and decrements attention to a distractor [62, 61]. Goals are often unknown and need to be discovered. The c-EB algorithm was modified to support multiple goals. After training, the biologically inspired algorithm could rapidly learn the context without supervision, flexibly apply attention to the appropriate goal, and rapidly detect and re-adapt to context changes.

Neural Implementation of Uncertainty Tracking

[63] proposed a Bayesian model of neuromodulation in which the ACh system tracked expected uncertainty and the NE system tracked unexpected uncertainty. The present study advances this work in two ways to support goal-driven perception: (1) The Bayesian model was recast as a neural model to make it compatible with neural networks. The neuromodulators were implemented as a neural network layer to drive attention toward a goal digit and divert attention away from distractors. (2) A neural network reset was implemented to rapidly re-adapt when a goal changes.

Neuroanatomical studies show the basal forebrain, which contains ACh neurons, has topographical connections specific to stimulus modalities and values [189]. Therefore, different ACh neurons tracked the expected uncertainties of different potential goals. In a dynamic situation, the goal identity can change unexpectedly. Empirical evidence suggests that the NE system detects such

changes and generates a "reset" signal to discard prior expectations when these expectations are violated [64, 65]. In our experiments, the NE system rapidly recognized a change in the goal contingency, and drove a reset of ACh and NE activities. This caused the neural network to quickly explore new goals. It should be noted that the "reset" does not erase the learned object categories (e.g., digit parity and magnitude). Instead, it clears the prior likelihood of potential goals, and results in a rapid re-adaptation to the new goal distribution.

In the real world, goals are often uncertain and unknown. In our noisy MNIST-pair experiment, the goal validity (i.e., probability of a goal being rewarded) ranged from 0.99 to 0.85 to 0.70, and the system had to respond by either choosing the most likely goal or exploring alternative goals. Furthermore, the experimental design had a hierarchy of goals. For example, the goal would be to attend to the parity goal class and the sub-goal might be to reward odd digits 70% of the time and even digits 30% of the time. Interestingly, the neural network would often stay within a goal class (i.e., to choose parity and not magnitude).

In the robot action-based attention experiment, the objects linked with a predicted goal action might or might not exist in the views and may probably be at different locations. The adapted c-EB attention mechanism could pay significantly higher attention to existing objects. Selecting the highest attention region helped further with object localization and prediction. In both the MNIST-pair and real-scenario experiments, the unexpected major goal (action) switch after some trials could be quickly caught by the network within an acceptable lag.

Exploration and Uncertainty Seeking

Exploring options, rather than always choosing the most likely goal, is known as probability matching behavior [190]. Similar to the results presented here, humans tend to under select the most rewarding goal [191]. Such behavior may be due to feature exploration, as subjects test hypotheses by switching between the features before deciding upon their most rewarding goal. In rodent studies, it has been shown that rats will seek uncertainty, and that this uncertainty seeking is governed by the ACh system [192]. These uncertainty seeking strategies that appear in natural systems may be advantageous for artificial systems that are deployed in dynamic environments.

Related Work

Top-down task-driven attention is an important mechanism for efficient visual search in humans and artificial systems [193]. Many computational models of attention have been proposed and implemented to either explain top-down attention or develop an application inspired by these mechanisms [59, 194, 195]. Of particular interest are attentional systems that can leverage the power of CNNs. In these cases attentional information can propagate backwards, highlighting the features of a given goal [57, 196]. Similar to the effect of the ACh system to increment attention to a goal and decrement attention to distractors [61], [57] proposed an Excitation Backpropagation (EB) mechanism with a contrastive top-down signal to enhance the perception of goal features. Similarly, [196] proposed a technique called Class Activation Mapping (CAM) for identifying regions in an attention map. [197] proposed Gradient-weighted Class Activation Mapping (Grad-CAM) to highlight regions of interest and generate visual explanations. Their model could be applied to any CNN with no re-training. Similarly, our model can work with any CNN. Moreover, our model replaces the WTA mechanism or rigid probabilistic methods, with a flexible and adaptable layer

based on neurobiological neuromodulation.

Intrinsic rewards and curiosity seeking have similarities to the exploration due to uncertainty demonstrated by our model. These intrinsic reward systems typically are rewarded for exploring infrequently observed states [198, 199, 200]. Whereas the model introduced here selects goals based on the expected uncertainty of stimuli. In future work, it may be of interest to combine intrinsic rewards with uncertainty seeking.

Uniqueness of our Model
Our model is unique compared to existing attention models, which only focus on highlighting predefined (and pre-trained) goal objects in test images [201, 202], without any ability to deal with unpredictable switching and validity of goals. c-EB, which we adapted in our work, has been shown to achieve top-down attention competitively and robustly [57]. Moreover, it is similar to how the ACh neuromodulatory system both increments attention to a goal and decrements attention to a distractor [62, 61]. The neuromodulatory layer on top of a top-down attentional network demonstrates a means toward goal-driven perception where the system can autonomously learn which objects to attend to and which objects to filter out in the noisy, dynamic setting.

In addition to the unique aspects of our neuromodulatory model, its robustness was ascertained via comparisons with neuromodulated WTA and "random-or-fixed" benchmarks. We also used ablation studies to show the necessity of having both ACh and NE neuromodulators to track the expected and unexpected uncertainties of goals and respond appropriately when the goal distribution changes. Further, generalization was ascertained via the HSR implementation in a real indoor scenario.

1.4.4 Self-Preserving World Model

Performance in the *M* network (here the average loss per output unit) was assessed on the held out test-set of rollouts for each task and was done on all potential tasks at every epoch of training. The top-left of Figure 85 shows the percent of total integrated loss over both experimental conditions (with and without pseudo-rehearsal), averaged over 10 random replications of task exposures. Desaturated bars shows loss without pseudo-rehearsal. Error bars are standard error of the mean. And the top-right shows pair-wise difference in total percent loss for each task. Error bars are 95% confidence intervals. Finally, the bottom of Figure 85 shows these performance curves for each of the three different Atari games as tasks. Solid lines show performance when simulated rollouts were interleaved during training, and dashed lines show performance when no interleaving of simulated rollouts occurred (with the label suffix of '_nosim'). The different lines colors in each curve correspond to when the network was being training (as dictated in the legend) on a particular task. Overlaid boxes indicate when a given task is engaged in training on its own data. Clear catastrophic forgetting is seen in the non-interleaved case while relatively little reduction in loss is observed when simulated rollouts were interleaved. The average percent loss, and pair-wise percent loss difference plots show that each task significantly more preserved when using pseudo-rehearsal.

In Figure 86, reconstructions of test rollouts from the first task are shown across task exposures. Left grid shows simulated rollouts when no pseudo-rehearsal was used in training, right grid shows simulated rollouts when trained with pseudo-rehearsal, and far right column shows real rollouts

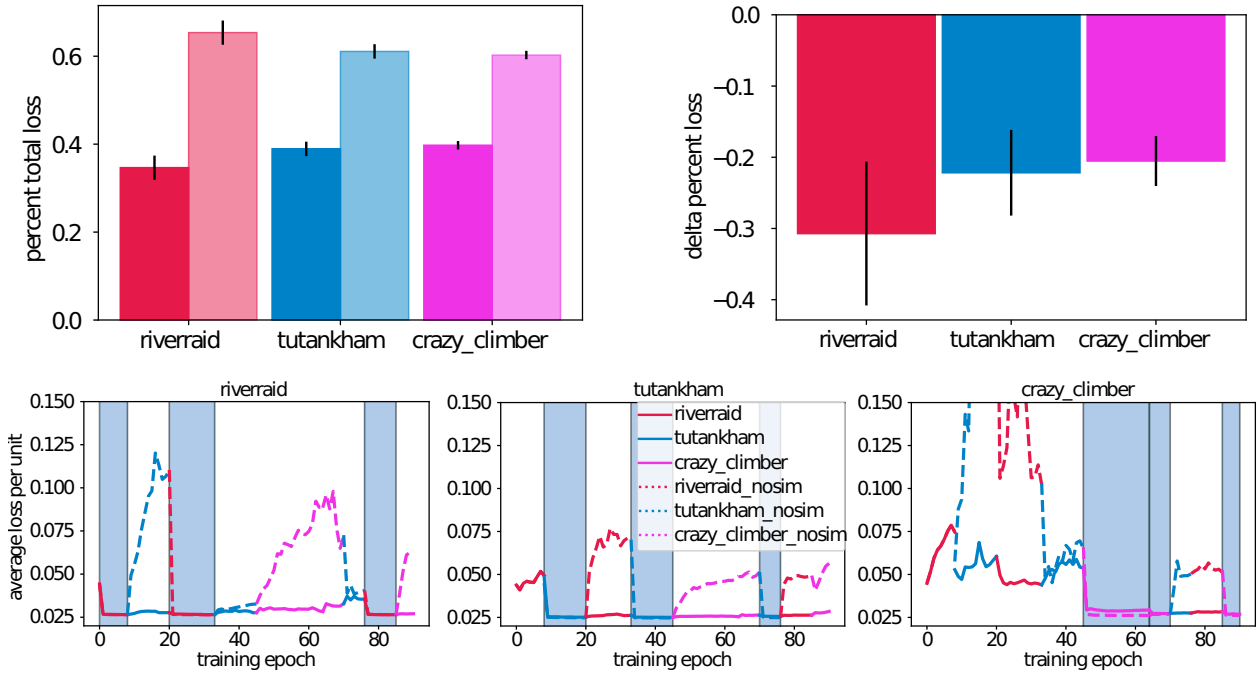


Figure 85: *M* network loss over task exposures.

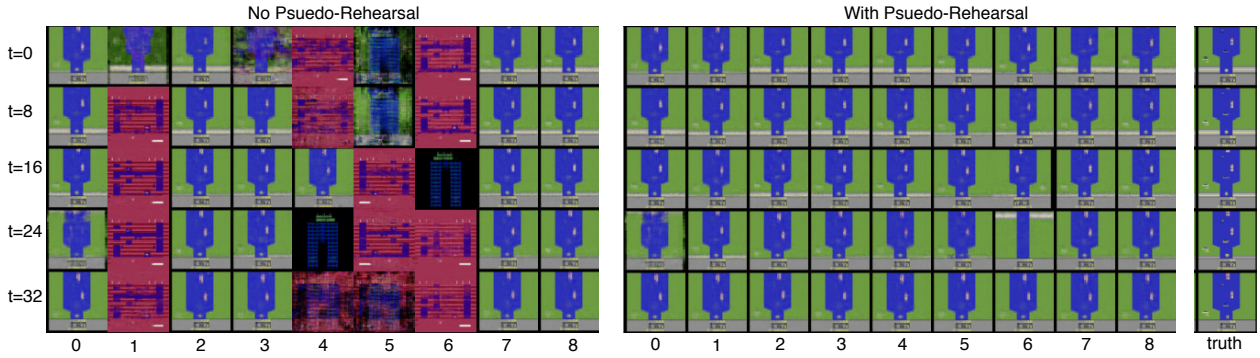


Figure 86: Reconstruction of test rollouts from River Raid Atari game using *M* networks after training in each task exposure.

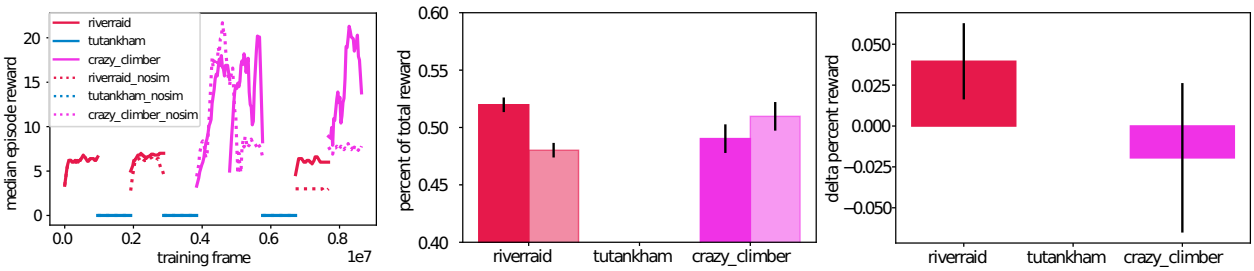


Figure 87: Cumulative reward in *C* network over task exposures.

from the environment. Grid rows correspond to a given rollout's time steps, columns are specific rollouts generated after training is complete in each task exposure. This figure provides a heuristic for translating the change in loss observed in Figure 85 into appreciable samples. This also gives the reader an appreciation for the types of errors made when using pseudo-rehearsal vs. not, for a comparable level of loss. Clear signs of catastrophic forgetting are seen in the reconstructed samples when simulated rollouts are not interleaved. Conversely, the impact of the relatively small increased loss when using pseudo-rehearsal is seen in Figure 86 to be relatively unintrusive when reconstructed into the original input space. This increase does illustrate the potential for accumulating loss when scaling this method to 100s of tasks. However, it can be seen in Figure 85) that each task showed significantly more accumulated loss when pseudo-rehearsal was not used.

We conducted further experiments using a larger set of tasks to investigate transfer and continual learning in the *M* network. These results suggest that this architecture can learn a disparate set of 10+ Atari games, however, some tasks start to show signs of reconstruction degradation if the duration between exposures is too long, or if their first exposure is not until late in training.

Controlling Agent

In Figure 87, the left plot shows median episode reward over 30,000 frames over task exposures for a particular replication, with solid lines corresponding to percent reward when using pseudo-rehearsal, and dashed lines without (labeled with '_nosim' suffix). The middle plot shows percent of total integrated reward averaged over 10 random replications, with desaturated bars corresponding to average percent reward when not using pseudo-rehearsal. Error bars are standard error of the mean for the pair-wise difference between percent reward with and without pseudo-rehearsal. In the right plot, error bars are 95% confidence intervals. As can be seen, continual learning in the controlling agent was observed in the Task 1 results were inconclusive for Task 2 or 3. Task 1 (River Raid) in general shows a larger percent of total reward over training epochs when using pseudo-rehearsal as compared to without. The plot of a particular replication illustrates that this is in large part due to a performance drop in later training for non-interleaved training. For Task 2 (Tutankham), in general no reward was observed for this level of exposure, and so performance differences were unmeasurable. Task 3, (Crazy Climber) however shows no significant difference between training conditions. This null effect for continual learning could be due to a lack of complexity in this environment such that one million frames is sufficient to retrain the agent at every exposure, or that variance in reward over replications is too high to show any discernible differences in experiment conditions.

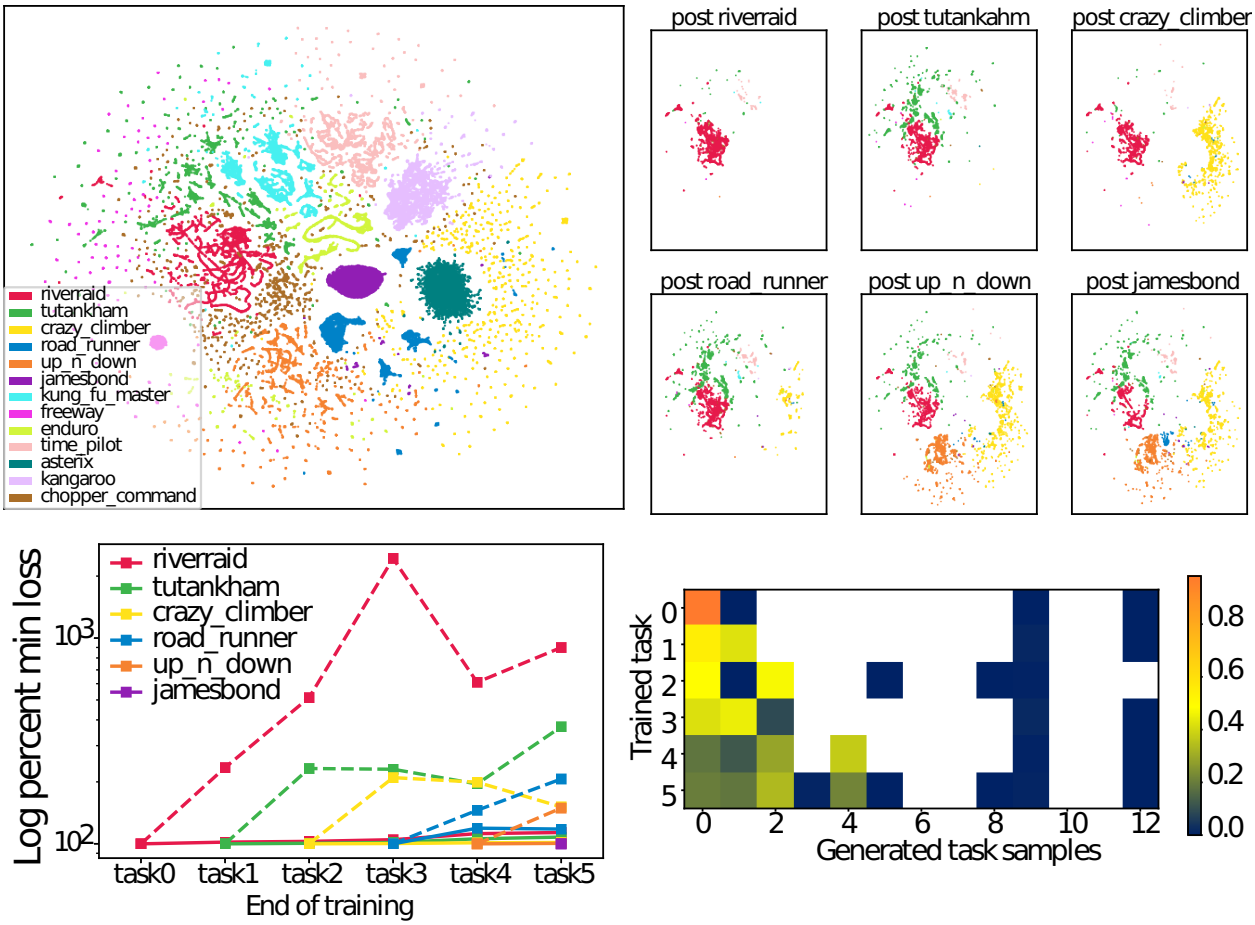


Figure 88: Results from Supplementary Experiment 1.

Two more experiments were done which focused on the M and V networks explicitly. The first was done to investigate if the simulated rollouts can be attributed to task-specific preservation. The second investigated the larger potential for between task interactions and the ostensible capacity of the current architecture by training on a larger set of potential tasks.

Supplementary Experiment 1

This experiment was done by first training the V network to encode and decode a set of 13 Atari games. This V network was then used to encode samples from six (River Raid, Tutankham, Crazy Climber, Road Runner, Upń Down, James Bond) of those original games, and sequentially trained the M network on those six games. During the course of training the M network saw each game only once and was trained for 30 epochs or until test error did not decrease for five consecutive epochs. No C network was trained during this experiment and all rollouts were done using the random policy with a 0.5 probability of repeating the last randomly sampled action.

The simulated rollouts for each iteration of sequential training are shown in Figure 88. The top-left plot shows the visualization of the learned V latent space in two dimensions using a UMAP embedding [203] derived from a random set of test rollouts totaling 10,000 samples from each

task. Using this embedding a K-Nearest Neighbor (KNN) classifier [204] is trained using the V embedding of the same data, and then used to estimate a task label for pseudo-samples, which themselves are a random set totaling 10,000 samples taken from the rollouts interleaved during the experiment (top-right plot). The bottom-left plot shows minimum loss after learning a given task. Line colors show Atari game corresponding to the x-axis and order of learned tasks; e.g., 'task0' corresponds to 'River Raid'. Solid lines show learning with pseudo-rehearsal and dashed line shows without. Performance is percentage change from minimum loss within a given task, and y-axis is shown in the log scale. And the bottom-right plot shows the proportion of generated samples within each task for each iteration in the sequential training; rows correspond to iterations in the sequential training, and columns correspond to the proportion of samples labeled by the KNN classifier. Each row sums to 1, and white squares correspond to cases which had a proportion of less than 0.004 of generated samples within a given iteration. As can be seen, the pseudo-samples generated stay mostly within and distribute relatively well across trained tasks, even with no explicit task labels provided. This sampling, however, clearly does deviate from uniform, and one consequence of obscuring task labels during training is that forcing a particular sampling over previous tasks becomes more difficult.

As above, performance was assessed on a held out test set of rollouts for each task. This assessment was done on all previous tasks after training was complete for a given task, and was normalized by the minimum loss achieved for each respective task, i.e., the loss after initial training within a given task. In this way a measure of the degradation of performance as a function of learned tasks was achieved. Bottom-left plot in Figure 88 shows these metrics over training where, solid lines show performance when simulated rollouts were interleaved during training, and dashed lines show performance when no interleaving of simulated rollouts occurred. Clear catastrophic forgetting is seen in the non-interleaved case while relatively little reduction in loss is observed when simulated rollouts were interleaved.

In this controlled experiment of task exposures we found a significant relationship between percentage of pseudo-samples generated within a given task and the increase in loss within a iteration of the sequential learning experiment (Spearman rank correlation $r = -0.58, p = 0.02$), suggesting that the more pseudo-samples used within a given task the smaller the increase in error. One potential improvement from the current approach would be finding methods to enforce a smarter pseudo-sampling, however what the optimal sampling should be is an active area of research.

Supplementary Experiment 2

Here the V network was again trained on the same 13 Atari games, and M training was done in the mostly the same fashion as done above; i.e., a random order of task exposures, however, here training consisted of only two exposures with a sum total of 30 epochs per task. Here we also used an early stopping criteria to try and limit over-training such that training was stopped in each task exposure if test error within a given task did not decrease over five e pochs. No C network was trained and rollouts were determined using the same random policy as done previously.

In Figure 89, the loss over training epochs for each task in random task exposure ordering is shown, similar to Figure 85. Each sub-plot shows loss as measured in the hold-out set of the task label above it, lines colors correspond to the task that is currently being trained at a given epoch and grey

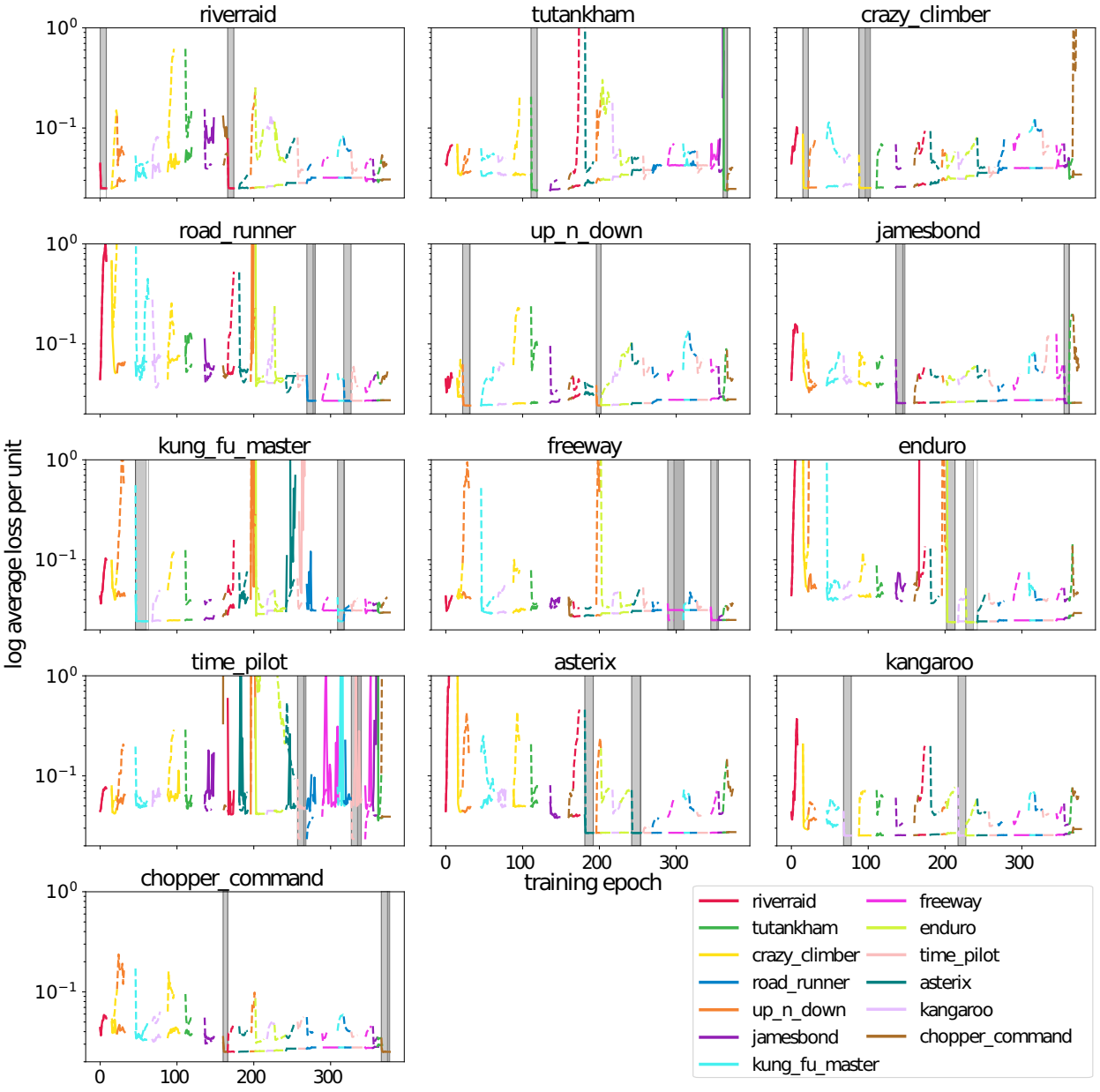


Figure 89: Results from Supplementary Experiment 2.

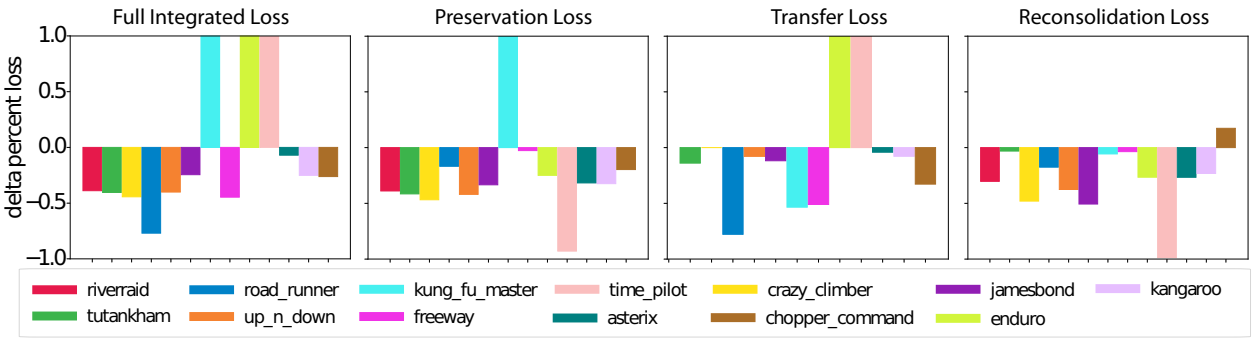


Figure 90: Percent of integrated loss plotted for each task.

overlays show when a that task is being exposed to its own training data. Solid lines show loss when using pseudo-rehearsal, dash lines without pseudo-rehearsal. Results illustrate the complex dynamics of the loss in the *M* model when training over many potential tasks. In general most tasks are learned as well as the three task experiment described above, however a larger accumulation of loss occurs for tasks that are not exposed until late in training. This can be seen in Figure 90 where the percent of integrated loss is plotted for each task. The left plot in Figure 90 illustrates the pair-wise difference of with pseudo-rehearsal minus without pseudo-rehearsal loss as integrated from the first epoch of training in the first task exposure. Additionally, the middle-left plot labeled Preservation Loss shows the same metric when loss is integrated when a given task is first exposed, the middle-right plot labeled Transfer Loss illustrates the loss integrated up to a given task's first exposure, and the far right plot labeled Reconsolidation Loss shows the integrated loss starting from the second exposure until the end of training. These metrics are similar to forward and backward transfer defined in [205]. Here we have a sign flip where Transfer Loss is analogous to the negative of forward transfer (i.e., the influence that a given task exposure has on the performance of previous task exposures, in our case negative is better), and Preservation Loss is analogous to Backward transfer (i.e., the influence that a given task exposure has on future task exposures, again negative is better).

These plots illustrate two main points. The first is that the full integrated loss can effectively be broken down into Preservation Loss and Transfer Loss. Here the potential for transfer between tasks can be seen in the middle-right plot of Figure 89. Some tasks' pseudo-rehearsal loss drop below the no-rehearsal loss before that task gets its first training exposure (e.g., negative values in the right most plot). Similarly, it can be seen that several tasks' losses interact with each other in a complex way that would be hard to quantify but suggests the temporal prediction of those tasks are related, perhaps through the auto-encoder *V*.

The second main result from this experiment is the illustration of continual learning in this large set of diverse tasks. This not only occurs in the first task exposure, but seems to take root more completely after the second exposure, as measured by the Reconsolidation Loss (naming based on the memory literature [206]). One interpretation of this fact is that the more tasks the *M* network is exposed to, the better it is at meta-learning. This is generally in line with results from other meta-learning approaches which effectively exposure a network to batches of many related tasks to 'initialize' the network's weights to be capable of quickly adapting to new incoming tasks [207, 82]. In contrast several tasks show Reconsolidation Loss comparable to when no pseudo-rehearsal is used. Without running these simulations for longer, it is hard to determine if this is a measurement artifact due to lack of epochs for loss to accumulate after the final task exposure (e.g., tasks that had their last exposure near the end of training) or if this measure is manifesting some capacity in the current architecture.

1.4.5 Context-Skill Model

The evolutionary learning results are first described, followed by evaluation of the generalization ability of the best solutions. For animated examples of behaviors in each domain, see https://drive.google.com/drive/folders/1GBdJzD9tDHJkd59YbQUOIQua6nCiLjXa.

Table 11: Parameter ranges for evaluating generalization of the Context-Skill Model.

FB (±75%)			LL (±50%)			CARLA (±35%)		
Flap_{min}	=	-21.0	Main_{min}	=	10.0	α_{min}	=	0.65
Flap_{max}	=	-3.0	Main_{max}	=	30.0	α_{max}	=	1.35
Fwd_{min}	=	1.25	Side_{min}	=	0.5	β_{min}	=	0.65
Fwd_{max}	=	8.75	Side_{max}	=	1.5	β_{max}	=	1.35
$\text{Gravity}_{\text{min}}$	=	0.25	Mass_{min}	=	4.0			
$\text{Gravity}_{\text{max}}$	=	1.75	Mass_{max}	=	12.0			
$\text{Drag}_{\text{base}}$	=	0.25						
$\text{Drag}_{\text{base}}$	=	1.75						

Learning

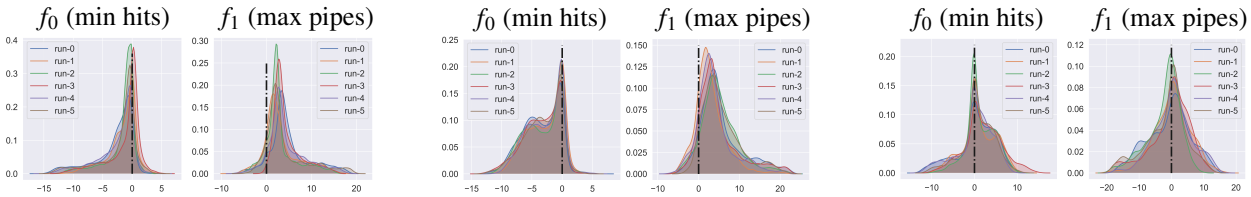
CS, C, and S architectures were evolved with different random seeds six times in FB, five times in LL, and once in CARLA (due to the computational complexity of this domain). To avoid overfitting, specific stopping criteria were selected for each domain after some experimentation. In FB, training was run until an individual in the population achieved a fitness scores of at least f_0=0.01 (hits) and f_1=22.0 (pipes). In LL, training was run until an individual reached f_0=200 (total rewards). In CARLA, evolution was run until an individual reached f_0=2043 (safety penalty) and f_1=55 meters (distance from the target). In FB and CARLA a minimum performance criteria was included to make sure the agent would not simply optimize safety by staying still. The final Pareto-optimal set in each run contained multiple individuals, of which one representative network was selected for the generalization evaluation. In FB and LL, it was a safe network (i.e., satisfying the stopping criteria) with the highest performance, and in CARLA, it was the network with the least Euclidean distance to the origin in the Pareto front.

The evolution of S in general took the shortest amount of generations since it has the least number of parameters (e.g., 287 compared with 982 in C and 1207 in CS in the FB domain). The same architecture is used for all three domains. To make sure the number of parameters was not a factor, another S with a larger Skill module, with the same number of parameters as CS, was also evolved with the same stopping criterion. However, it performed poorly compared to the smaller S in the generalization studies, apparently because it was easier to overfit. Thus, it was excluded from the comparisons.

Generalization

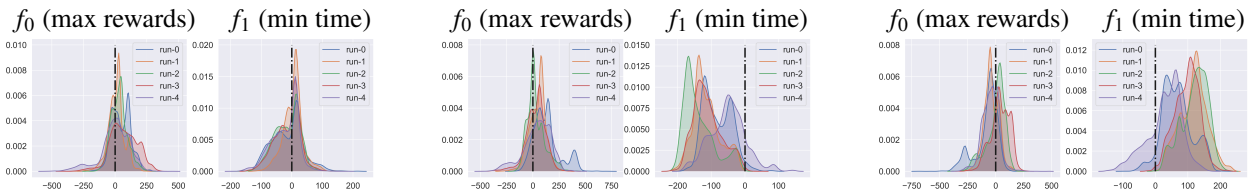
To evaluate the generalization performance of the best performing networks, the task parameters in each domain were changed in the following ways:

- The range of variation in the task parameters was increased 35-75% beyond the training limits (Table 11);
- All parameters were varied simultaneously instead of one at a time; and
- In the CARLA domain, the agents were evaluated on a new track that had never been encountered during training. This track included curves with significantly higher curvature than those encountered during training.



(a) CS - S (b) CS - C (c) C - S

Figure 91: Comparing generalization of CS and its ablations C and S in the FB domain.

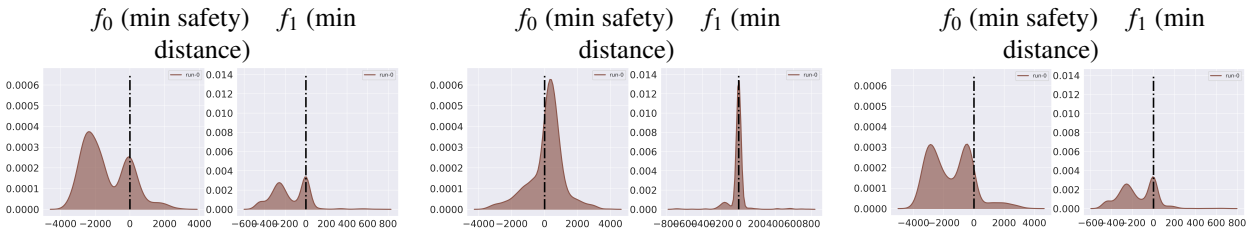


(a) CS - S (b) CS - C (c) C - S

Figure 92: Comparing generalization of CS and its ablations C and S in the LL domain.

More specifically, the range of each game parameter was discretized into 10, 20 and 35 values in FB, LL, and CARLA, respectively, and each network was tested with every possible combination of these values, i.e., with 1,225, 8,000, and 10,000 different settings. In FB and LL, each parameter combination was tested three times and the results averaged; in CARLA, only one sample was used. Figures 91, 92 and 93 show the difference in generalization performance across the 3×10^4 test episodes between CS and S, CS and C, and C and S. They are histograms indicating how often each difference was observed in the episodes. The first network in the subtraction performs better wrt. a minimization objective if the histogram is skewed to the left, and wrt. a maximization objective if the histogram is skewed to the right.

Thus, the plots show that in both FB and LL, CS generalizes much better than S and C both in terms of safety and performance. In CARLA, CS generalizes better than S and C in terms of performance; in terms of safety, CS generalizes much better than S in terms of lane distance and about the same in wobbliness, and it generalizes much better than C in terms of wobbliness and about the same or slightly less in terms of lane distance. Thus, CS consistently achieves superior performance and mostly superior safety across multiple domains.



(a) CS - S (b) CS - C (c) C - S

Figure 93: Comparing generalization of CS and its ablations C and S in the CARLA domain.

Behavior Analysis

To understand how the CS architecture outperforms its individual components C and S, an FB task with parameters [Flap=-7.0, Gravity=0.58, Fwd=8.75, Drag=0.58] was evaluated further. This setting has a previously unseen exaggerated effect for forward flap, and a previously unseen diminished effects for upward flap, gravity and drag. Thus, actions tend to push up and speed up the agents more than expected, and it is difficult for it to slow down and come down.

Neither the C nor the S network performed well in this task: The C network collided with six pipes and S with five. Remarkably, CS managed to pass all 21 pipes. Both C and S used all four actions (flap upward, forward, simultaneously upward and forward, and glide, i.e., do nothing), but CS interestingly never uses flap upward alone. That action simply lifts the agent up, which is rarely optimal action in this environment where it takes such a long time to come down. If it is necessary to go up it is because the opening is high, and in that case it is more efficient to move forward at the same time.

As an illustration, second row in Figure 94 shows a situation at the fourth and fifth pipe. Both C and S make a similar mistake by flapping up and forward. They end up too high too fast, do not have enough time to come back down, and crash into the fifth pipe. In contrast, even before the fifth pipe becomes visible, CS refrains from both actions while there is enough time for weaker gravity and drag to slow and pull down the agent, and it reaches the opening in the fifth pipe just fine.

To understand how CS manages to implement this behavior whereas C and S do not, the outputs of the Context and Skill modules are compared between this generalization task (where C and S hit the fifth pipe) and a task where all parameters are at their base values (where C and S do not hit the fifth pipe). Their 10 and 5-dimensional outputs are first reduced to two dimensions through principal component analysis and then subtracted. The top row of Figure 94 shows these differences for the Context and Skill modules of CS, for the Context module of C, and for the Skill module of S, at the locations in the image below.

One might intuitively expect that Context module in C and Skill module in S would not change their behavior much in the generalization task, but the Context module in CS would vary significantly to modulate the output of the Skill module. Surprisingly, the opposite is true: Both the Context and the Skill module in CS vary very little compared to those in C and S (see also the quantitative comparison in Table 12). The generalization task presents novel inputs that results in novel outputs in C and S, and the controller does not know how to map them to correct actions. In contrast, the Context and Skill modules in CS have learned to standardize their output despite the change in context; their outputs are what the controller expects as its input, and is able to output the correct actions. Interestingly, this effect is similar to standardizing context in sentence processing, which makes it possible to generalize to novel sentence structures [208]. Remarkably, whereas in sentence processing the standardization was implemented by a hand-designed architecture, in the Context-Skill approach it is automatically discovered by evolution.

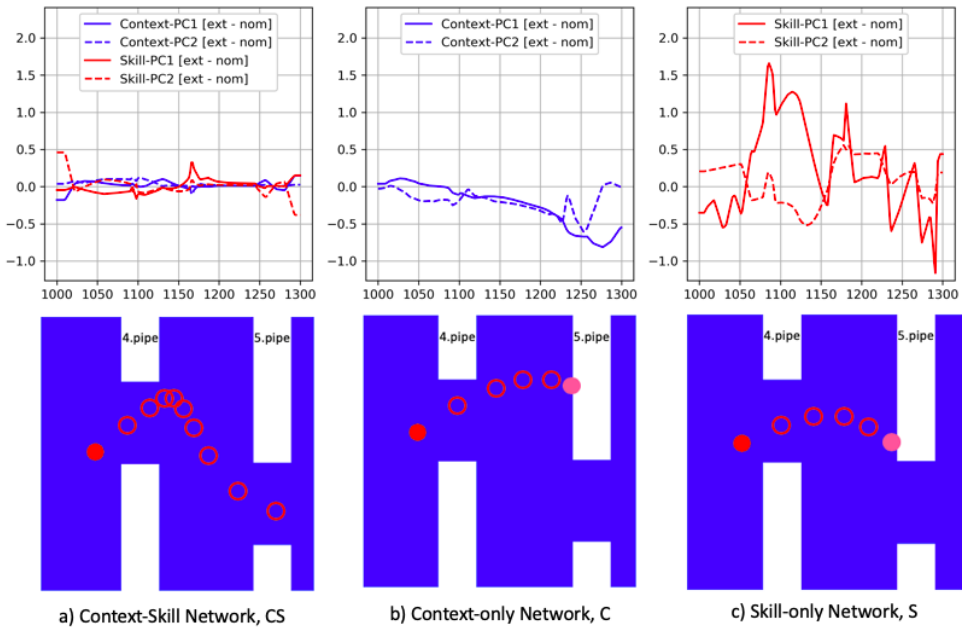


Figure 94: Contrasting the generalization ability of (a) CS, (b) C, and (c) S networks.

Table 12: Change in Context and Skill modules during generalization.

Network	PC	MSD	STD
CS	Context-PC1	0.004	±0.058
	Context-PC2	0.003	±0.039
	Skill-PC1	0.007	±0.082
	Skill-PC2	0.018	±0.133
C	Context-PC1	0.139	±0.282
	Context-PC2	0.061	±0.147
S	Skill-PC1	0.414	±0.600
	Skill-PC2	0.088	±0.286

1.4.6 Reflexive Adaptation

Simulation experiments were conducted to test the MTME algorithm developed, as well as the overall reflex module. While the OpenAI Gym Atari games were the primary testing environments used, the former also showed benefit in other application domains. Hence, those results are also presented here.

Multi-task MAP Elites Simulation Results

While the MTME algorithm was tested on Atari games, it was found to also show significant benefit over a current leading evolutionary approach in other applications. As such, results from tests in all these environments are presented here. The step-by-step procedure for the MTME algorithm was given above. However, for the purposes of implementation, step 5 required a means to select a task on which to potentially apply a given candidate solution. The following alternatives were considered here: (a) Evaluate the fitness of the candidate solution on every task in the task-descriptor space individually and select the task on which the candidate solution gives the greatest improvement in fitness. Each fitness computation here, however, counts as one evaluation towards the evaluation budget; (b) Simply choose a task at random on which to test a candidate solution; and (c) A tournament approach, where we randomly select a pool of tasks of size s, and then select the task closest to the first parent used in the evolution.

With each repetition of steps 3 to 9, a single evaluation is completed. To compare each task-selection approach fairly, we consider how the mean fitness over all the current elites in the map improves with respect to the number of evaluations, up to the predefined budget.

In order to show the versatility and applicability of this algorithm, we applied it to three different problems: (1) A planar kinematic arm that must reach a predefined target in the plane; (2) A 6-legged robot that must walk forward as fast as possible; and (3) Generating a repertoire of reflexes for OpenAI Gym Atari game environments.

Planar Kinematic Arm Problem

Figure 95 shows the schematic layout of the planar kinematic arm used for this problem scenario. The arm consists of 10 links, each having the same length, L, and each also having the same maximum angular range of α_{max}. The objective of any given task for the kinematic arm is to find the joint angles, $(\alpha_1, \alpha_2, \ldots, \alpha_{10})$, that bring the tip of the arm (i.e., the end-effecter) as close as possible to the predefined target, denoted by the red 'X'.

A task is defined by a particular combination of L and α_{max}. Each task is therefore a two-dimensional vector, resulting in a 2D task-descriptor space. A candidate solution for a task is a 10-dimensional vector, $(\alpha_1, \alpha_2, \ldots, \alpha_{10})$, consisting of the angular positions of each of the ten joints of the arm. The fitness function, $f(\alpha, \{L, \alpha_{max}\})$, is defined as the Euclidean distance from the position of the tip to the position of the target. The position of the target was selected arbitrarily, and is the same for all tasks.

For the purposes of the experiments, a total of 5,000 tasks were defined in such a way as to be spread evenly using CVT. A budget of one million evaluations was specified, and multiple methods

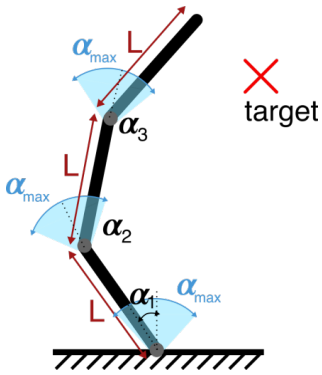


Figure 95: Schematic representation of the planar kinematic arm problem.

were used to find the best solutions for all these tasks within this budget. In particular, a total of 20 replicates of each of the following methods were conducted: (1) MAP-Elites with candidate solutions evaluated on all tasks (i.e., using approach (a) to select a task for a newly generated candidate solution). (2) MAP-Elites with candidate solutions evaluated on a randomly-chosen task (i.e., using approach (b) to select a task for a newly generated candidate solution). (3) MAP-Elites with candidate solutions evaluated on multiple tasks selected via a tournament (i.e., using approach (c) – this is the standard MTME approach). (4) Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [209] applied to each task separately to find a corresponding optimal solution (an independent instance of CMA-ES is run for each task, but all the CMA-ES instances are run in parallel). (5) Random sampling, where each candidate solution is constructed by a random selection of each of the ten angles, and tested on a random task. A solution is kept if it improves upon an existing elite.

Figure 96 shows the result of the experiments conducted. In particular, the plot on the left shows how the mean fitness over all 5,000 tasks change with increasing number of evaluations, up to the one million evaluation budget, for each of the four methods tested. The plot on the right shows the final mean fitness after all one million evaluations were complete. The solid line in these plots represents the median mean-fitness value over the 20 replicates, while the shaded regions around it indicate the interquartile range. Clearly, the MTME algorithm (which uses the tournament approach to matching candidate solutions to tasks) outperforms all the other methods with respect to the best mean fitness attained using the same computational budget.

It is interesting to note that MTME performed noticeably better than CMA-ES, recognized to be a particularly good evolutionary optimization method in the literature. Moreover, CMA-ES performed about as well as MAP-Elites with random task selection. This perhaps surprising result in fact lends further credence to the hypothesis of genome similarity among elites. While candidate solutions are applied to randomly-selected tasks, those solutions were created through the use of elite parents. If indeed the elites among different tasks share key solution characteristics, we should still see MAP-Elites arrive at good solutions to different tasks quickly, even with this random selection approach. And, indeed, this is what is observed, resulting therefore in performance that is very similar to independent task optimization via CMA-ES.

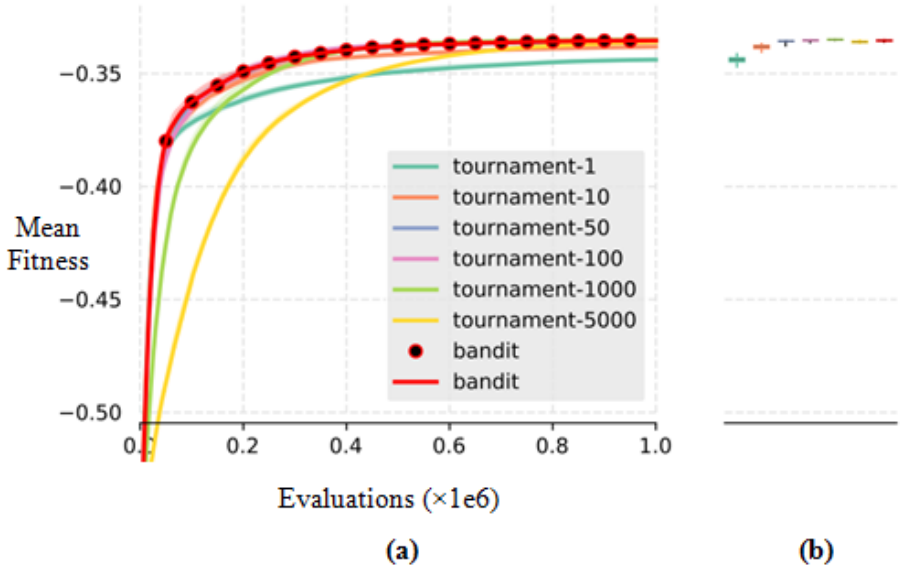


Figure 96: Test results on kinematic arm with variable morphology.

Not surprisingly, MAP-Elites with candidate solutions evaluated on all tasks, as well as random solution generation, both performed significantly poorly. For the former of these two, with each candidate-task pair evaluation counting toward the one million computational budget, only a few generations of evolution could be completed. Hence, the solutions do not experience enough evolution to improve much.

Six-Legged Locomotion Problem

A second, more challenging problem scenario was also used to test the MTME algorithm, involving finding optimal gaits to maximize forward speed in a six-legged robot. The robot was a simulated hexapod (simulated using PyBullet, [210]), where each of its six legs consisted of two segments. Hence, the robot was parameterized by the length of each segment of each leg, resulting in a 12-dimensioal task-descriptor. Figure 97 shows four different example morphologies of a hexapod robot.

MTME can be used here to build a repertoire of gaits for different morphologies. Each morphology can be viewed as a different damage recovery scenario, where different parts of each leg are damaged, resulting in having to find a new gait to maintain locomotion. This results in a repertoire of gaits for different anticipated damage scenarios.

The solution vectors, or genomes, for this problem scenario are the values of a 36-dimensional controller defined in previous work (for example, [211]). In summary, each leg has two joints, and each joint is controlled by a periodic smoothed square wave parameterized by three parameters: a phase shift, an amplitude, and an offset. A task is defined by a specification of each of the leg-segment lengths, with which the robot must move forward along the x-axis of a flat plane in the simulation from a starting position at the origin, (0, 0). The fitness function is defined as the distance moved along the x-axis until one of the following conditions is met:

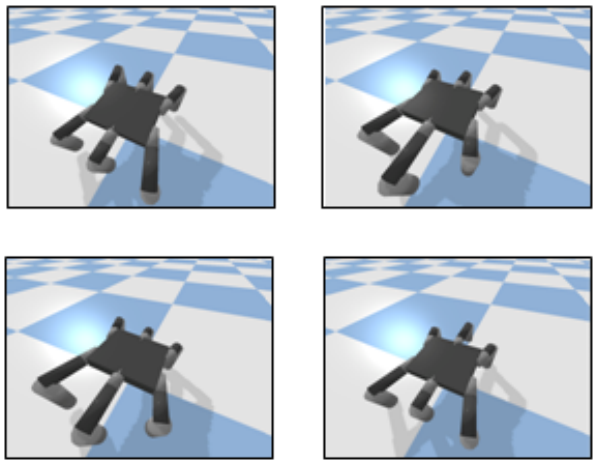


Figure 97: Example 6-legged robot morphologies.

the simulation lasts more than three simulated seconds;

the absolute value of the pitch or roll of the body exceeds $\pi/8$ (indicating that the body is not horizontal enough);

the absolute value of the y position is above 0.5 (indicating that the robot has turned too much).

For the experiments, a total of 2,000 tasks were generated by creating 2,000 morphologies of randomly selected leg-segment lengths. The same methods as those tested in the kinematic arm problem scenario were compared here once again, using a one million evaluation budget and 20 replicates. Figure 98 shows the results.

Similar to the first problem scenario, we see a noticeable benefit in using the MTME algorithm with tournament task selection for candidates compared to all other methods. As well, MAP-Elites with random task selection remains competitive with MTME, once again due to the MAP-Elite algorithm's ability to exploit the similarity of genomes of elite solutions to quickly find good solutions to other tasks, even if they are randomly selected for each candidate solution.

Surprisingly, we also see CMA-ES performing relatively very poorly, and about the same as MAP-Elites with candidate solutions tested on all tasks. While the sheer number of evaluations involved in the latter, and thus only a few evolutions being performed, explains the lower mean fitness over the map after the one million evaluation budget is used up, the comparable performance of CMA-ES is quite strange. One possible explanation could be due to the dimensionality of the problem. With 2,000 tasks, CMA-ES computes a default population of 14 for a 36-dimensional solution vector, which results in only 500 evaluations for each instance. This is much lower than what one would typically use, which is around 106 evaluations. Secondly, this 6-legged locomotion task is a significantly more complicated problem, with a less-smooth gradient requiring more global optimization.

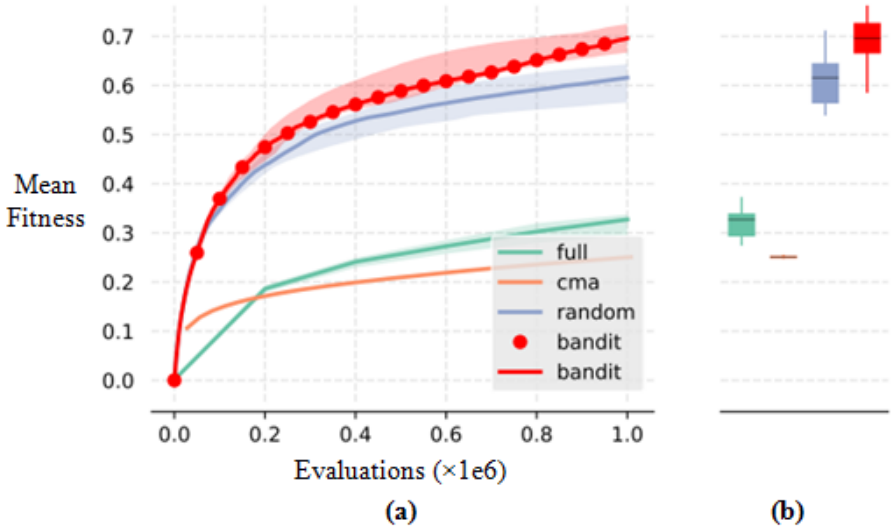


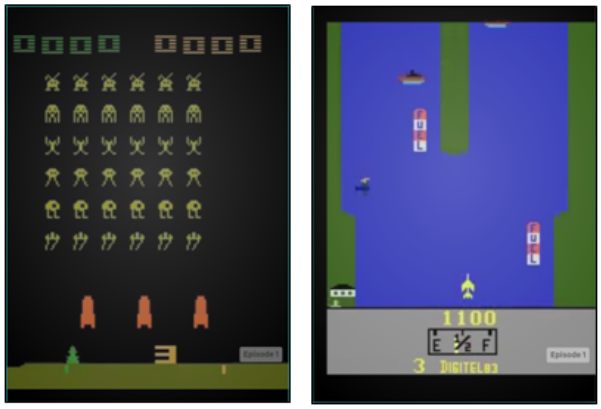
Figure 98: Test results on the 6-legged locomotion problem scenario.

Repertoire of Reflexes for OpenAI Gym Environments
Finally, we applied the MTME algorithm to generate a repertoire of reflexes for two OpenAI Gym Atari game environments: Space Invaders and River Raid (Figure 99).

In Space Invaders, the agent is situated at the bottom of the screen and can only move left and right. It must shoot at enemy targets that move in a simple pattern, and also avoid incoming fire. Three small, destructible barriers provide some protection if the agent moves underneath them. On the other hand, River Raid is a bit more challenging environment, in which the agent can move forward, backwards, left and right, thereby exploring any part of the screen. As it moves forward, the environment also changes introducing new mobile hostile agents as well as obstacles. The agent also has a limited fuel supply and must use fueling depots to maintain sufficient fuel as they appear on the screen. In both games, each agent is given a fixed number of lives, which are lost when the agent is struck by an enemy bullet, and, for the River Raid environment, when the agent collides with the side wall, an obstacle, or an enemy agent.

The objective here is to build a repertoire of reflexes, which are a string of action-sequences that are used to avoid emergency situations that occur in the games. These are defined as situations that result in the loss of life of the agent. Through numerous rollouts of game-play of both of these games, a set of 1,000 emergency situations were collected. An emergency situation is a saved game-state in which a loss of life is imminent within the next 20 time steps based on the agent's policy at the time (which was random). MTME is then tasked with finding high-performing action-sequences (i.e., reflexes) to avoid the imminent death in each emergency situation.

The task descriptor here consists of a two-dimensional vector of values representing a latent representation of the emergency-situation input information. In particular, each emergency situation is characterized by a stack of five consecutive observation images leading up to the beginning of the 20 time steps that led to the loss of life. Using the Uniform Manifold Approximation and Projection



(a) (b)

Figure 99: Sample screenshot from (a) Space Invaders and (b) River Raid OpenAI Gym Atari games.

(UMAP) algorithm [203], which is a general-purpose dimensionality-reduction algorithm, the set of all 5-image sequences representing the emergency situations for a given game were converted to a two-dimensional vector that still maintains the relative "distances" between emergency situations as determined from the input image information.

The genome is an N-dimensional vector of integers, where N is the length of the action-sequence that forms the reflex. Thus, a 20 time step reflex action-sequence would require a 20-dimensional vector of integers. Each integer value ranges from 0 to 5, and corresponds to a particular possible action in the game. The fitness function is the cumulative sum of rewards obtained over the course of the reflex action-sequence performed. If a death results, the fitness is automatically assigned a value of -1.

In the experiments conducted, MTME was compared with MAP-Elites using both random task selection and using testing on all tasks. Figure 100 shows how the mean fitness over the map varies with the number of evaluations. It may be surprising to see here that all three methods perform almost identically. The tournament-selection and random-selection methods rise at the almost the same rate and even reach the same final mean fitness after 200,000 evaluations. The all-tasks method rises slower, which is to be expected given the larger number of computations required per evolution, but also manages to reach the same final mean fitness after using up its evaluation budget.

However, there is a very plausible explanation for why the MTME did not clearly outperform the other methods in this problem scenario. Simply put, the Atari game environments do not present very challenging emergency situations. With only a 20 time step policy to plan and very few actions and, thus, few variations in outcomes, one can very quickly reach a high-performing solution for which there is not sufficient game complexity to be improved upon. As such we see all methods reach almost the same mean fitness after only 200,000 evaluations.

In light of this revelation, a second set of experiments were also tried on the more complicated of the two games, River Raid, in which the reflex length was increased 5-fold to 100 time steps. It

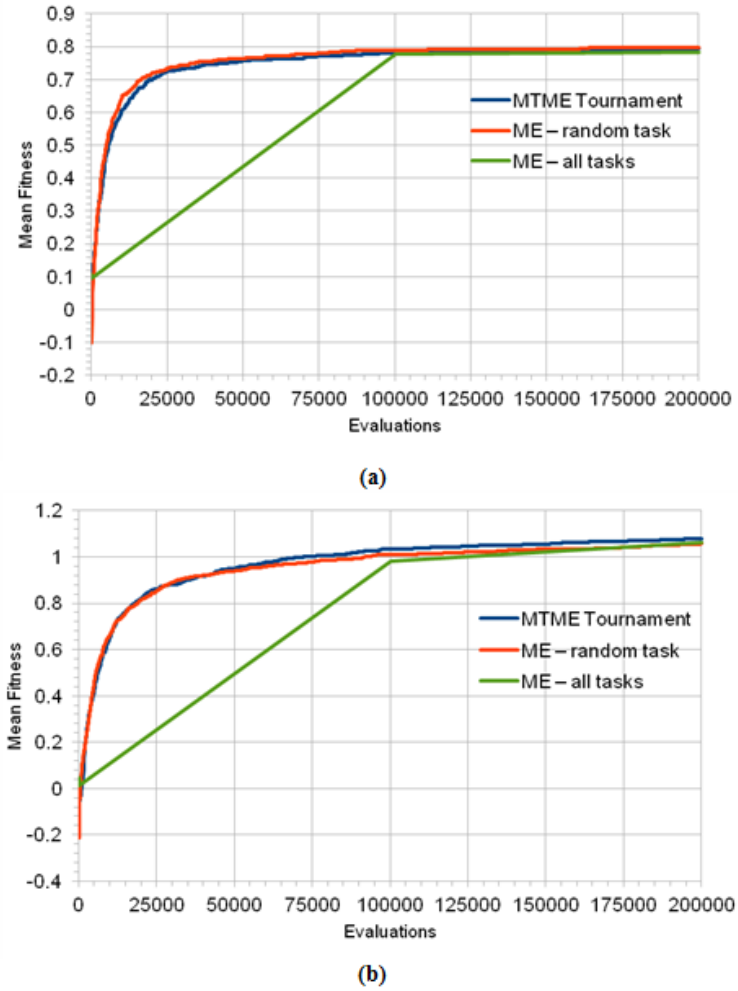


Figure 100: Test results for a 20-timestep reflex length.

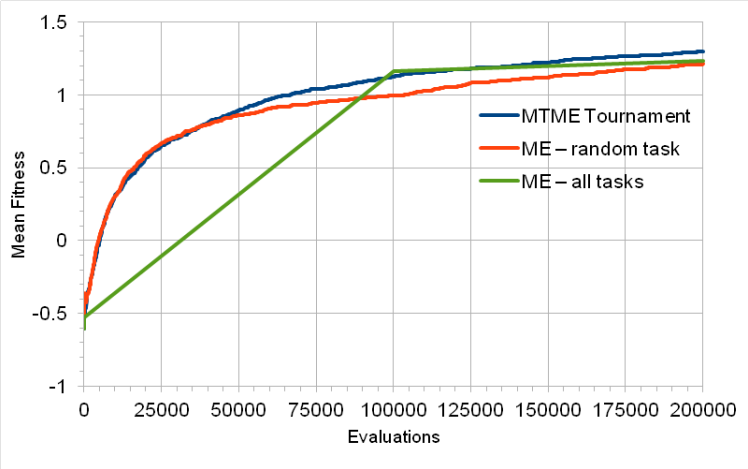


Figure 101: Test results for a 100-timestep reflex length.

was hypothesized that by extending the length of the planning horizon one would encounter more variability in outcomes and thus see a greater difference in performance amongst the methods tried. Thus, the three methods were repeated with a 100-timestep reflex length (i.e., genome size) and the results are shown in Figure 101.

We see once again from the results in Figure 101 that even with the longer planning horizon, there is no significant differentiation in the performance of the three methods. It is possible that a much longer planning horizon policy could eventually begin to create more variations in the game-play so as to put the methods to a better test, however, at this point, it ceases to become a reflex-maneuver and starts to become more of a long-term game-play policy.

As a final point of investigation, we also tried using the 5-observation image-stack directly as the task-descriptor (by flattening all the corresponding pixel values into a single, 1D vector for each emergency situation). By changing the task-descriptor it would change the distance measure used in finding a task for a candidate solution. By using the input 5-observation images directly we would be certain that no critical information is being lost compared to using UMAP, which could very well be producing misleading task-descriptors. With the significant increase in task-descriptor dimensionality, the all-task MAP-Elites method was found to produce memory deficiency errors and, thus, was not included in these tests. Nevertheless, we still see from the results in Figure 102 that this did not have any significant impact on relative performance of the two methods compared to what was observed in earlier test.

Nevertheless, the above results are still a confirmation of the ability of the MTME algorithm to quickly find effective, high-performing solutions to address emergency situations in Atari games. Of course, with elite genomes sharing many key features, even a random task-selection approach can quickly find high performing solutions that cannot be easily improved on. The resulting archive of genomes, then, that results after the 200,000 evaluation budget represents the repertoire of reflexes for the two games tested here. Hence, we can confidently say that the MTME algorithm devised here can successfully develop a repertoire of reflexes in Atari game environments.

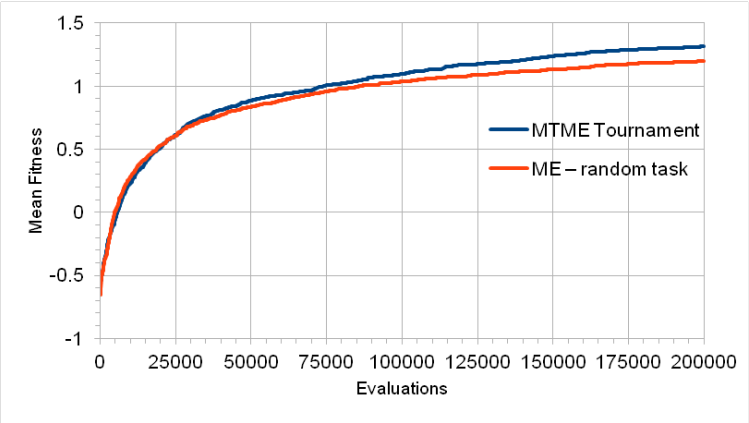


Figure 102: Test results using the 5-image emergency situation data directly as a task-descriptor.

Overall Reflex Module Tests

In order to conduct overall tests of or reflexive adaptation approach, a standalone reflex module was developed using the latest approaches for the components involved, coded in Python. It is designed to work with OpenAI Gym environments. Considerable effort was placed in incorporating all the components seamlessly into a single module that can be used to perform prediction to detect emergency situations, and employ the BO approach to find reflexes online to circumvent them.

While the module also includes functions that can be called to implement MTME to construct an archive of priors, a reliable and effective method for conducting reflex retrieval still needs to be established. Results of investigations presented above showed that the simple approach of using a nearest-neighbor search on latent representations of the emergency situations to perform the matching did show some promise. Also, a CNN-based classifier could successfully be constructed and trained to map emergency situations to action-sequences. However, more research is necessary to properly test this approach. Due to time constraints and re-focusing of priorities, more effort was directed toward developing a working solution for reflex generation, and establishing the CARLA autonomous driving simulator as a viable environment for future work and validation. This, and the fact that reflex retrieval is an optional component that could potential expedite the BO approach, but is not mandatory, it was left out of the overall reflex tests that were conducted.

Moreover, it is also clear that conducting reliable future prediction for the purposes of generating reflexes continues to remain a very challenging problem that will require more time to develop. Hence, in the experiments conducted herein, an ideal predictor was employed as a temporary place-holder, where another game instance was used to rollout reflex action-sequences in order to test them and obtain corresponding fitness values. In addition, seeding the BO search instances was done purely through a random selection of the initial samples.

The overall reflex module tests involved employing the reflex module in a monitoring roll, as intended. The main agent is tasked with playing an Atari game for the first time, and is thus randomly initialized. The scenario is therefore an agent introduced to a new environment and tasked with learning the best policy to continue playing the game as along as possible, until all its lives are exhausted. As with earlier tests that were conducted, the ACER RL algorithm is used to train

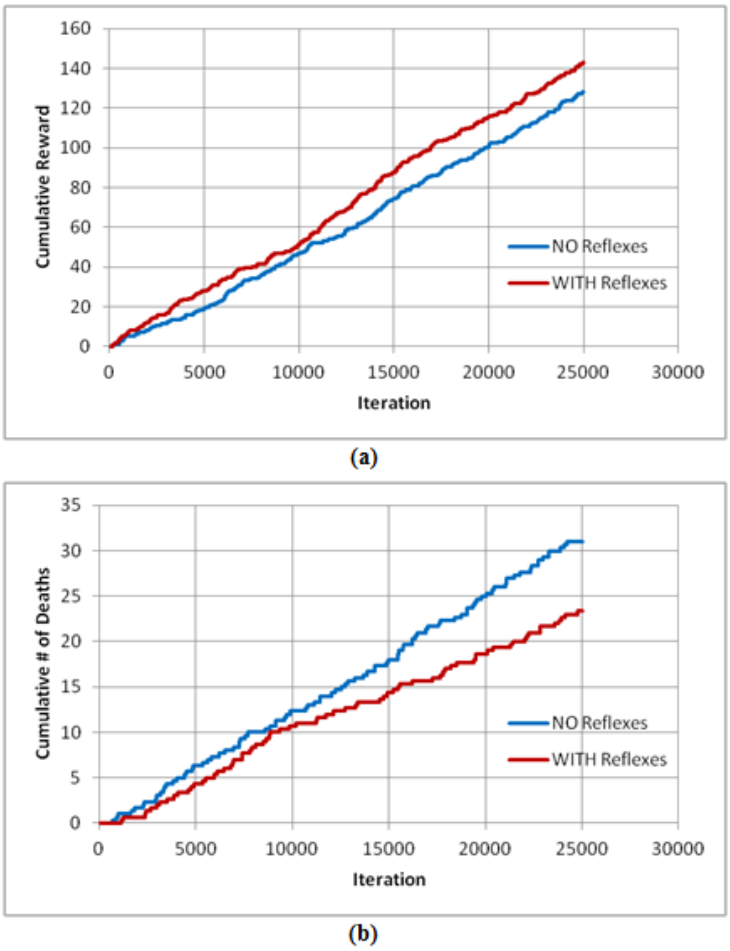
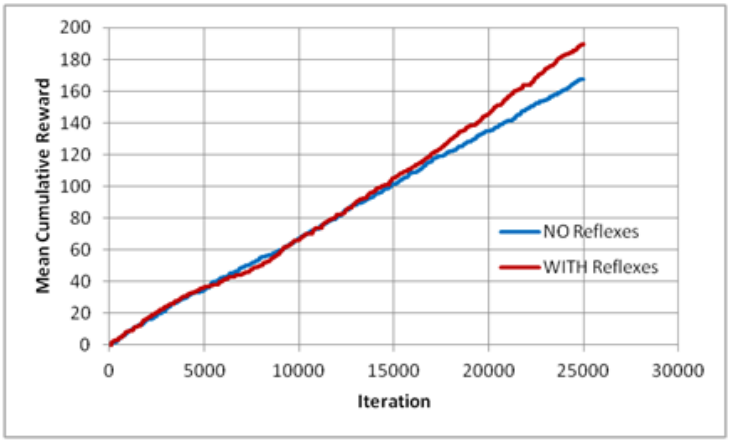


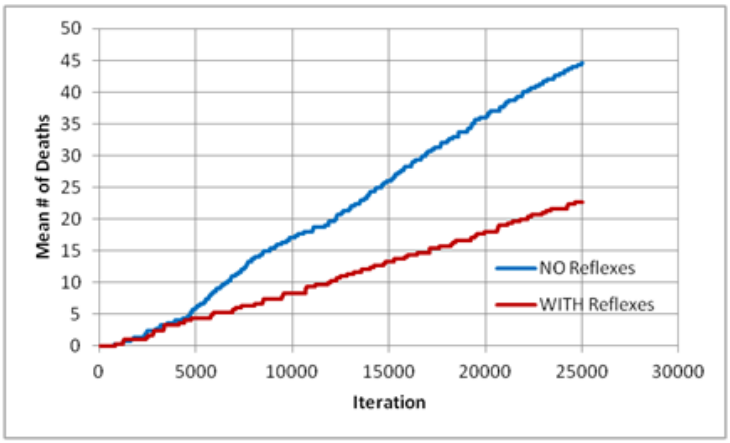
Figure 103: Reflex module performance results on the Space Invaders Atari game.

the agent on the game. The Stable Baselines implementation of the ACER algorithm was used to conduct the training, with the agent using the provided CNN policy model [212]. Since reflexes are most useful during the initial period of training when an agent has just been introduced to a new environment or task, the impact of the reflex module was evaluated during the first 25,000 iterations of training. Two scenarios were simulated, one in which the reflex module was used and one with the agent trained without its presence. Three replicates for each scenario were simulated on two different games: Space Invaders and River Raid. Figure 103 shows the results for Space Invaders while Figure 104 shows them for the River Raid game.

In Figure 103(a) we see the mean cumulative reward over the 25,000 training iterations for the two scenarios in the Space Invaders game environment. In terms of the average percentage increase in the mean cumulative reward over these iterations, the presence of reflexes was found to provide a 24.7% improvement. This benefit is seen to arise relatively early on in the training and gradually increases. Comparatively, in Figure 103(b) we see the development of the mean cumulative number of deaths among the three replicates over the 25,000 iterations, which, on average, showed a similar percentage improvement of 25.3% (i.e., there were, on average, 25.3% fewer deaths when reflexes



(a)



(b)

Figure 104: Reflex module performance results on the River Raid Atari game.

are used compared to when they are no). Clearly, reflexes do have a marked impact on reducing how often the agent dies, and this, in turn, is reflected in the increase in cumulative reward achieved by the agent.

Similarly, in Figure 104(a), the mean cumulative rewards for the two scenarios over the 25,000 iterations of training are plotted. Here, however, we see a much smaller improvement when reflexes are used; on average, about 3.9%. Moreover, we see that this benefit is not immediately realized, but rather slowly develops over time. However, in Figure 104(b) we see the development of the mean cumulative number of deaths among the three replicates over the 25,000 iterations, which, on average, showed a much more significant 41.0% improvement. Indeed, the main purpose of the reflex module is to help keep the agent alive as it learns a new task, and thus, sure enough, the largest impact is clearly on the cumulative number of deaths observed.

What we can see, then, is that performance can vary with the particular environment or application used, but the presence of the reflex module seems to always provide benefit in terms of reducing the deaths, or catastrophic events, experience by the agent learning a new task or environment. More complicated environments, such as the relatively harder game, River Raid, can still make obtaining rewards difficult, but this is really the job of the underlying training algorithm to bring the agents performance up as it learns over time. If the reflex module can keep the agent alive for longer as this is happening, the agent will have more opportunities to learn, and will remain in its environment for longer periods of time, thereby experiencing more and newer environment states (i.e., richer learning data), and, thus, the reflex module will have served its purpose.

Clearly, then, the concept of reflexive adaptation has the potential to provide a significant benefit to L2 in an autonomous agent. A better quantification of the benefit that reflexes are providing could perhaps be developed, other than simply the cumulative rewards. Still, our preliminary work with incorporating the BO reflex generation approach to the CARLA autonomous driving environment has shown much more potential for showing the benefit of reflexes. Not only does it provide a more complicated and realistic environment for the testing of L2 methods, wherein real driving scenarios complete with different road and traffic conditions, pedestrians, and customized emergency situations can be simulated, it also facilitates the incorporation of Python code to carry out agent training. This will therefore be the primary environment in which further development and validation of the reflex module will take place. It should also be noted that there is still much to explore with the current reflexive adaptation approach, including how the MTME algorithm and reflex retrieval can best facilitate L2 as new tasks are encountered, as well as the impact of varying key parameters in the BO algorithm.

1.4.7 Probabilistic Program Neurogenesis

We compared the performance of our approach to those of the GA using the experimental setup described above. We ran a total of 150 trials for the two approaches combined. Each trial started with a newly initialized population, and parameters in the case of PPL, and then the optimization process was run for 21 minutes of wall-clock time. Wall-clock time was used because both the GA and our approach are randomized search algorithms and on average one generation of our approach takes longer than the GA due to the probabilistic inference. The average accuracy attained on the new

task by the solutions from our approach was 87.7% and for the GA it was 87.6%. The relationship between accuracy and fitness is shown in Equation 41. The fact that the accuracy is close to that of the parent network on the old task (87.4%) illustrates the effectiveness of neurogenesis for continual learning. The typical solution found by Probabilistic Program Neurogenesis (PPN) was 12-17 new neurons added only to the first layer. The typical solution found by the GA had greater variability and was either 11-20 new neurons added only to the first layer or 13-20 new neurons added only to the second layer. Based on a large number of experimental trials, we observe that the global optimal solution is approximately 16 new neurons added only to the first layer.

The results of the first analysis are shown in Figures 55 and 56. Each figure shows the average best fitness achieved by the PPN (blue) and GA (red) as a function of elapsed run time in minutes. The fitness values used for plotting have been shifted by -8.0 and scaled by 100.0. This was done solely for the purpose of improving visual interpretability of the results. And the error bars are 90% confidence intervals. Figure 55 is from 2 to 5 minutes and the Figure 56 is from 5 to 21 minutes. Based on the curves in Figure 55, it can be seen that on average PPN reaches near optimal solutions (a fitness of about 60.0) within the first 2 minutes of simulation time, whereas it takes the GA about 5 minutes to reach a comparable level of fitness. Our approach starts much higher due to the informative prior in the probabilistic program. As a point of reference, when a parent network makes predictions on the new task prior to neurogenesis ($complexity = 0$ in Equation 41) the expected fitness is -400 ($accuracy = 0.4$). Figure 56 shows that in the long run our approach continues to improve and outperform the GA.

The next analysis examined the consistency with which the PPN and GA were able to find solutions that achieved particular fitness levels. Figure 57 shows the fraction of trial runs on which the best fitness found exceeded various lower bounds. The results for the PPN are in blue and those of the GA are in red. We can see that for each fitness lower bound on the x-axis, our approach exceeds the success frequency of the GA, and for the higher (more difficult to achieve) fitness levels (> 61) the success rate of our approach is at least double that of the GA. These results demonstrate that our approach consistently finds better solutions than the GA.

We have demonstrated the effectiveness of our approach by showing that it is able to consistently find better performing solutions than the GA and it is able to do so faster. We believe that one of the primary reasons for this is that the probabilistic program represents a distribution over distributions, which promotes exploration and thus the discovery of better solutions more quickly. As can be seen in Figure 53, the parameters (p_s, p_n, p_a) of the distributions governing the observable random variables (Ind_{max}, L, N_m), which control the structural attributes of an individual, are themselves (latent) random variables. When running the program in the forward direction to generate a new individual, we must sample the latent random variables first and then use them to determine the distribution over the observables.

Another reason for the effectiveness of our approach is that it is easy to incorporate prior knowledge about a domain directly into the search process. Our probabilistic program, rather than using crossover and mutation operators to implicitly define the search distribution, enables the designer/user to build in semantics that are believed to be important. We were able to take advantage of this with the portion of the program that selects the layers that are permitted to receive new

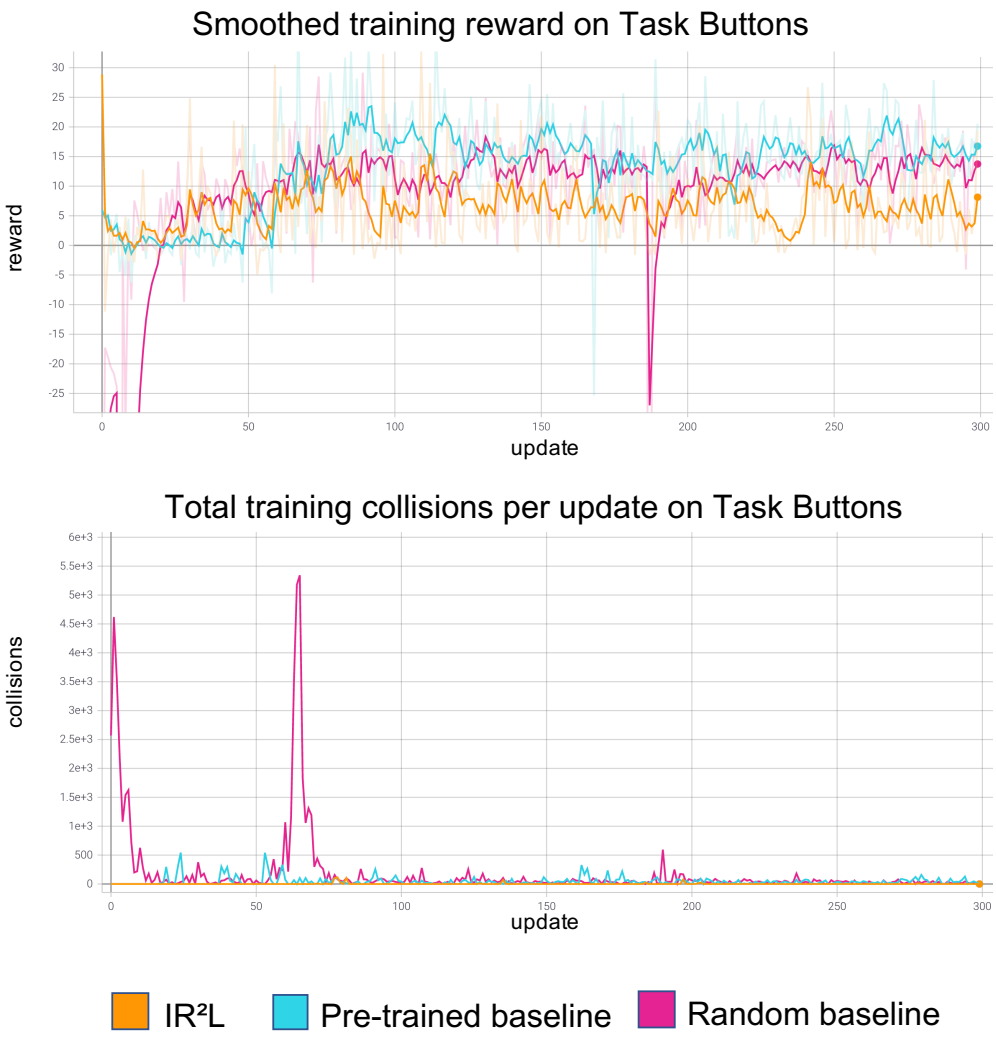


Figure 105: Reward and collisions during training in the “buttons” task.

neurons. Our original formulation of the PPL did not have this logic, but we quickly realized based on empirical observations that it is important to efficiently explore regions of the search space where one or more layers do not receive any new neurons. Subsequently, a simple modification to our program enabled this enhancement.

1.4.8 Meta-Learned Instinct Network

We compared the final task reward and the number of collisions during training between IR^{2}L that uses a pre-trained instinct network and two baselines that do not use an instinct network. The first is a random baseline, i.e., a randomly initialized policy that has to learn the “buttons” and “push” tasks from scratch. The second one represents a baseline that undergoes pre-training on “goal” task and then transferred to the other two tasks. The idea is that pre-training a policy, even without an instinct network, should reduce safety violations when learning another task afterward. The task reward

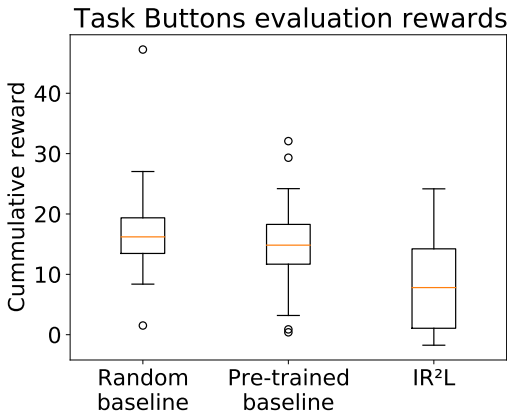


Figure 106: Cumulative reward for the final baselines and IR^2L on the "buttons" task.

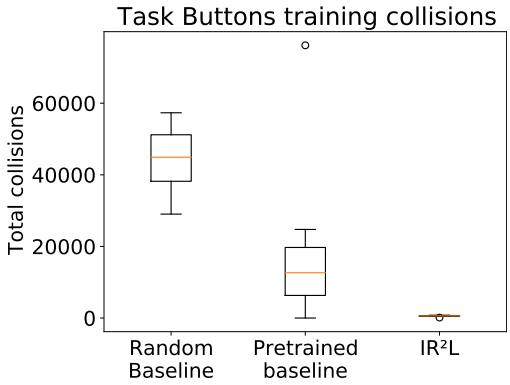


Figure 107: Cumulative collisions on training runs for the final baseline and IR^2L on the "buttons" task.

during experiments is $r_t^*(s) = r_t(s) - h(s)H_t$, where $r_t(s)$ is the task-type specific reward, and $h(s)$ is a binary function indicating hazard violations. Hyperparameter H_t is a task-specific punishment for colliding with a hazard. It was chosen to optimize baseline's learning to avoid hazards while still being able to solve the tasks. We found that $H_t = 1$ and $H_t = 10$ works best for baseline training on the "buttons" and "push" tasks, respectively. We used the same task reward for baselines and IR^{2}L. Videos of the resulting behaviors can be found: https://youtu.be/lqRvHimqvAc.

Transfer to "Buttons" Task

We trained the baselines and IR^{2}L for 300 training epochs with each epoch having 215 sampled trajectories in the training buffer (Figure 105). This setup leads to a significant reduction in training collisions. In the case of baseline, the noise in the training curves is due to reward punishments caused by colliding with hazards, while in the case of instincts they can be the results of an agent unable to move, with hazards blocking the path to the correct button. The rewards are measured on a single episode with a deterministic policy after a weight update, while collisions are measured during the sample collection before the update.

The reward calculations during training (Figure 105) also include the hazard violations. To better distinguish the performance on the task and the ability to avoid hazards, we evaluated the models on 50 episodes purely on the task rewards (hazards not accounted for). The baseline agents move

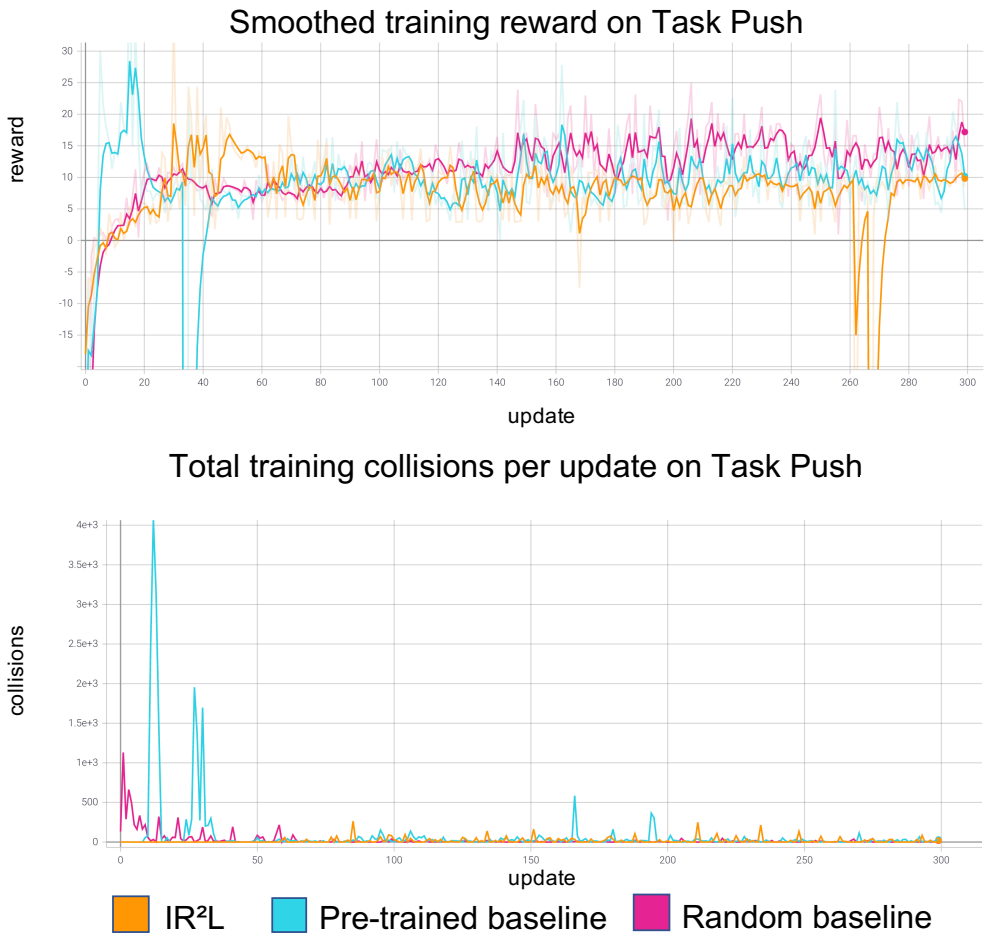


Figure 108: Rewards and collisions during training on the “buttons” task.

faster than the IR^{2}L agent, which is moving more carefully and thus receives a slightly reduced task return (Figure 106).

In the original Safety Gym paper [97], the benchmarks for the vanilla “buttons” type task show cumulative rewards to be 25 on average with 200 collisions per episode on average for unconstrained methods (the task reward here does not include hazards punishments). Constrained methods show the cumulative reward to be between 0 and 5 on average, while collisions to be 25 per episode on average during training.

We repeated the “buttons” task experiment 10 times and plotted the cumulative hazard collisions during the learning period for the baseline policies and IR^{2}L (Figure 107). Due to the stochastic nature of the task and imperfect instinct training during the pre-training phase, IR^{2}L performs some hazard violations but shows substantial improvements over the two baselines. As expected, the pre-trained baseline shows a strong transfer of hazard-avoiding skills to the new task compared to the random baseline.

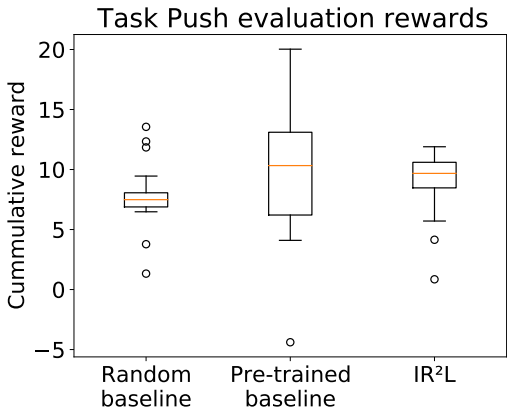


Figure 109: Cumulative reward for the final baseline and IR^{2}L on the "push" task.

Transfer to "Push" Task

The number of training epochs and samples is the same as for the "buttons" task (Figure 108). Training with instinct protection allows the policy to almost completely avoid hazards. Not surprisingly, the random baseline displays a large spike of hazard collisions at the start of training until it start learning to avoid them. The pre-trained baseline also shows large spikes in collisions during training since there is nothing to protect a stochastic policy from stepping over hazards in the dense grid of hazards present in the "push" task. The task reward over training steps shows a similar performance between all methods. Large dips in task reward are sometimes observed during training, which is due to the agent occasionally stepping over hazards and receiving a punishment of $H_t = 10$. The dips are not visible in the collisions plot since the rewards are measured on a single episode with a deterministic policy after a weight update, while collisions are measured during the exploration phase before the update. The Safety Gym benchmarks [97] for the vanilla "push" type task show cumulative rewards to be 7 on average with 40 collisions per episode on average. Constrained methods show the cumulative reward to be 2 on average, while collisions to be 25 per episode on average during training. The "push" task implementations in Safety Gym and here are significantly different so the numbers are not perfectly comparable.

The result of 50 evaluation episodes of the final policy is shown in Figure 109. The IR^{2}L approach shows a similar final median performance on the task reward as the b aselines. The pre-trained baseline is showing better performance in some episodes compared to the random baseline and IR^{2}L, likely due to a large skill transfer between "goal" and "push" tasks. We also repeated the "push" task experiment 10 times and plotted the cumulative hazard collisions during learning for the baseline policies and IR^{2}L (Figure 110). The pre-trained baseline has a strong hazard-avoiding and goal-reaching skill transfer. Still, the IR^{2}L method has substantially better hazard-avoiding skills while maintaining similar reward returns on average. We observe that the random baseline policy is very quick to clear the hazards grid and reach the box and the goal on the other side but has a much higher risk of hitting the hazards. The instinct-protected policy is, on the other hand, more careful around the hazards grid. Figure 111 shows a trajectory of the agent with high instinct activation within the hazards grid, keeping the agent safe from collisions. Even with the relaxation of the task with separate hazard and box/goal zones, the task is challenging to learn for the baseline and IR^{2}L.

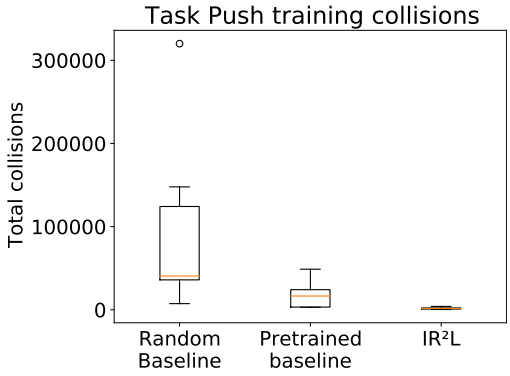


Figure 110: Cumulative collisions on training runs for the final baseline and IR^{2}L on the "push" task.

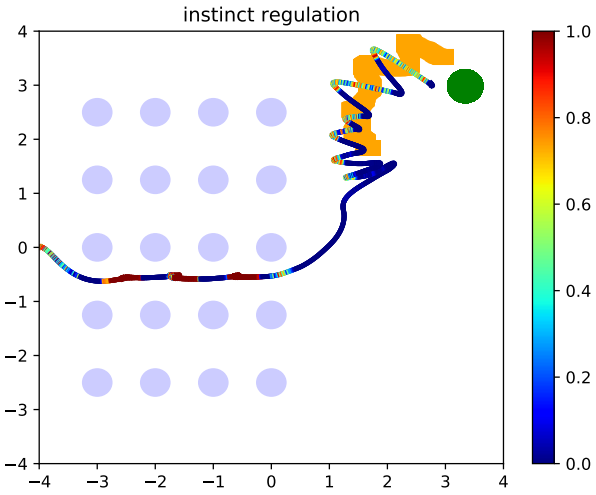


Figure 111: Agent trajectory on the "push" task.

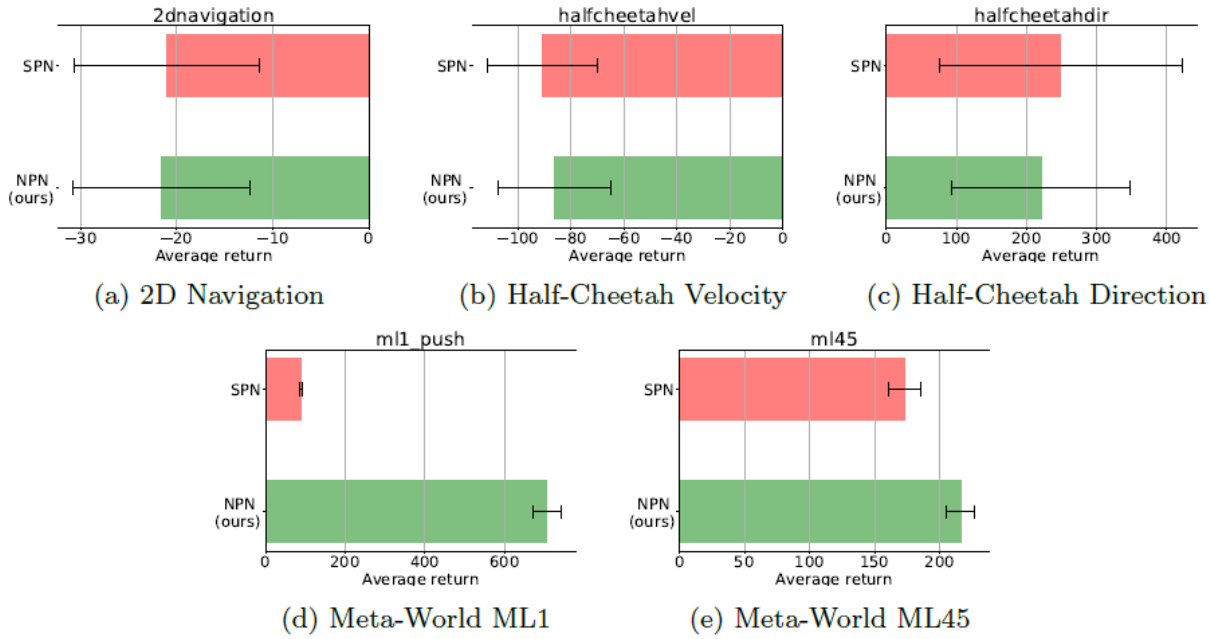


Figure 112: Adaptation performance across tasks of SPN and NPN in CAVIA framework.

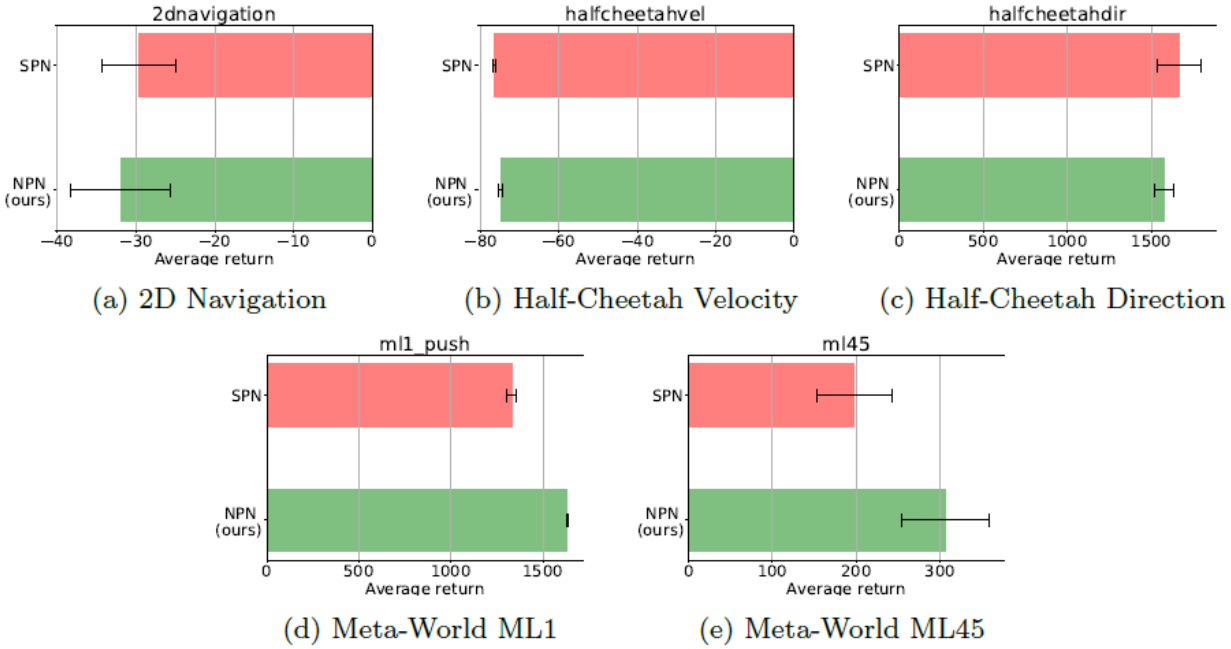
1.4.9 Plastic Neuromodulated Network

Here we present the results of the neuromodulated policy network evaluations across high-dimensional discrete and continuous control environments with varying levels of complexity are presented. The continuous control environments are the simple 2D navigation, the Half-Cheetah direction [82] and velocity [82] Mujoco [213] based environments and the Meta-World ML1 and ML45 environments [214]. The discrete action environment is a graph navigation environment that supports configurable levels of complexity called the CT-graph [215, 1]. The experimental setup focused on investigating the beneficial effect of the proposed neuromodulatory mechanism when augmenting existing meta-RL frameworks (i.e., neuromodulation as complementary tool to meta-RL rather than competing). To this end, using CAVIA meta-RL method [122], a Standard Policy Network (SPN) is compared against the Neuromodulated Policy Network (NPN) across the aforementioned environments. Similarly, SPN is compared against NPN using the PEARL meta-RL method [123] only in the continuous control environments because the soft actor-critic architecture employed by PEARL is designed for continuous control.

Performance

The experimental setup for CAVIA and PEARL as in [122] and [123] were followed. For PEARL, neuromodulation was applied only to the actor neural network. The performance reported are the meta-testing results of the agents in the evaluation environments after meta-training has been completed (Figures 112, 113, 114 and 115). During meta-testing in CAVIA, the policy networks were fine-tuned for four inner loop gradient s teps. Lastly, depending on the evaluation environment, the metric used to judge evaluation performance was either return or success rate.

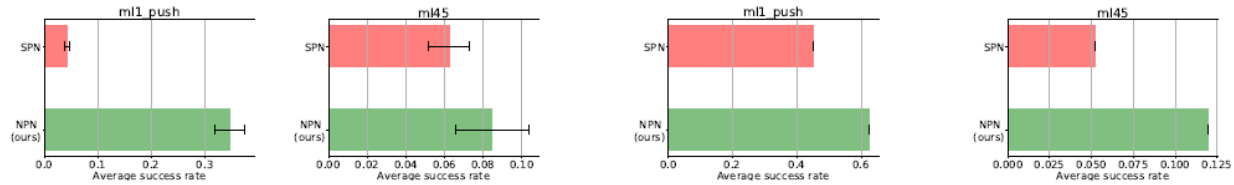
2D Navigation Environment



(a) 2D Navigation (b) Half-Cheetah Velocity (c) Half-Cheetah Direction

(d) Meta-World ML1 (e) Meta-World ML45

Figure 113: Adaptation performance across tasks of SPN and NPN in PEARL framework.



(a) CAVIA, ML1 (b) CAVIA, ML45 (c) PEARL, ML1 (d) PEARL, ML45

Figure 114: Adaptation performance, based on success rate metric, across tasks of SPN and NPN in CAVIA and PEARL.

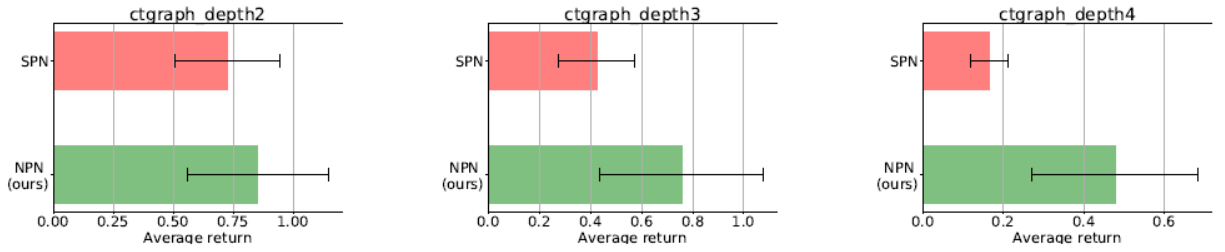
The first set of simulations were done in the 2D point navigation experiment introduced in [82]. An agent is tasked with navigating to a randomly sampled goal position from a start position. The goal position is sampled from the interval [-0.5, 0.5]. The reward function is the negative squared distance between the current agent position and the goal. An observation is the agent's current 2D position while the actions are velocity commands clipped at [-0.1, 0.1]. The result of the meta-testing performance evaluation comparing both the standard policy network and neuromodulated policy network is presented in Figure 112(a) for CAVIA and Figure 113(a) for PEARL. The result shows that both policy networks had a relative good performance. Such optimal performance is expected from both policies as the environment is simple and the dynamic representations required for each task are not very distinct.

Half-Cheetah

Half-Cheetah is an environment based on the MuJoCo simulator [213] that requires an agent to learn continuous control locomotion. We employ two standard meta-RL benchmarks using the environment as proposed in [82]: (i) the direction task that requires the cheetah agent to run either forward or backward and (ii) the velocity task that requires the agent to run at a certain velocity sampled from a distribution of velocities. Although challenging (due to their high-dimensional nature) in comparison to the 2D navigation task, these benchmark are still simplistic as the direction benchmark contains only two unique tasks and the velocity benchmark samples small range of velocities ($[0, 2.0)$ or $[0, 3.0)$). Therefore, the optimal policies across tasks in these benchmarks possess similar representations. The results of the experiments for both benchmarks are presented in Figures 112(b) and 112(c) for CAVIA, and Figures 113(b) and 113(c) for PEARL. Unsurprisingly, the results show comparable level of performance between the standard policy network and the neuromodulated policy network across CAVIA and PEARL. These benchmarks are of medium complexity and the optimal policy for each task is similar to others.

Meta-World

The neuromodulated policy network was next evaluated in a complex high-dimensional continuous control environment called Meta-World [214]. In Meta-World, an agent is required to manipulate a robotic arm to solve a wide range of tasks (e.g., pushing an object, pick and place objects, opening a door and more). Two instances of the benchmark (namely, ML1 and ML45) were employed. In the ML1 instance, the robot is required to solve a single task that contains several parametric variations (e.g., push an object to different goal locations). The parametric variations of the selected task are used as the meta-train and meta-test tasks. ML45 is a more complex instance that contains a wide variety of tasks (each task with parametric variations). It consists of 45 distinct meta-train tasks and five distinct meta-test tasks. The standard policy network and neuromodulated policy network were evaluated in ML1 and ML45 instances using CAVIA and PEARL. The results are presented in Figures 112(d) and 112(e) for CAVIA, and Figures 113(d) and 113(e) for PEARL. In these complex benchmarks, the results show that the neuromodulated policy network outperforms the standard policy network in both CAVIA and PEARL, highlighting the advantage neuromodulation offers in complex problem settings. In addition to judging the performance based on reward, results are also presented using the success rate metric (introduced in [214] as a metric about whether or not an agent is able to solve a task) in Figure 114. The results again show that the neuromodulated policy network achieved significantly higher average success rate both in CAVIA and PEARL in comparison to the standard policy network.



(a) CT-graph $(b = 2, d = 2)$ (b) CT-graph $(b = 2, d = 3)$ (c) CT-graph $(b = 2, d = 4)$

Figure 115: Adaptation performance across tasks of SPN and NPN for discrete control in CAVIA.

CT-graph

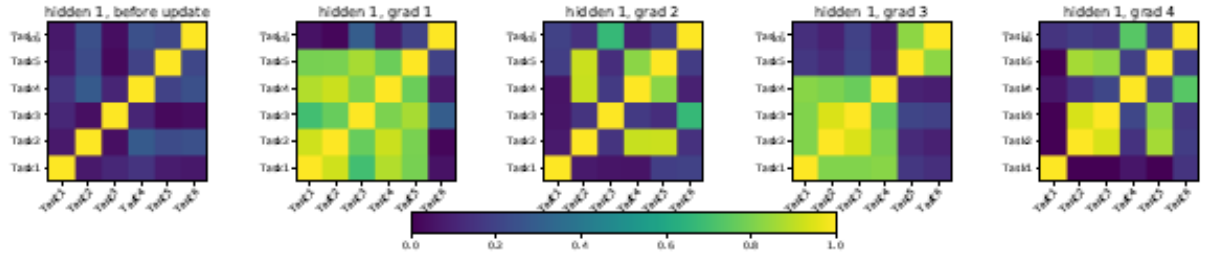
The CT-graph is a sparse reward discrete control graph environment with increasing complexity that is specified via parameters such as branch b and depth d. An environment instance consists of a set of states including a start state and a number of end states. An agent is tasked with navigating to a randomly sampled end state from the start state. The three CT-graph instances used in this work were setup with varying depth parameter. With increasing depth, the sequence of actions grows linearly, but the search space for the policy network grows exponentially. The simplest instance has d set to 2 (CT-graph depth2), and the next has d set to 3 (CT-graph depth3) and the most complex instance has d set to 4 (CT-graph depth4). The meta-testing results are presented in Figure 115. The results show a significant difference in performance between standard and neuromodulated policy networks. The optimal adaption performance from the neuromodulated policy network stems from the rich dynamic representations needed for adaptation as discussed below.

Analysis

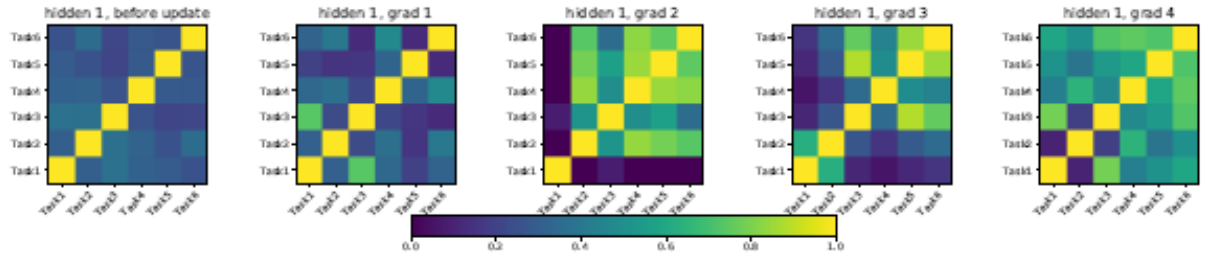
We conducted analysis on the learnt representations of the standard and neuromodulated policy networks for tasks in the 2D Navigation and CT-graph environments. The policy networks trained using CAVIA was chosen for the analysis as the single neural component in CAVIA (i.e., the policy network) makes it easier to analyze in comparison to PEARL, which contains multiple neural components. Furthermore, PEARL experiments were conducted only in continuous control environments (similar to the original study), whereas CAVIA experiments covered both discrete and continuous control environments. Hence, analysis in CAVIA allowed for more coverage across benchmarks.

To measure representation similarity across task, we employ the use of the Centered Kernel Alignment (CKA) [216] similarity index, comparing per layer representations of both standard and neuromodulated policy networks across different tasks. There exist several similarity index measures such as Canonical Correlation Analysis [217], Representation Similarity Analysis (RSA) [218], Hilbert-Schmidt Independence Criterion (HSIC) [219], and more.

The principle behind CKA is the generation of a similarity measure between two representations by comparing their similarity structure. Each similarity structure is produced from the measure of similarity between pairwise examples or data points in a representation. Furthermore, CKA is a generalized extension of HSIC, with the inclusion of normalization that introduces the property



(a) standard policy network, first hidden layer.



(b) neuromodulated policy network, first hidden layer.

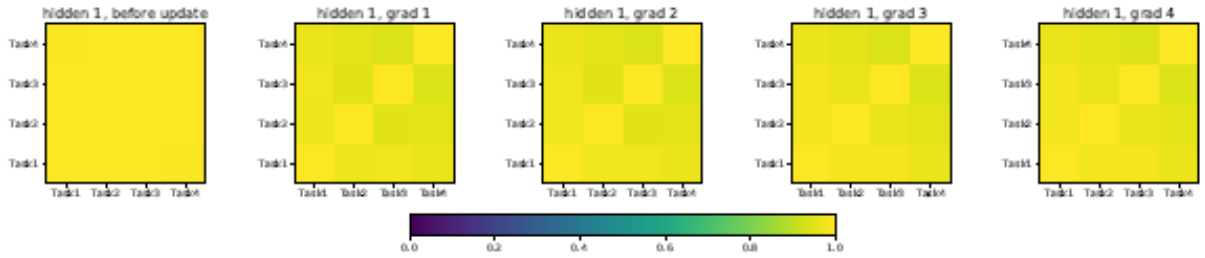
Figure 116: Representation similarities between tasks in the 2D navigation environment.

of isotropic scaling invariance. While the RSA similarity index measure employed in [220] is a valid alternative, we chose CKA as it is known to be robust to random initialization and can enable comparison within layers of the same network as well as comparison across networks. Furthermore, CKA has been employed previously in meta-RL settings (e.g., [221]).

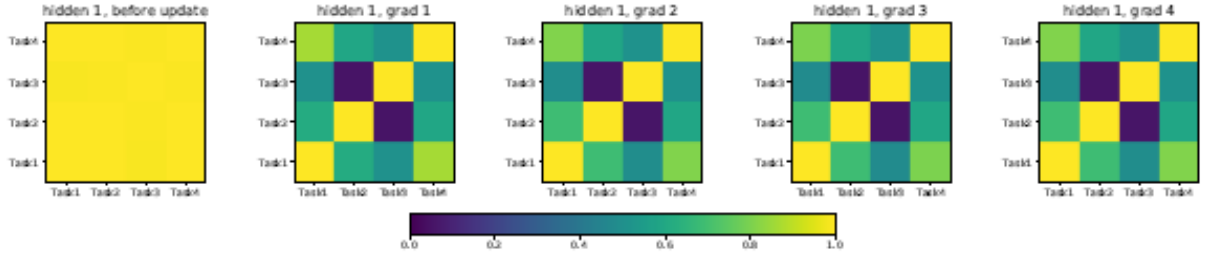
Representation similarities for standard and neuromodulated policy networks across tasks The per layer output representation similarities between tasks were plotted as heatmaps in Figure 116 and 117. Each heatmap in a row (for example 116(a)) depict the similarity before or after few steps of gradient updates to the layer. Before any gradient updates, the representations are similar between the tasks. After gradient updates, some dissimilarities between tasks begin to emerge.
2D Navigation. For the simpler 2D navigation environment, the plots for the first hidden layer of

the standard policy network shown in Figure 116(a) depict good dissimilarity between tasks, thus highlighting the fact that the learnt representations are sufficient to produce distinct task behaviors. The same is true as well for the first hidden layer of the neuromodulated policy network (Figure 116(b)). This further justifies why both policies obtained roughly comparable performance in this environment. The simplicity of the problem enables distinct task representations to be obtained easily.

CT-graph. In Figures 117(a) and 117(b), we compare the representation similarity of the first hidden layer of the standard and neuromodulated policy networks in the CT-graph depth2 environment. We see that representations of the neuromodulated policy are more dissimilar between the tasks than those of the standard policy. Due to the complexity of the environment, the task-specific



(a) standard policy network, first hidden layer.



(b) neuromodulated policy network, first hidden layer.

Figure 117: Representation similarities between tasks in the CT-graph depth2 environment.

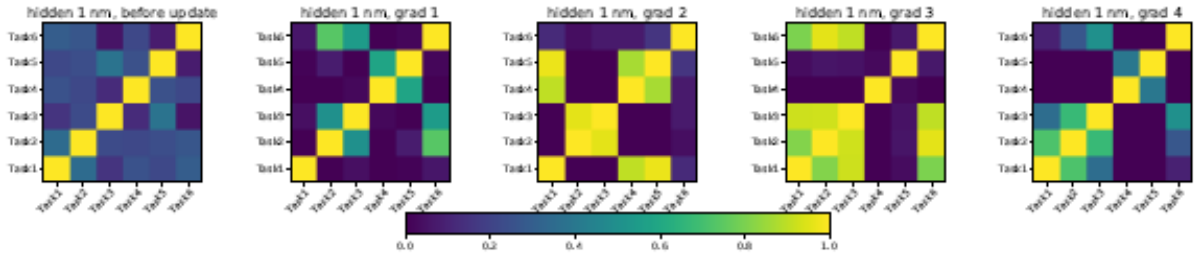
representations required to solve each task are distinct from one another. Therefore, adaptation by fine-tuning the representations of a base network via few gradient steps of parameters update would require a significant jump in the solution space. Standard policy network struggles to enable such jumps in the solution space. However, by incorporating neuromodulators that dynamically alter the representations, such jumps becomes possible.

Representation similarities of the neuromodulatory units across tasks

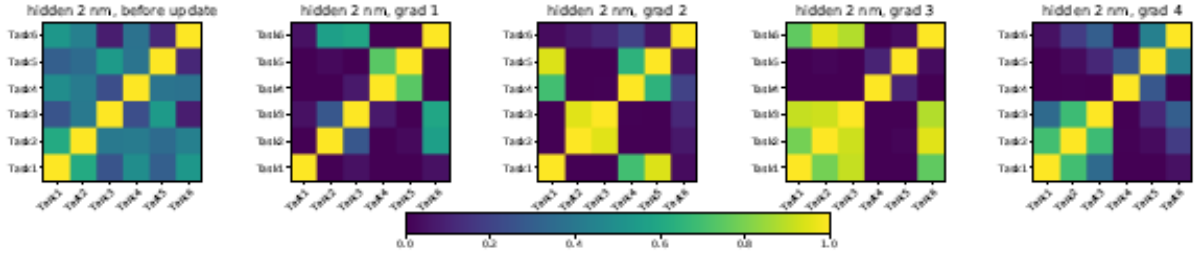
We next analyzed where the representational diversity (dissimilarity in representations across tasks) comes from, and if the neuromodulatory layer effectively contributes to rich representations as Figure 117(b) appears to suggest. The analysis we present here shows task representation similarities measured more specifically across the neuromodulatory layers of the proposed architecture. From Figure 117(b), it appears that such dissimilarity is enhanced by the neuromodulatory activities in the NPN. Again CKA was employed to compare the neuromodulatory activities per layer across different tasks. Figures 118, 119, and 120 present the heatmap plots for the 2D navigation, CT-graph depth 2 and ML45 environments. The non-uniformity in the heatmap plots, in contrast to those of Figure 117(a), indicates that those layers encode diverse or dissimilar representations across tasks. We can therefore conclude that the neuromodulatory activities, when projected onto a layer's standard neurons, produce the desired dissimilar representations across tasks.

Control Experiments: Larger SPN, equaling the number of parameters of the NPN

Since the inclusion of neuromodulators increases the number of parameters in a neuromodulated policy network, a set of control experiments were conducted in which the number of parameters in a standard policy network was configured to approximately match that of a neuromodulated

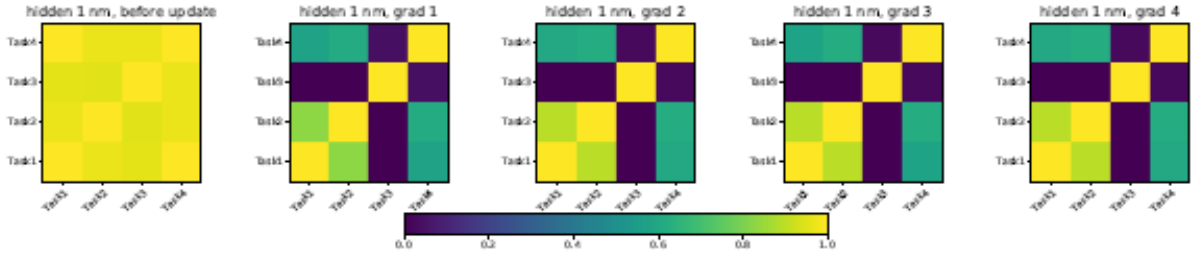


(a) first hidden layer.

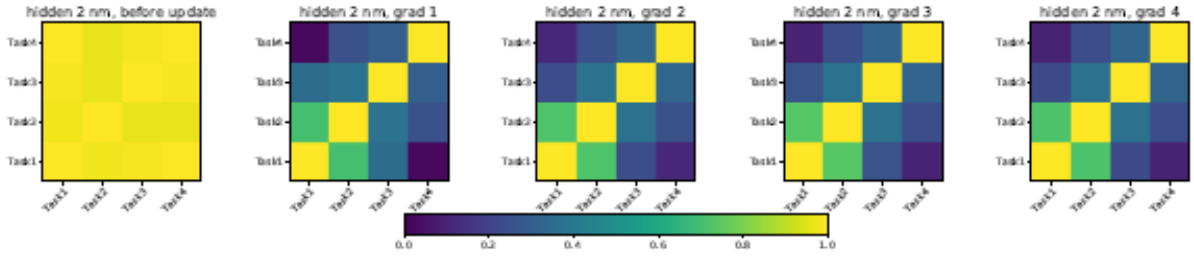


(b) second hidden layer.

Figure 118: Representation similarities of neuromodulatory activities h^m between tasks in the 2D navigation environment.

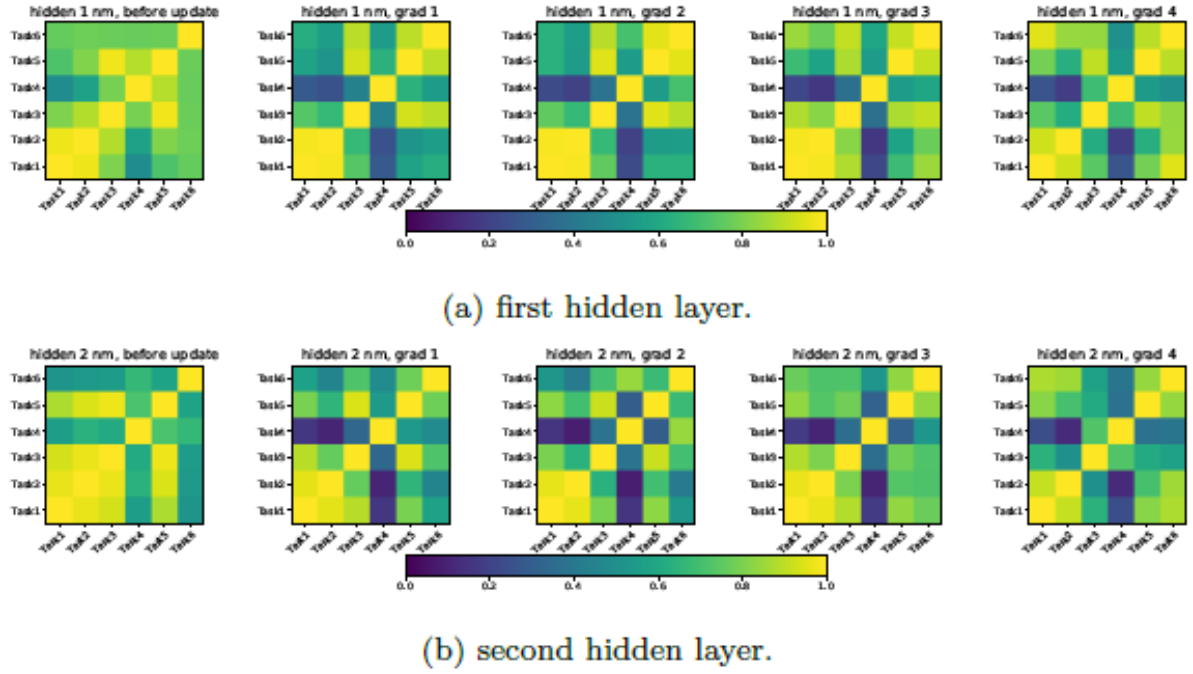


(a) first hidden layer.



(b) second hidden layer.

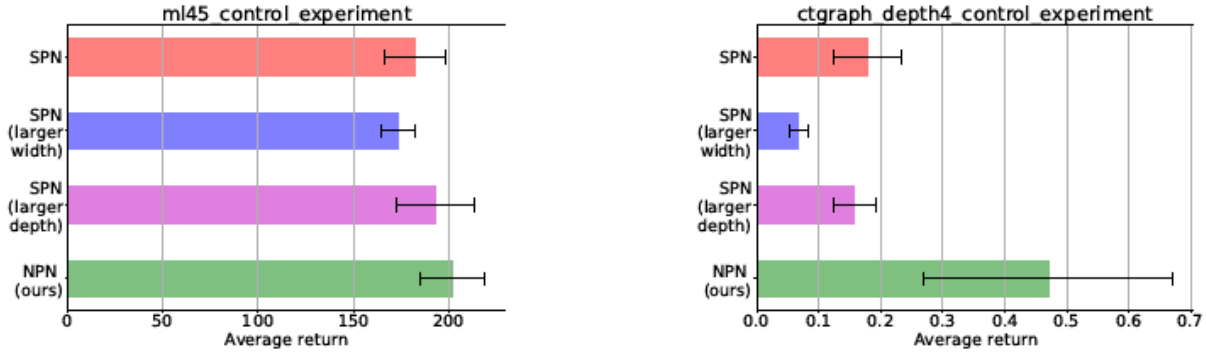
Figure 119: Representation similarities of neuromodulatory activities h^m between tasks in the CT-graph depth2 environment.



(a) first hidden layer.

(b) second hidden layer.

Figure 120: Representation similarities of neuromodulatory activities h^m between tasks in the Meta-World ML45 environment.



(a) ML45

(b) CT-graph depth4

Figure 121: Control experiments: Adaptation performance of NPN and variants of SPN.

policy network. This was achieved by increasing the size of each hidden layer of the standard policy network (called SPN larger width) in one experiment, and increasing the depth or number of hidden layers by 1, i.e., an additional layer (called SPN larger depth) in another experiment. Using CAVIA, experiments were conducted in the CT-graph depth4 and the ML45 Meta-World environments, comparing the standard policy network (i.e., the original size), its larger variants and a neuromodulated policy network. The results are presented in Figure 121. We observe from the results that the increase in the size of the policy network does not lead to matching the performance of the neuromodulated policy network.

1.5 Conclusions

1.5.1 MOdulated Hebbian Network

We addressed confounding POMDPs using a new neural architecture (MOHQA) for deep RL. The key novelty in MOHQA is the addition of a modulated Hebbian learning network with neural eligibility traces (MOHN) to a standard DQN architecture. The objective is to provide basic RL algorithms with the ability to ignore confounding observations and delays, and associate key cause-effect relationships to delayed rewards. It was shown that the combination of DQN and MOHN can match and even outperform competing deep RL algorithms such as A2C, AMRL, and QRDQN+LSTM on confounding POMDPs. While this is the first proof-of-concept study to propose a combined Hebbian and backpropagation-learned architecture for deep RL, the promising results encourage further tests on a wider range of standard deep RL benchmarks.

1.5.2 Sliced Cramer Preservation

We introduced a new generic approach towards selective synaptic plasticity for preserving the distribution of the network's output, at an arbitrary layer, for previously learned tasks. We started from a memory replay-based regularization that penalized the change in the distribution of the network's output and showed that a second-order Taylor expansion of such regularization would lead to a selective synaptic plasticity approach that does not require memory of samples from previously seen tasks. Furthermore, we proposed the sliced Cramér distance as a suitable metric for preserving these distributions, which leads to a straightforward algorithm for selective plasticity. Also, using a similar approach, we reverse-engineered the MAS framework and provided a geometrically meaningful regularization that leads to this algorithm. We then compared the online-EWC, MAS, and SCP methods on a variety of learning tasks, including supervised and unsupervised/self-supervised learning, and consistently showed competitive performance.

1.5.3 Neuromodulated Attention

We introduced a model of ACh and NE neuromodulation to perform goal-driven perception. The proposed network architecture discovers goals using online learning, and highlights the stimulus features corresponding to the goal. Moreover, the proposed system rapidly adapts when goal contingencies change. This neurobiologically inspired model can be applied to other problem domains and other top-down attentional networks.

Handling New Goals

In the present work, the goal classes were known, and the system guessed the appropriate goal given a goal identity and goal validity. However, the system might need to adapt to new goals or new goal classes. Adding multiple heads to the output layer of the network is one way this could be handled. This would not require retraining the stimuli (e.g., digits or real objects), but some additional training for the new goal classes. However, the architecture might be more scalable with a single head that learns the goals online without any *a priori* assumptions. Similar to the present model, these unknown goals would initially be guessed. After sufficient reward feedback, the model would associate different goals with different reward likelihoods. The introduction of the ACh/NE

neuromodulation should make the goal search fast and flexible. This will be explored in future iterations of our work.

Different Attentional Mechanisms
The choice of c-EB for a top-down attentional mechanism was motivated by its similarity to ACh system and its effect on top-down attention. However, as mentioned above, we believe that our system could also work with other SotA attentional mechanisms, including the CAM [196] and its more general variation Grad-CAM [197]. As long as the neural network structure can support an additional neuromodulation layer, and there is some means to flow goal information from the top to lower layers, our neuromodulatory goal-driven perception system should be compatible.

ML Applications
We had shown the compatibility of the adapted c-EB attention mechanism with the Microsoft COCO dataset and an indoor scenario. Our model is applicable in broader ML scenarios. If a system (e.g., a self-driving car, a HSR) faces many known and unknown task structures, our neuromodulatory goal-driven architecture should help it to choose tasks wisely regarding seen/unseen goals in a complex scenario.

Inspiration for Cognitive Neuroscience
Our experimental design could be replicated in biological studies with non-human primates or rodents to investigate relevant neuromodulatory signals in the brain. We predict that NE neurons would increase phasic activity after a goal switch. [222] have shown that the locus coeruleus/norepinephrine system redirects attention from one object to another, and switches attention between networks. Attention is strongly modulated by acetylcholine through its projections to sensory cortex [223]. Cholinergic activation has been shown to increase goal-driven attention in V1 by increasing the firing rate of neurons coding the attended objects [224, 225]. It would be of interest to test whether ACh activity to V1 becomes somewhat random after phasic NE responses and if ACh modulation varies depending on goal validity.

The robot experiments highlight a somewhat unexplored aspect of attention. In addition to feature or spatial attention, attention is deployed to intended actions (for a review, see [226]). Recent results suggest that attention is required for both action planning and movement outcome monitoring [227]. In our robot experiments, an intended action led to attention to an object associated with the desired action. Such an attentional network could have benefits for human-robot interaction, especially when the intended actions can change due to context.

1.5.4 Self-Preserving World Model

We have shown that pseudo-rehearsal based methods are capable of supporting continual learning within the World Model framework. Our main focus was to illustrate that simulated rollouts generated from the internal model can be used to preserve its own temporal prediction across random task exposures with no explicit task labels. The utility of this internal model, however, is generally measured in terms of its ability to inform a controlling agent. Continual learning results when using the learned World Model as input to modern policy gradient methods seem mixed.

What has led to these mixed results in the controlling network? One possible explanation is that the World Model architecture explored here and in [67] is not well suited for discreet action spaces. [228] suggest methods for modifying the loss function of the internal model for discreet actions, while [160] learn an action embedding. Similarly, [67] used evolutionary optimization on the controlling network as opposed to gradient descent. Finally, in contrast to [67], we did no explorations on temperature variation in the M network for improving agent performance. In principle these methods could be adopted in the current work to improve performance.

We assumed a latent representation that is capable of encoding and decoding the features of new tasks. There exists some preliminary work investigating continual learning in a VAE framework for classification [229]. Their approach essentially builds in a prior to the loss function that preserves the already existing latent space, which in principle would work in the current architecture. Integrating this approach with the World Model framework, however, is left for future studies.

Our main goal in this work was to explore the potential for continual learning within the World Model architecture. We have shown that pseudo-rehearsal is sufficient for continual learning within the internal model, or M network, and that there exists methods for potentially improving the controlling agent's performance. Future work should focus on exploring a combined V and M framework that controls the latent space during continual learning, and allows for both discreet and continuous action embeddings.

1.5.5 Context-Skill Model

We have addressed a major challenge in deploying artificial agents in the real world related to how they can only perform well in situations for which they were trained. Our work demonstrates a potential solution based on separating contexts from the actual skills. Context can then be used to modulate the actions in a systematic manner, significantly extending the unseen situations that can be handled. This principle was successfully evaluated in three domains: challenging versions of the Flappy Bird and LL games as well as the CARLA autonomous driving simulation. The results suggest that the Context-Skill approach should be useful in many control and decision making tasks in the real world. In these experiments, however, the neural networks have a fixed t opology. It may be possible to customize their architecture further through evolution [230, 231], and thereby delineate and optimize their roles further. Besides the architecture, the choice of training tasks plays an important role; methods that automatically design a curriculum, i.e., a sequence of new training tasks [232, 233, 234, 235, 236], could lead to further improvements. Third, instead of using handcrafted features, convolutional layers added in front of the Context and Skill modules could be used to discover features while training, extending the approach to more general visual tasks.

Lifelong ML tries to mimic how humans and animals learn by accumulating the knowledge gained from past experience and using it to adapt to new situations incrementally [237]. The generalization ability of the Context-Skill approach can serve as a foundation for continual learning. It provides an initial rapid adaptation to new situations upon which further learning can be based. How to convert generalization into a permanent ability in this manner is an interesting direction of future research.

1.5.6 Reflexive Adaptation

We have presented the first steps taken towards devising a module that can allow an autonomous agent to develop and use reflexes. Analogous to this ability in humans and animals, reflexes in an autonomous agent is a quick, short sequence of actions that can allow it to circumvent a sudden, surprise, emergency situation that could lead to a catastrophic event, such as damage or destruction of the agent and/or its surroundings. This is particularly important when an agent is introduced to a new environment and/or is learning a new task. Reflexive adaptation, therefore, promotes safe exploration. When an agent is introduced to a new task one can expect a transition period wherein task performance drops significantly while the agent explores and gradually learns how to function appropriately. However, for applications such as autonomous driving, where such performance drops could lead to dangerous actions that result in damage or death, it is vital to have some type of mechanism in place within the overall system to ensure all exploration is conducted safely. This necessitates a reflex module that continuously monitors the agent's actions during this critical time.

Three components that comprise the overall reflex module were investigated and developed: Emergency Prediction, Reflex Retrieval, and Reflex Generation. While progress was made on the first two components, the preliminary investigations revealed that further research is required to develop a fully working solution for them that could be seamlessly integrated to form a complete reflex module. Most of the focus was placed on devising a solution for the third and more important component, reflex generation, which represents the method and algorithm by which the actual reflex action-sequence to be implemented can be devised online. For this, the BO method was presented.

Through the simulation experiments conducted with the OpenAI Gym Atari game environments, the BO approach to reflex generation has shown good potential for devising effective reflex action-sequences, online, to provide a customized response to emergency situations. However, more research is required to develop a reliable and effective means of predicting emergency situations over short-term time horizons. While the predictive model based on the World Model's approach in [171] provided a good starting point into gauging the feasibility and potential for such a predictive ability in Atari games, a suitable approach will have to be found as we move into more complex environment such as the CARLA autonomous driving simulator. This will most likely require a revised look at the type of model to be used, but will nonetheless be a crucial endeavor in order to properly validate the reflex module, wherein a feasible option must be established to replace the assumption of an ideal predictor. This would make the reflex module a much more substantial contribution to overall L2 systems. Another avenue of research will be to explore further and definitively establish the role of the MTME algorithm and the archive of priors when learning new tasks. While the MTME algorithm has been shown to effectively be able to create such an archive, questions still remain on how it can be updated and expanded as new experiences are encountered by the agent.

Application of Reflexes to Autonomous Driving

In the context of autonomous driving, one way in which reflexes would be useful is in the safe adoption of new policies obtained from software updates. It can be expected that new updates would come from the consolidation of data obtained from various new learning experiences and/or from modifications to an agent's model. The resulting software updates would have been tested on

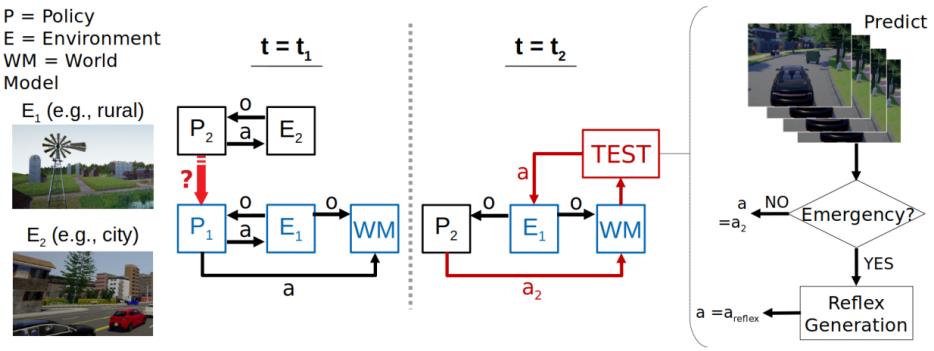


Figure 122: Illustration of how a new policy can be safely adopted using reflexes.

environments representative of those in which most cars will be used, but not necessarily exactly the same.

The actual environment in which any given car must function will have some differences from these representative environments that have been used for training and testing. It is therefore possible that the nuances of a particular environment can still make new software policies unsafe in certain situations. Ensuring that implementing new policies output by updated software do not result in dangerous actions can be accomplished by having a reflex module monitor the actions of the new policy to determine if they are likely to lead to an emergency situation, and takeover with reflexive actions when necessary.

Figure 122 illustrates how the reflex module can be used in this context. Consider a given autonomous vehicle that has been deployed in a rural environment and has been functioning there for some time under some policy P_1. At some point in time, t_1, a software update may be sent with a new policy P_2 that must replace P_1. The new policy may have been trained and tested in a city environment, for example, and may sometimes have difficulty in properly perceiving and computing appropriate actions in certain scenarios within the rural environment.

In order to adopt the new policy safely, then, at any subsequent time t_2, the actions output under this new policy would first be tested by the World Model. Having functioned within this vehicle's particular environment for an extended period of time, one can expect that the World Model would have learned a good model for the dynamics of the vehicle and the result of its actions in this environment. Hence the World Model would be able to make sufficiently accurate predictions into the short-term future of the result of actions that policy P_2 takes.

The workflow would thus change to that shown on the right-hand-side of Figure 122, wherein actions output by P_2 are first provided as input to the World Model, which in turn performs a prediction of the outcome of those actions. If an emergency situation is predicted to occur under P_2, a reflex is generated to circumvent that situation. Otherwise, the actions of P_2 are used as-is.

In order to demonstrate the reflex module being used in this context to circumvent an emergency situation, a dynamic object crossing scenario, as illustrated in Figure 123, could be used. Here, an

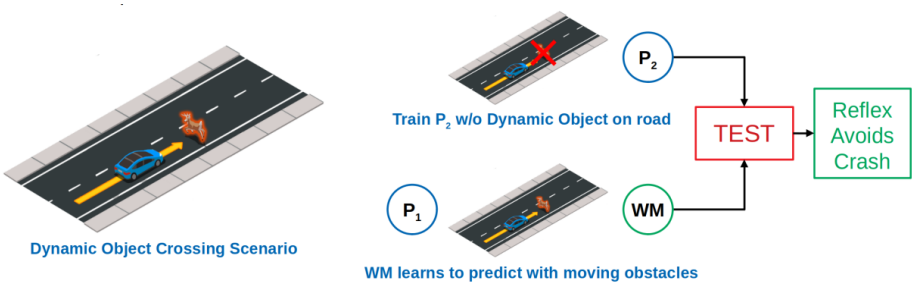


Figure 123: A possible simulated driving scenario for demonstrating reflexes in autonomous driving.

object moves unexpectedly into the path of the autonomous vehicle while it is using a new policy, P_2. We assume that, while the World Model is trained to detect an imminent collision under this scenario, the new policy P_2 is not trained to handle it and, thus, does not provide the necessary actions to avoid collision. Hence, an appropriate reflexive action-sequence is generated and the vehicle performs a maneuver to avoid collision, namely, some type of emergency braking or a brake and swerve combination.

1.5.7 Probabilistic Program Neurogenesis

We have demonstrated a new approach to neurogenesis based on probabilistic program learning. Our method helps to address the continual learning problem wherein learning occurs sequentially on different tasks. In this setting, when learning a new task, it is of utmost importance to avoid catastrophic forgetting of previously learned tasks. However, previously learned information that generalizes to the new task should be leveraged to limit growth in the complexity of the learning model.

Past work on neurogenesis has largely focused on employing ad hoc rules or leveraging local statistical properties of neurons and/or layers [89, 90] to decide the number and location of the new neurons. In contrast, our approach employs an efficient global search process using a probabilistic program, which has a higher probability of finding optimal or near-optimal solutions. Moreover, the probabilistic program process allows for easy integration of new features and rules to better regulate the addition of new neurons. In the future we will also validate our framework for continual learning using RL tasks (e.g., autonomous driving in the CARLA simulation environment) and more challenging datasets such as ImageNet [238], and make comparisons with additional methods such as evolution strategies. Furthermore, we have begun extending our approach to CNNs. In this application, instead of adding new neurons, PPL adds new feature maps to the convolutional layers.

1.5.8 Meta-Learned Instinct Network

The modular network approach we developed is a promising avenue for achieving safe online reinforcement learning. Without a computationally expensive outer meta-learning loop, IR^{2}L shows that an instinct network can be efficiently trained that can be transferred to multiple tasks and different policies than the ones from a pre-training phase. Although the results are encouraging, there is still work to be done in finding instincts with an optimal and predictable trade-off between safety and performance, as well as better guarantees of generality in task transfer. Future work involves

implementing the IR^{2}L architecture for self-driving to protect the car agents during cross-task adaptations.

1.5.9 Plastic Neuromodulated Network

The neuromodulatory gating mechanism that we introduced is reminiscent of the gating in recurrent/memory networks (LSTMs [156] and GRUs [239]). In this respect (with the observation of improved performance as a consequence of neuromodulatory gating in this work), the noteworthy performance demonstrated by meta-RL memory approaches [120, 119] could also be a consequence of such gating mechanisms. Nonetheless, the present study aims to highlight the advantage of a simpler form of gating (i.e., neuromodulatory gating) on a Multi-Layer Perceptron (MLP) feed-forward network, and thus could help to pinpoint the advantage of such dynamics in isolation. Furthermore, the advantage of our approach over gated recurrent variants is somewhat similar to the advantages derived from decoupling attention mechanism from recurrent models (where it was originally introduced) and applying it to MLP networks (i.e., Transformer models) [240]. By decoupling neuromodulatory (gating) mechanism from recurrent models and applying it to MLP models (as in our work), the advantages of faster training and better parallelization were achieved while maintaining the benefit of neuromodulatory g ating. Therefore, our proposed approach is faster to train and more parallelizable in comparison to memory variants, while maintaining the advantages that neuromodulatory gating offers. Memory based approaches will still be required for problems where memory is advantageous such as sequential data processing and POMDPs.

Task Similarity Measure and Robust Benchmarks. Increasing task complexity was presented in this work by moving from simple 2D point navigation environment to Half-Cheetah locomotion and then to the complex robotic arm setup of the Meta-World environment. Furthermore, exploiting the benefits of configurable parameters in the CT-graph environment, we were able to control the complexity in the environment. Overall, task complexity was viewed through the perspective of task similarity (i.e., environments with dissimilar task were viewed as more complex and vice versa). Despite these efforts, a precise measure of task complexity and similarity was not clearly outlined in this work and this is widely the case in meta-RL literatures. There is a need for the development of precise metrics for measuring task similarity and complexity in the field. The CT-graph with its configurable parameters allow for tasks to be mathematically defined, which is a first step towards alleviating this issue. However, a separate future research investigation would be necessary to develop explicit metrics that can be incorporated into meta-RL benchmarks.

We hypothesize that such a task similarity metric should be able to capture the precise change points in a task relative to other tasks. For example, a useful metric could be one that captures task change either as a function of change in reward, or state space, or transition function, or a combination of these factors. Most benchmarks in meta-RL have been focused on task change as reward function change. However, a more robust benchmark could include the aforementioned change points in order to further control the complexity. The CT-graph, Meta-world, and the recently developed Alchemy [241] environment are examples of benchmarks with early stage work in this direction, albeit implicitly. Therefore, the development of a precise measure of task similarity and complexity, as well as robust benchmarks with configurable change points (i.e., reward, state/input, and transition) would be highly beneficial to the meta-RL field.

Future Work. We introduced an architectural extension of the standard meta-RL policy networks to include a neuromodulatory mechanism, investigating the beneficial effect of neuromodulation when augmenting existing meta-RL frameworks (i.e., neuromodulation as complementary tool to meta-RL rather than competing). The aim is to implement richer dynamic representations and facilitate rapid task adaptation in increasingly complex problems. The effectiveness of the proposed approach was evaluated in meta-RL setting using CAVIA and PEARL algorithms. In the experimental setup across environments of increasing complexity, the neuromodulated policy network significantly outperformed the standard policy network in complex problems while showcasing comparable performance in simpler problems. The results highlight the usefulness of neuromodulators to enable fast adaptation via rich dynamic representations in meta-RL problems. The architectural extension, although simple, presents a general framework for extending meta-RL policy networks with neuromodulators that expand their ability to encode different policies. The projected neuromodulatory activity can be designed to perform other functions apart from the one introduced in this work e.g., gating plasticity of weights, or including different neuromodulators in the same layer. The neuromodulatory extension could also be tested with a recurrent meta-RL policy, with the goal of enhancing the memory dynamics of the policy. Our analysis indicates that this framework is most suited to problems that require rapid change in optimal representations across tasks, while its advantage is reduced when tasks can be solved using similar representations.

2. Phase 2: Lifelong Learning System Integration

2.1 Summary

The goal of Phase 2 was to further extend and integrate algorithms developed in Phase 1 for demonstrations of continual learning, transfer and adaptation, and sustainability in a domain specific simulation environment, as well as to perform ablation studies to determine the conditions under which each of the pertinent STELLAR components operates best for various aspects of L2.

Consistent with this goal, we focused on rigorous testing and evaluation alongside the integration of the innovative components developed in Phase 1 (Figure 124) into our complete L2 system (Figure 125). We also collaborated with Johns Hopkins University and Baylor College of Medicine teams in the L2M program to additionally integrate their SCP++ and Metaplasticity Kernel Model components, respectively, into our L2 system. In Figure 124, the various components are grouped according to shared functions. The last column lists the hypotheses for the system-level L2 metrics that would be most affected if the pertinent components were to be ablated. Note the components are highlighted in yellow within the system architecture shown in Figure 125.

We first developed a large-scale software architecture for the seamless integration of components, then completed their standalone implementations for the autonomous driving domain in the CARLA simulator, and gradually integrated them into a complete working system. We gradually demonstrated the efficacy of the fully integrated L2 system for the autonomous driving domain in the CARLA simulator with online adaptation to different weather conditions, different vehicle models (wheel asymmetries, vehicle dynamics), and different driving tasks (driving in the correct lane vs. opposite lane in regular traffic) across several quarterly evaluations (Month 9, Month 12, Month 15, and Month 18). Further, we participated in the final Month 21 evaluation in the common MiniGrid environment for continual RL. We also performed ablation studies in Month 18 and Month 21 evaluations to study various research hypotheses and questions. Across the evaluations the STELLAR system achieved significant performance improvements relative to conventional ML Single Task Experts (STEs), exceeding key program targets set by DARPA:

- about 25% improvement in performance,
- about 7.5x increase in learning efficiency, and
- about 3.5x improvement in forward transfer (leveraging prior learning to facilitate learning a new task)

2.2 Introduction

For the Month 9 evaluation, we integrated two components (namely, SCP and Self-Preserving World Model) and evaluated the performance of the resulting Month 9 system on a low-complexity L2 scenario in the CARLA simulator. Following the Month 9 evaluation, we integrated four more components resulting in the 6-component Pre-Month-12 system and compared its performance to the Month 9 system.

Component	Lifelong Learning Function	Primary Lifelong Learning Metric
Sliced Cramer Preservation	To overcome catastrophic forgetting of old tasks (selective plasticity)	Performance Maintenance
Metaplasticity Kernel Model		
Sliced Cramer Preservation ++		
Self-Preserving World Model	For backward transfer as well as forward transfer (generative replays, context representation)	Forward Transfer Backward Transfer
Context-Skill Model		
Neuromodulated Attention	For rapid performance recovery when an old task repeats (task-relevant features)	Performance Recovery
Modulated Hebbian Network	To rapidly adapt to new tasks (neuromodulation)	Relative Performance Sample Efficiency
Plastic Neuromodulated Network		
Reflexive Adaptation	To safely adapt to new tasks (reflexes, instincts)	Learn Burn
Meta-Learned Instinct Network		
Neurogenesis	To scale up the learning of new tasks (architectural changes)	Cumulative Gain

Figure 124: Summary of the innovative components developed in Phase 1.

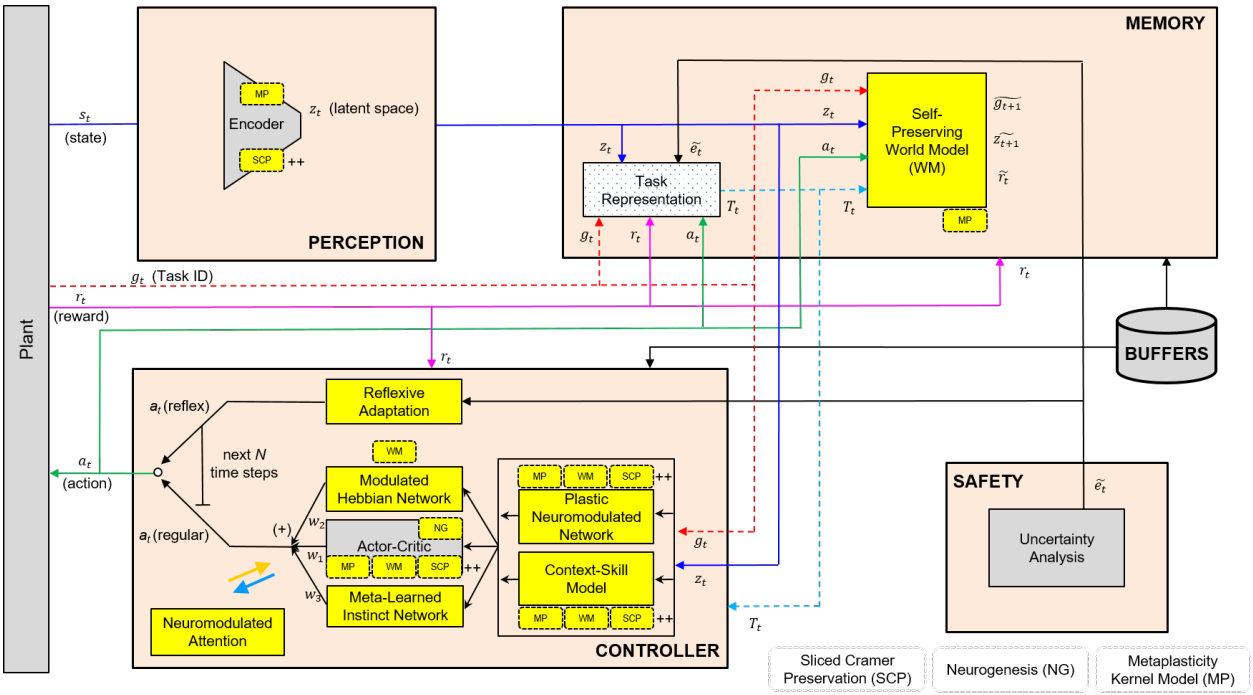


Figure 125: Static architecture of the fully integrated STELLAR system for L2 in autonomous systems.

For the Month 12 evaluation, we integrated a total of eight components and performed the initial demonstration of the resulting Month 12 system showing intermediate metrics for the three L2 conditions (namely, continual learning; transfer and adaptation; sustainability) on a medium-complexity L2 scenario in the CARLA simulator. We demonstrated that the Month 12 system performs better than the 2-component Month 9 system and the 6-component Pre-Month-12 system on the Month 9 evaluation protocol, as well as the 10-component Pre-Month-15 system performs better than the 8-component Month 12 system on the Month 12 evaluation protocol.

For the Month 15 evaluation, we demonstrated our fully integrated 11-component STELLAR system on the Month 15 evaluation protocol on a high complexity L2 scenario in the CARLA simulator. Since the Month 12 evaluation, we had integrated the three remaining L2 components; namely, Meta-Learned Instinct Network from IT University of Copenhagen, Metaplasticity Kernel Model from Baylor College of Medicine, and SCP++ from Johns Hopkins University (Figure 124). We also showed that the fully integrated Month 15 system performs better than the 8-component Month 12 system on the Month 12 evaluation protocol.

For the Month 18 evaluation, we documented the performance of our fully integrated STELLAR system on the Month 15 Condensed Scenario with different hyperparameters for Neuromodulated Attention, Plastic Neuromodulated Network, and Reflexive Adaptation. We not only showed L2, but also exceeded program targets set by DARPA for three of the five metrics; namely, Forward Transfer (by about 3.5x) to use expertise in a known task to facilitate learning a new task, Performance Relative to a Single Task Expert (by about 25%), and Sample Efficiency (by about 7.5x) to make better use of learning experiences than equivalent Single Task Expert. We also performed three ablation experiments to gain insight into the workings of the integrated system.

Finally for the Month 21 evaluation, we documented the performance of one particular configuration of our STELLAR system on the Month 21 evaluation protocol in the common MiniGrid environment. We also compared its performance to that of the Baseline Model, without any L2 components. For the MiniGrid domain, the L2M community decided to use encoded inputs as opposed to pixel RBG inputs. In this case, the encoded state space gives agent-centered partial view (7x7), with the identity (e.g., door), color (e.g., green), and status (e.g., door open) of objects encoded as integers. As a result, we noticed non-smooth changes in successive states, making the input space transitions fundamentally different than the visual CARLA domain (Figure 126).

This led to a subset of STELLAR components being not included for the Month 21 evaluation; namely, Neurogenesis (due to computational costs), Neuromodulated Attention (due to non-visual, encoded inputs), Reflexive Adaptation (due to non-visual, encoded inputs and the lack of the braking response), Meta-Learned Instinct Network (due to the lack of pre-deployment training), and MOdulated Hebbian Network (due to insufficient hyperparameter tuning). Further, we relied on explicit, experience replays than generative replays from the Self-Preserving World Model due to the difficulty in learning the non-smooth dynamics of the encoded inputs for the MiniGrid domain.

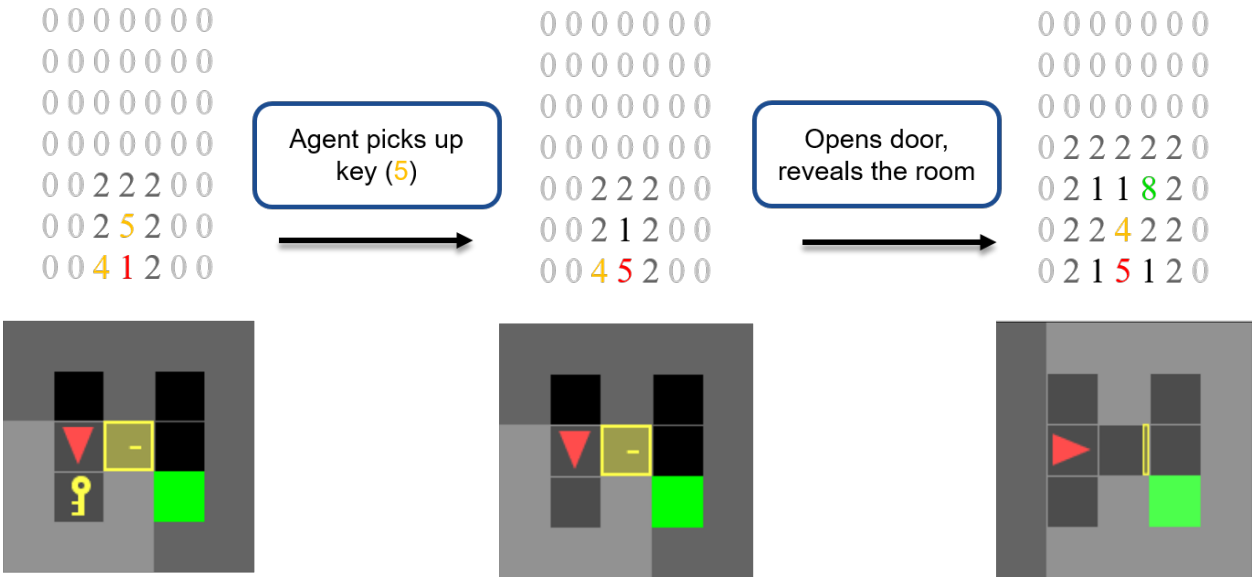


Figure 126: Non-smooth changes in successive states for the encoded version of the MiniGrid domain.

2.3 Methods, Assumptions, and Procedures

2.3.1 Month 9 Evaluation

Figures 128 and 129 show the system architectures of the Month 9 system with two L2 components integrated and the Pre-Month-12 system with six L2 components integrated (highlighted in yellow), respectively.

Tasks and Scenarios

For Month 9 evaluation, we investigated L2 for a sequence of five variants of a single task (low complexity scenario; Tables 13 and 14).

Single Task: Point-to-point navigation

Description:

Agent needs to drive safely from one point to another

What constitutes a single learning experience (LX)?

A time step every 50 ms. CARLA operates at 20 Hz. Agent gets instantaneous reward in each time step from which it can learn. Learning blocks are specified in number of time steps. An episode terminates when one of the following occurs: (1) destination is reached, (2) collision with another vehicle or a pedestrian, (3) maximum number of time steps per episode has elapsed (set to 600).

What constitutes a single evaluation experience (EX)?

An episode

What is the learning signal?

(+) reward for distance traveled towards the goal and increasing speed; (-) rewards for collisions, lane crossings, off-road driving

Table 13: Differences across the five task variants in the Month 9 Evaluation Protocol.

Task variant	Sun altitude angle	Precipitation	Left wheel radii	Right wheel radii
1A	90°	0	35	105
1B	61.25°	100	35	70
1C	32.5°	0	35	35
1D	3.75°	100	70	35
1E	-25°	0	105	35

Table 14: Common task elements across the five task variants in the Month 9 Evaluation Protocol.

Common task element	Value
Map	Town 01
Spawn point	Uniform distribution over 25 spawn points
Maximum episode duration	600 time steps
Number of vehicles	20
Number of pedestrians	0
Navigation difficulty	Straight ahead to destination
Sun azimuth angle	0°
Vehicle model	Audi TT

Similar to [242], instantaneous reward is a weighted sum of distance traveled towards the goal d in m, change in forward speed v in m/s, any collision c (0 or 1), intersection with the sidewalk s (between 0 and 1), and intersection with the opposite lane o (between 0 and 1):

$$r_t = (d_{t-1} - d_t) + 0.1(v_t - v_{t-1}) - 5(c_t) - 2(s_t - s_{t-1}) - 2(o_t - o_{t-1}) \tag{55}$$

The reward function (Equation 55) encourages the agent to correct any off-lane driving whenever it occurs. If the agent crosses into either the opposite lane or the sidewalk, it incurs a negative reward. But the negative reward can be eliminated in the cumulative reward for the episode with a later positive reward if it can return to its normal lane. Also, off-lane driving often results in collisions with other vehicles or objects leading to a lower cumulative reward for the episode. As noted above, any collision with a pedestrian or another vehicle terminates the episode immediately.

Application-specific metrics for task:
Cumulative reward per episode

Low Complexity Scenario: Single Task, Multiple Variants
Our scenario was of hard difficulty due to variations in both input space as well as state transitions. It turns out it is hard to incrementally learn a policy that works well for vehicles that lean to the right as well as those that lean to the left.

Characterize Task Relationships
We used a Baseline Model to quantitatively characterize relationships between all pairs of task variants in both scenarios. The Baseline Model was set to the L2 system with all the L2 components ablated (Figure 127. It comprises a Deep Convolutional Encoder for Perception and an Actor-Critic

Algorithm 2 Notional pseudo-code using transfer:

for each task variant T_i over task variants T_1 ... T_N:
  set Baseline Model to Blank state (common random initialization)
  evaluate Baseline Model on T_i
  train Baseline Model on T_i for 250K time steps (LXs)
  save agent state S_i

  for each T_j over task variants T_1 ... T_N (excluding T_i):
    restore agent state to S_i
    evaluate Baseline Model on task T_j
    assess Forward Transfer Contrast from T_i to T_j

Table 15: Task similarity matrix for the Month 9 evaluation.

	1A	**1B**	**1C**	**1D**	**1E**
1A	1	1	0.78	0.49	0.29
1B	0.7	1	0.76	0.44	0.22
1C	0.71	0.85	1	0.73	0.14
1D	0.33	0.63	0.85	1	0.36
1E	-0.28	0.39	0.73	1	1

Algorithm 3 Notional pseudo-code

for each task variant T_i over task variants T_1 ... T_N:
  set STE/Baseline Model to Blank state (common random initialization)
  train STE/Baseline Model on T_i for 500K time steps (LXs)

Network for Controller and trains using the PPO algorithm. It provides metrics to contrast with when integrating/ablating various components. We used the Forward Transfer Contrast metric as a proxy for task relationships, whereby larger values in the range -1 to 1 correspond to more similarity between the task variants. Table 15 provides the task similarity matrix for the Month 9 evaluation, where each row shows the Forward Transfer Contrast for all task variants when trained on a given task variant.

Pre-deployment Training of the L2 Agent
For Month 9 evaluation, we did not use any pre-deployment training. We directly transitioned the L2 agent from the Blank state to the Deployed state.

Train Single-Task Experts (STEs)
We used the same architecture as the Baseline Model for the STEs, and thereby characterized the ability of the STEs to learn each task.

Learning during Deployment for a Single Scenario
We ran the agent through the given scenario, with multiple runs for statistical reliability.

Considerations for Learning during Deployment
Averaging Over Multiple Lifetimes

Algorithm 4 Notional pseudo-code

```
randomly generate K task variant sequences for the scenario
for each run i of K runs:
    obtain corresponding task variant sequence
    set L2 agent to Ready-to-deploy/Blank state (common random initialization)
    for each learning block in the sequence:
        train L2 agent on the pertinent task variant
    for each evaluation block in the sequence:
        evaluate L2 agent on all task variants in the scenario
```

A single lifetime introduces several confounders into the resulting L2 metrics (e.g., specific tasks, specific task orderings, stochasticity in environment and agent learning). In order to factor out these confounders, the L2 metrics were averaged over multiple runs or lifetimes for a given scenario. We planned to run as many lifetimes as we could within the evaluation period to increase the statistical power for reliable estimation of the L2 metrics. In the end, we were able to collect a total of 30 lifetimes for each of three model configurations; namely, (1) Baseline Model, (2) Month 9 system (integrated system with two L2 components), and (3) Pre-Month-12 system (integrated system with six L2 components). The task variant sequences for the 30 lifetimes were matched across the three model configurations to have more statistical power for pairwise comparisons.

Generating Multiple Task Variant Sequences for a Scenario
As noted above, we generated multiple sequences of task variants for multiple lifetimes for a given learning scenario. In particular, our "Task Sequence Generator" generated a sequence of learning blocks and associated number of LXs for each block. Our procedure randomly sampled from different permutations of task order without replacement. There was at most one lifetime for each unique permutation of task order. As noted above, each lifetime comprised two cycles of exposures to task variants, with random order in each cycle.

Task Labeling and Feedback
Task Labeling
For Month 9 evaluation, we did not provide any information about the different variants of a given task to the L2 agent.

Task Feedback
For each LX, the L2 agent receives a reward signal that is organically available in the environment. The reward function shown above employs information such as distance to the destination, speed, intersection with the sidewalk, intersection with the opposite lane, and collisions.

2.3.2 Month 12 Evaluation

Figure 130 shows the system architecture of the Pre-Month-15 system with 10 L2 components integrated (highlighted in green and yellow).

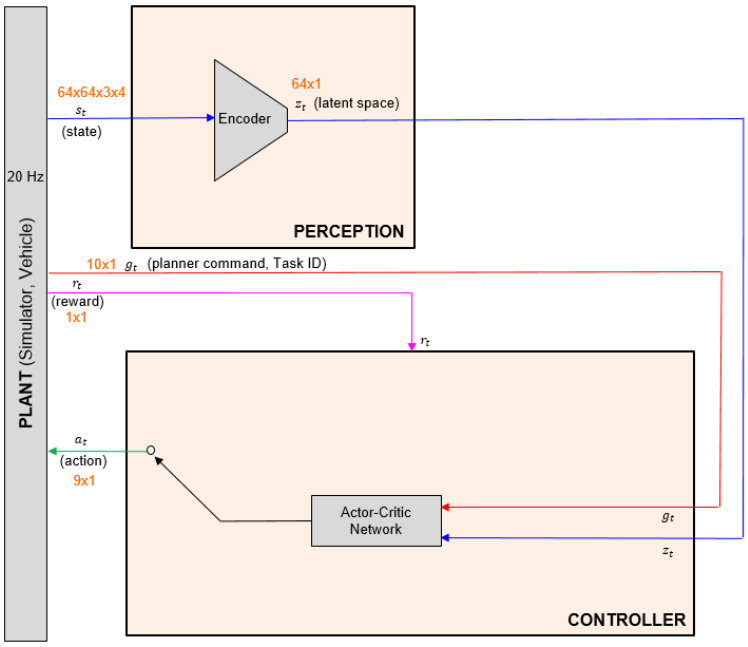


Figure 127: Architecture of the Baseline Model.

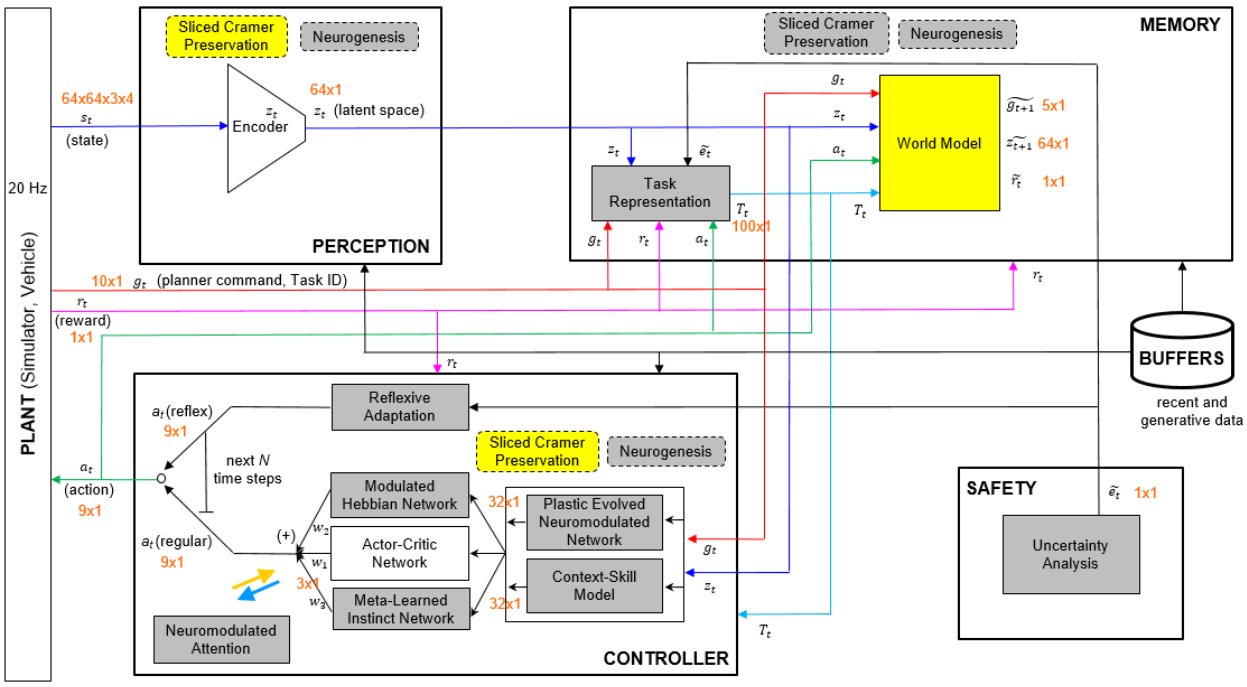


Figure 128: System architecture of Month 9 system with two L2 components (highlighted in yellow).

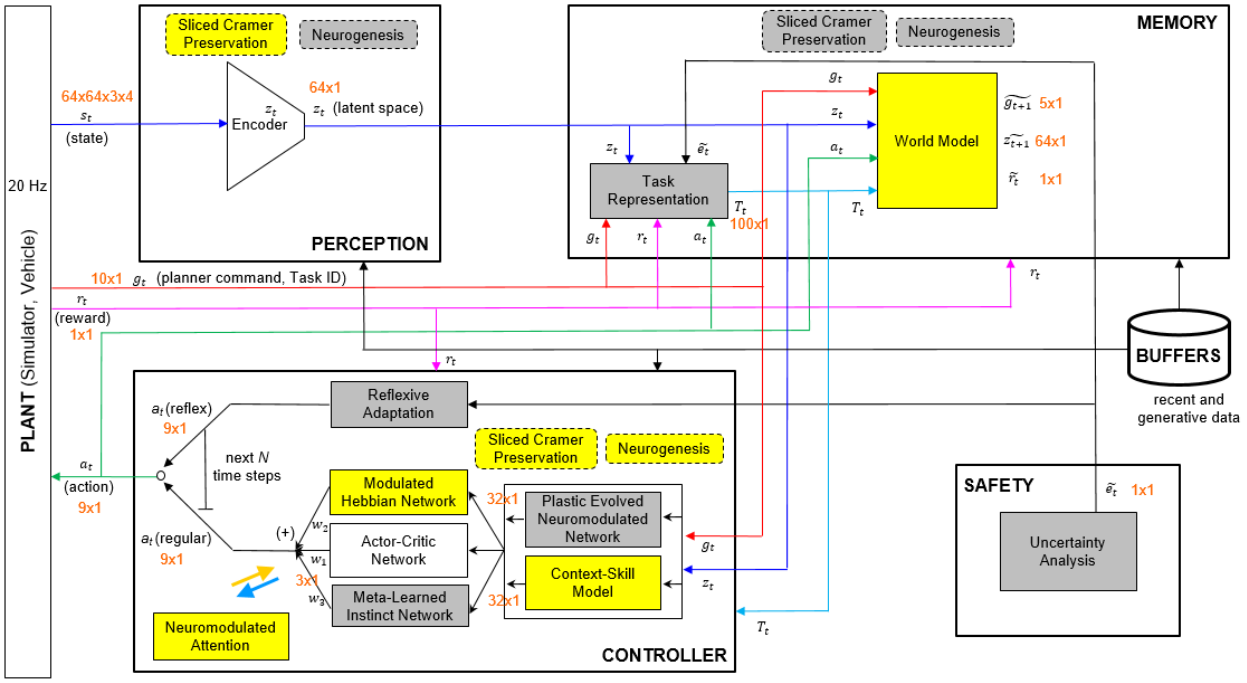


Figure 129: System architecture of Pre-Month-12 system with six L2 components (highlighted in yellow).

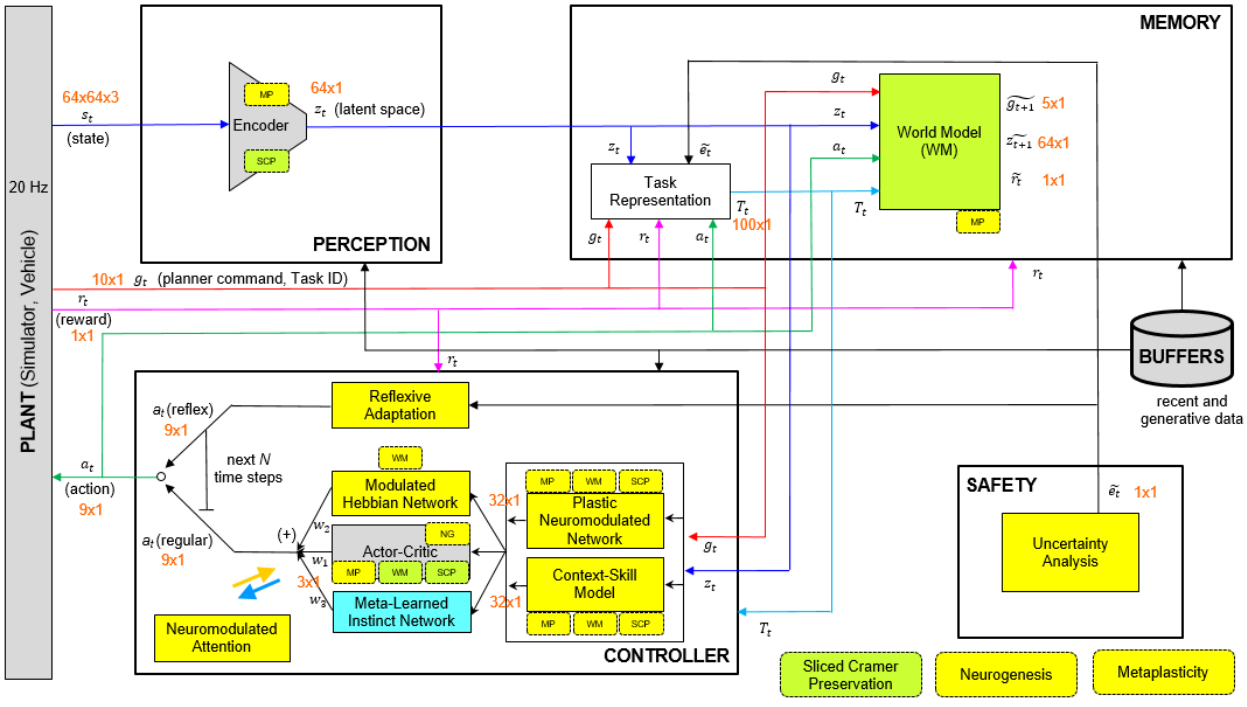


Figure 130: System architecture of Pre-Month-15 system with 10 L2 components (highlighted in green and yellow).

Tasks and Scenarios
For Month 12 evaluation, we investigated L2 for intermediate complexity scenarios involving up to four tasks that are clustered into two dissimilar pairs of similar tasks.

Single Variant of Task 1
Informal Name
Drive car within-lane

Programmatic Name
T1_1

Description
The car agent needs to drive from one point to another within the correct lane

Extrinsic parameters

- Maximum episode duration: 600 time steps
- Number of non-player vehicles: 20
- Sun azimuth angle: 90°
- Vehicle model: Audi TT
- Cloudiness (0%), Precipitation (0%)

Intrinsic parameters

- CARLA Town: 1
- Uniform distribution over 25 pairs of start and destination points
- Navigation difficulty: Drive straight ahead to destination
- Non-players will be spawned at random locations and have default behaviors (including following all traffic rules)

What constitutes a single LX?
A time step every 50 ms

What constitutes a single EX?
One episode

What is the learning signal?
Positive reward for distance traveled towards the destination, increasing speed; Negative rewards for decreasing speed, collisions, opposite-lane driving, off-road driving

Application-specific metrics
Cumulative reward per episode

Single Variant of Task 2
Informal Name
Drive motorcycle within-lane

Programmatic Name
T2_1

Description
The motorcycle agent needs to drive from one point to another within the correct lane

Extrinsic parameters

- Maximum episode duration: 600 time steps
- Number of non-player vehicles: 20
- Sun azimuth angle: 90°
- Vehicle model: Kawasaki Ninja
- Camera yaw = 45°
- Cloudiness (0%), Precipitation (0%)

Intrinsic parameters

- CARLA Town: 1
- Uniform distribution over 25 pairs of start and destination points
- Navigation difficulty: Drive straight ahead to destination
- Non-players will be spawned at random locations and have default behaviors (including following all traffic rules)

What constitutes a single LX?
A time step every 50 ms

What constitutes a single EX?
One episode

What is the learning signal?
Positive reward for distance traveled towards the destination, increasing speed; Negative rewards for decreasing speed, collisions, opposite-lane driving, off-road driving

Application-specific metrics
Cumulative reward per episode

Single Variant of Task 3
Informal Name
Drive car opposing-lane

Programmatic Name
T3_1

Description
The car agent needs to drive from one point to another within the opposite lane

Extrinsic parameters

- Maximum episode duration: 600 time steps
- Number of non-player vehicles: 20
- Sun azimuth angle: 90°
- Vehicle model: Audi TT
- Cloudiness (80%), Precipitation (75%)

Intrinsic parameters

- CARLA Town: 1
- Uniform distribution over 25 pairs of start and destination points
- Navigation difficulty: Drive straight ahead to destination
- Non-players will be spawned at random locations and have default behaviors (including following all traffic rules)

What constitutes a single LX?
A time step every 50 ms

What constitutes a single EX?
One episode

What is the learning signal?
Positive reward for distance traveled towards the destination, increasing speed, opposite-lane driving, off-road driving; Negative rewards for decreasing speed, collisions, within-lane driving

Application-specific metrics
Cumulative reward per episode

Single Variant of Task 4
Informal Name
Drive motorcycle within-lane

Programmatic Name
T4-1

Description
The motorcycle agent needs to drive from one point to another within the opposite lane

Extrinsic parameters

- Maximum episode duration: 600 time steps
- Number of non-player vehicles: 20
- Sun azimuth angle: 90°
- Vehicle model: Kawasaki Ninja
- Camera yaw = 45°
- Cloudiness (80%), Precipitation (75%)

Intrinsic parameters

- CARLA Town: 1
- Uniform distribution over 25 pairs of start and destination points
- Navigation difficulty: Drive straight ahead to destination
- Non-players will be spawned at random locations and have default behaviors (including following all traffic rules)

What constitutes a single LX?
A time step every 50 ms

What constitutes a single EX?
One episode

What is the learning signal?
Positive reward for distance traveled towards the destination, increasing speed, opposite-lane driving, off-road driving; Negative rewards for decreasing speed, collisions, within-lane driving

Application-specific metrics
Cumulative reward per episode

The agent got instantaneous reward in each time step from which it could learn. Learning blocks were specified in number of time steps. An episode terminated when one of the following occurred: (1) destination was reached, (2) maximum number of time steps per episode had elapsed, (3) any collision. As noted above, we employed two vehicle models (Audi TT, Kawasaki Ninja) with built-in differences in physical parameters such as for the body (e.g., mass, drag coefficient) and wheels (e.g., friction, damping rate, maximum steering angle, radius). The vehicle models also differed in camera orientation (0°yaw for car vs. 45°yaw for motorcycle). Non-player vehicles were spawned at random locations and exhibited default behaviors; namely, follow traffic rules, follow current lane, make random choices at intersections).

Similarity between tasks in same cluster (Cluster 1: T1_1, T2_1; Cluster 2: T3_1, T4_1) was ensured by using a common reward function combined with differences in physical parameters and camera orientation. And dissimilarity between tasks in different clusters was ensured by using different

reward functions. For the Month 12 evaluation, we relied on differences in input space rather an explicit Task ID signal for the agent to learn and exhibit different task behaviors (i.e., within-lane vs. opposite-lane driving).

The difficulty level of our scenario was hard due to the challenge of learning to switch from a task of one cluster to a task of another cluster (especially from within-lane driving to opposite-lane driving). We found that the differences in input space were not adequate enough for the necessary exploration as the vehicle is always spawned in the right lane.

Train Single-Task Experts (STEs)
We used the same architecture as the Month 12 STELLAR system to train the STEs, and thereby characterized the ability of the STEs to learn each task. We collected 10 STE runs per task, which were all initialized with the same "ready-to-deploy" state as the Month 12 system.

Performance threshold (P^{th}) was set to 0.99 of the terminal performance of the smoothed learning curve averaged across runs for each task, and the number of LXs required to reach it (LX^{th}) was determined accordingly.

In Month 12, we collected data for two scenario types of task sequences; namely, Permuted Tasks and Alternating Tasks.

Scenario 1 (Complexity=Intermediate, Scenario Type=Permuted Tasks)
LB = learning block, EB = evaluation block.
Label for scenario
"2-intermediate_permuted"

Structure of each run
EB LB_1 EB LB_2 EB LB_3 EB LB_4 EB
For a given permutation of tasks (T1_1, T2_1, T3_1, T4_1), LB_1 is for Task T1_1, LB_2 is for Task T2_1, etc. A given task only occurs once during each run. Each EB includes all four tasks.

Generation of task sequences across runs
Ensured uniform sampling across the six patterns (AABB, BBAA, etc.)

Selection of number of runs
Ran all 24 permutations, x2

Length of learning blocks within each run
LXs = 0.5 of LX^{th} for each task

Task labeling
No

Task feedback
A reward signal for each LX that is organically available in the environment

Scenario 2 (Complexity=Intermediate, Scenario Type=Alternating Tasks)

Label for scenario
"2-intermediate_alternating"

Structure of each run
EB LB_1 EB LB_2 EB LB_1 EB LB_2 EB LB_1 EB LB_2 EB
For a given pair of tasks (T1_1, T2_1) LB_1 is for Task T1_1, LB_2 is for Task T2_1, etc. Each task occurs three times in each run. Each EB includes the two tasks (T1_1, T2_1).

Generation of task sequences across runs
Ensured uniform sampling across the four patterns (AA, BB, AB, BA)

Selection of number of runs
Ran all 12 permutations, x4

Length of learning blocks within each run
LXs = 0.3 of LX^{th} for each task

Task labeling
No

Task feedback
A reward signal for each LX that is organically available in the environment

2.3.3 Month 15 Evaluation

Figure 131 shows the system architecture of the Month 15 system with all 11 components integrated (highlighted in green (Month 9), yellow (Month 12), and cyan (Month 15) by when they were integrated).

Tasks and Scenarios
For Month 15 evaluation, we investigated L2 for high complexity scenarios involving three tasks and two variants per task. Similar to Month 12 evaluation, we employed two vehicle models (Audi TT, Kawasaki Ninja) with built-in differences in physical parameters such as for the body (mass, drag coefficient, etc.) and wheels (friction, damping rate, maximum steering angle, radius, etc.). Please see below for details of the tasks.

Task 1 Variant 1
Informal Name
Drive car within-lane - clear midday

Programmatic Name
T1_1

Description
The car agent needs to drive from one point to another within the correct lane in clear midday conditions

Extrinsic parameters

- Maximum episode duration: 600 time steps

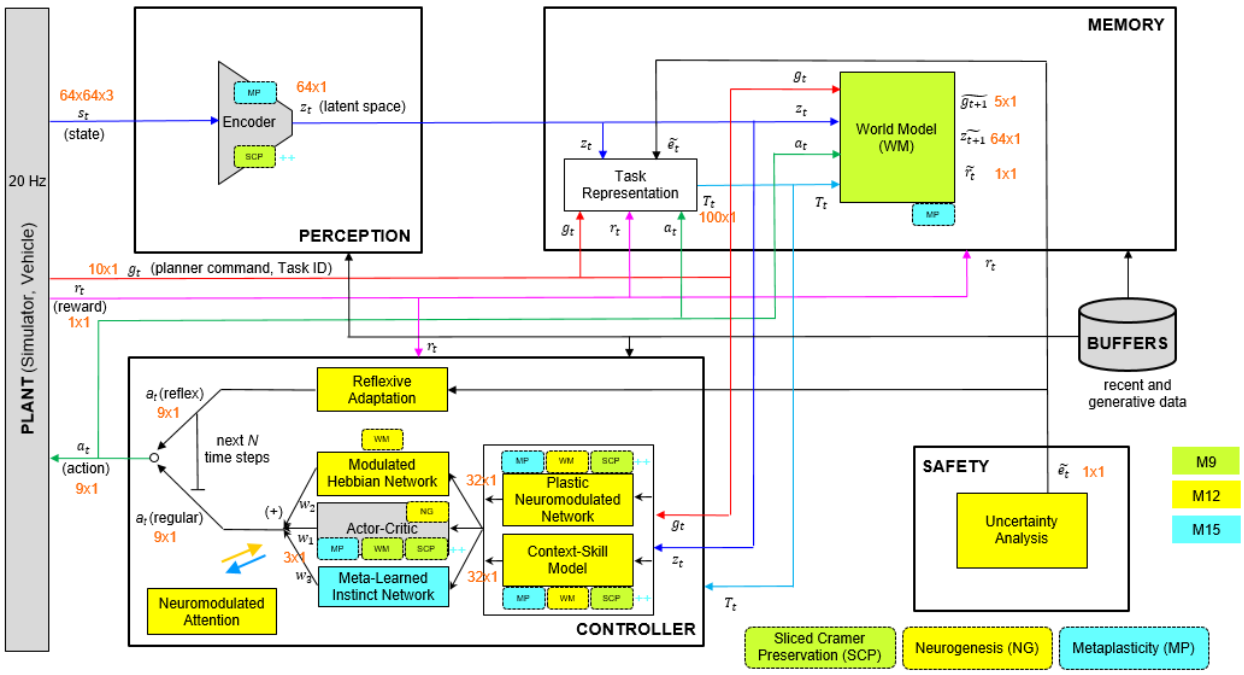


Figure 131: System architecture of the Month 15 system.

- Number of non-player vehicles: 20
- Number of non-player pedestrians: 0
- Vehicle model: Audi TT
- Precipitation (0%)
- Sun altitude angle: 90°

Intrinsic parameters

- CARLA Town 1
- Uniform distribution over 25 pairs of start and destination points
- Navigation difficulty: Drive straight ahead to destination
- Non-players will be spawned at random locations and have default behaviors (including following all traffic rules)

What constitutes a single LX?
A time step every 50 ms

What constitutes a single EX?
One episode

What is the learning signal?
Positive rewards for distance traveled towards the destination and increasing speed within the correct

lane; Negative rewards for distance traveled away from the destination and decreasing speed within the correct lane, and any collision

Application-specific metrics
Cumulative reward per episode

Task 1 Variant 2
Informal Name
Drive car within-lane - rainy midday

Programmatic Name
T1_2

Description
The car agent needs to drive from one point to another within the correct lane in rainy midday conditions

Extrinsic parameters

- Maximum episode duration: 600 time steps
- Number of non-player vehicles: 20
- Number of non-player pedestrians: 0
- Vehicle model: Audi TT
- Precipitation (100%)
- Sun altitude angle: 90°

Intrinsic parameters

- CARLA Town 1
- Uniform distribution over 25 pairs of start and destination points
- Navigation difficulty: Drive straight ahead to destination
- Non-players will be spawned at random locations and have default behaviors (including following all traffic rules)

What constitutes a single LX?
A time step every 50 ms

What constitutes a single EX?
One episode

What is the learning signal?
Positive reward for distance traveled towards the destination and increasing speed within the correct lane; Negative rewards for distance traveled away from the destination and decreasing speed within the correct lane, and any collision

Application-specific metrics
Cumulative reward per episode

Task 2 Variant 1
Informal Name
Drive motorcycle within-lane - clear dusk

Programmatic Name
T2_1

Description
The motorcycle agent needs to drive from one point to another within the correct lane in clear dusk conditions

Extrinsic parameters

- Maximum episode duration: 600 time steps
- Number of non-player vehicles: 20
- Number of non-player pedestrians: 0
- Vehicle model: Kawasaki Ninja
- Camera yaw: 45°
- Precipitation (0%)
- Sun altitude angle: 15°

Intrinsic parameters

- CARLA Town 1
- Uniform distribution over 25 pairs of start and destination points
- Navigation difficulty: Drive straight ahead to destination
- Non-players will be spawned at random locations and have default behaviors (including following all traffic rules)

What constitutes a single LX?
A time step every 50 ms

What constitutes a single EX?
One episode

What is the learning signal?
Positive rewards for distance traveled towards the destination and increasing speed within the correct lane; Negative rewards for distance traveled away from the destination and decreasing speed within the correct lane, and any collision

Application-specific metrics
Cumulative reward per episode

Task 2 Variant 2
Informal Name
Drive motorcycle within-lane - rainy dusk

Programmatic Name
T2_2

Description
The motorcycle agent needs to drive from one point to another within the correct lane in rainy dusk conditions

Extrinsic parameters

- Maximum episode duration: 600 time steps
- Number of non-player vehicles: 20
- Number of non-player pedestrians: 0
- Vehicle model: Kawasaki Ninja
- Camera yaw: 45°
- Precipitation (100%)
- Sun altitude angle: 15°

Intrinsic parameters

- CARLA Town 1
- Uniform distribution over 25 pairs of start and destination points
- Navigation difficulty: Drive straight ahead to destination
- Non-players will be spawned at random locations and have default behaviors (including following all traffic rules)

What constitutes a single LX?
A time step every 50 ms

What constitutes a single EX?
One episode

What is the learning signal?
Positive rewards for distance traveled towards the destination and increasing speed within the correct lane; Negative rewards for distance traveled away from the destination and decreasing speed within the correct lane, and any collision

Application-specific metrics
Cumulative reward per episode

Task 3 Variant 1
Informal Name
Drive motorcycle opposing-lane - clear night

Programmatic Name
T3_1

Description
The motorcycle agent needs to drive from one point to another within the opposite lane in clear night conditions

Extrinsic parameters

- Maximum episode duration: 600 time steps
- Number of non-player vehicles: 20
- Number of non-player pedestrians: 0
- Vehicle model: Kawasaki Ninja
- Camera yaw: 45°
- Precipitation (0%)
- Sun altitude angle: -60°

Intrinsic parameters

- CARLA Town 1
- Uniform distribution over 25 pairs of start and destination points
- Navigation difficulty: Drive straight ahead to destination
- Non-players will be spawned at random locations and have default behaviors (including following all traffic rules)

What constitutes a single LX?
A time step every 50 ms

What constitutes a single EX?
One episode

What is the learning signal?
Positive rewards for distance traveled towards the destination and increasing speed within the opposite lane; Negative rewards for distance traveled away from the destination and decreasing speed within the opposite lane, and any collision

Application-specific metrics
Cumulative reward per episode

Task 3 Variant 2
Informal Name
Drive motorcycle opposing-lane - rainy night

Programmatic Name
T3_2

Description
The motorcycle agent needs to drive from one point to another within the opposite lane in rainy night conditions

Extrinsic parameters

- Maximum episode duration: 600 time steps
- Number of non-player vehicles: 20
- Number of non-player pedestrians: 0
- Vehicle model: Kawasaki Ninja
- Camera yaw: 45°
- Precipitation (100%)
- Sun altitude angle: -60°

Intrinsic parameters

- CARLA Town 1
- Uniform distribution over 25 pairs of start and destination points
- Navigation difficulty: Drive straight ahead to destination
- Non-players will be spawned at random locations and have default behaviors (including following all traffic rules)

What constitutes a single LX?
A time step every 50 ms

What constitutes a single EX?
One episode

What is the learning signal?
Positive rewards for distance traveled towards the destination and increasing speed within the opposite lane; Negative rewards for distance traveled away from the destination and decreasing speed within the opposite lane, and any collision

Application-specific metrics
Cumulative reward per episode

In general, the tasks were focused on the agent driving safely from one point to another within a designated lane (either correct or opposite). The agent got instantaneous reward in each time step from which it could learn. There were positive rewards for distance traveled towards the destination and increasing speed within the designated lane, and there were negative rewards for decreasing speed within the designated lane and any collision. Learning blocks were specified in number of time steps. An episode terminated when one of the following occurred: (1) destination was reached, (2) maximum number of time steps per episode had elapsed, or (3) any collision. Non-player vehicles were spawned at random locations and exhibited default behaviors (follow traffic rules, follow current lane, make random choices at intersections). The application-specific metric was the cumulative reward per episode.

For Month 15, we relied on differences in the input space rather an explicit Task ID signal for the agent to learn and exhibit different task behaviors (e.g., correct-lane vs. opposite-lane driving). As for Month 12, the difficulty level of the scenarios was hard because of the challenge of learning to switch to opposite-lane driving from correct-lane driving. We again found that the differences in the input space were not adequate enough for the necessary exploration as the vehicle was always spawned in the right lane.

Pre-deployment training of the L2 agent
For Month 15 evaluation, we introduced a pre-deployment training phase for the Meta-Learned Instinct Network to be integrated into the STELLAR system. A pre-deployment agent was used that drove aggressively to increase the probability of collisions in order to train the Meta-Learned Instinct Network for potential interventions during deployment to avoid collisions. Note that the pre-deployment training was agnostic about tasks and task sequences during deployment.

Single-Task Experts (STEs) per task variant
We used the same architecture as the fully integrated STELLAR system to train the STEs to saturation, and thereby characterize the ability of the STEs to learn each task. We collected 10 STE runs per task, which were all initialized with the same "ready-to-deploy" state as the STELLAR system.

Performance threshold was set to 0.99 of the terminal performance of the smoothed learning curve averaged across runs for each task, and the number of LXs required to reach it (LX^{th}) was determined accordingly.

Month 15 Scenarios
In Month 15, we collected data for two scenario types of task sequences; namely, Condensed and Dispersed.

Scenario 1 (Complexity=High, Scenario Type=Condensed)
LB = learning block, EB = evaluation block.
Label for scenario

"3-high_condensed"

Structure of each run
EB LB_{A1} EB LB_{B1} EB LB_{C1} EB LB_{C2} EB LB_{B2} EB LB_{A2} EB
For a given permutation of task variants (e.g., A1, B1, C1, C2, B2, A2), where LB_{A1} is for task variant A1, etc., a given task variant occurs exactly once during each run. Each EB includes all six task variants.

Generation of task sequences across runs
Random permutation of the six task variants

Selection of number of runs
Collected data for 30 runs

Length of learning blocks within each run
LXs = 0.75 of LX^{th} for each task variant

Task labeling
No

Task feedback
A reward signal for each LX that is organically available in the environment

Scenario 2 (Complexity=High, Scenario Type=Dispersed)
Label for scenario
"3-high_dispersed"

Structure of each run
A superblock was generated for a permutation of task variants (e.g., A1, B1, C1, C2, B2, A2), with a short learning block (SLB), where SLB_{A1} is for task variant A1, etc. SLB indicates that this is a short learning block with length one third (1/S for S =3) of the Condensed LB size for that task variant. A given task variant occurred exactly once during each superblock. There was an EB after each SLB. Each EB included all six task variants. Three superblocks were generated, each using a random permutation of the six task variants.

Generation of task sequences across runs
Within each of the three "superblocks", task ordering was randomly permuted

Selection of number of runs
Collected data for 30 runs

Length of learning blocks within each run
LXs = 0.25 of LX^{th} for each task variant

Task labeling
No

Task feedback
A reward signal for each LX that is organically available in the environment

Table 16: Description of changes from Month 15 to Month 18 evaluation.

Hyperparameter Change from Month 15 to Month 18	Intended Impact
Reflexive Adaptation: Made the system more reflexive	Improve Performance Relative to a STE, Sample Efficiency
Plastic Neuromodulated Network: Increased layer size (to learn better implicit dynamic task representations)	

We additionally ran the Month 15 agent on the Month 12 Permuted and Alternating Scenarios to evaluate the progress of the STELLAR system.

2.3.4 Month 18 Evaluation

Here we documented the performance of our fully integrated STELLAR system for the three L2 conditions on the Month 15 Condensed Scenario with different hyperparameters for Reflexive Adaptation and Plastic Neuromodulated Network (Table 16). Reflexive Adaptation ensures that periods of significant performance drops leading to potential catastrophic events can still be handled safely. And Plastic Neuromodulated Network (PNN) learns and leverages task similarities for rapid adaptation. We also performed three ablation experiments to gain insight into the workings of the integrated system.

Experiment 1 (Ablation Study)
Research Question and Hypothesis
The lack of selective plasticity in the full system can be compensated by increasing the number of experience replays for performance maintenance

Predictions
If SCP++ is ablated, then Performance Maintenance and Backward Transfer will decrease. This can be compensated by increasing the number of experience replays.

For this experiment, we collected data for SCP++ ablation with three levels of experience replays (nominal, 5x nominal, and 10x nominal). Nominal replays consisted of 16 sequences of length 16 with 10 STELLAR updates for every preservation update. Our prediction was that both Performance Maintenance and Backward Transfer would increase across the three levels of replays. Fully-integrated STELLAR system served as the control/baseline condition.

Experiment 2 (Ablation Study)
Research Question and Hypothesis
Learning implicit dynamic task representations is critical for more transfer

Predictions
If PNN is ablated, then Forward Transfer, Backward Transfer, and Performance Relative to a STE will decrease.

For this experiment, we collected data for ablation of PNN. Our prediction was that Forward

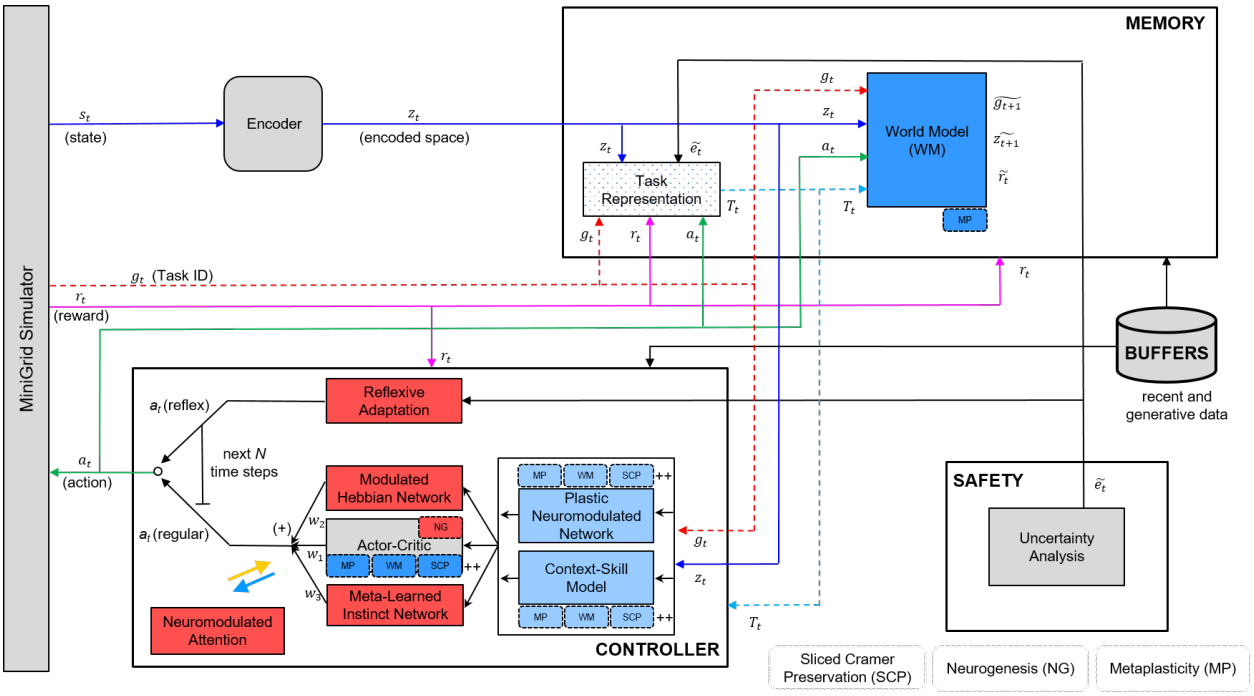


Figure 132: System architecture of the Month 21 system.

Transfer, Backward Transfer, and Performance Relative to a STE metrics would all decrease with the ablation. Fully-integrated STELLAR system served as the control/baseline condition.

Experiment 3 (Hyperparameter Variation)
Research Question and Hypothesis
Stronger preservation mechanisms reduce L2 in the Dispersed Scenario with task repetitions

Predictions
In Dispersed Scenarios, task performances especially in earlier learning blocks are not expected to be high. In this case, strong preservation of sub-optimal task representations would interfere with subsequent learning blocks. So, the hyperparameters that control the degree of preservation should be reduced to improve Backward Transfer, Performance Relative to a STE, and Sample Efficiency.

For this experiment, we collected data for the Month 15 Dispersed Scenario with SCP++ stiffness coefficient reduced to 10% of the nominal value. Our prediction was that Backward Transfer, Performance Relative to a STE, and Sample Efficiency metrics would all increase with the reduction in SCP++ stiffness. Fully-integrated STELLAR system served as the control/baseline condition.

2.3.5 Month 21 Evaluation

Figure 132 shows the system architecture of the STELLAR system used for the Month 21 evaluation, with included components highlighted in yellow and removed components highlighted in red..

Month 21 Tasks and Scenarios
For the Month 21 evaluation, each System Group used six tasks each with three variants. Each

variant had the same observation space and action space:

- Observation Space: (7, 7, 3) partial observation from agent perspective where each tile is encoded as (OBJECT_IDX, COLOR_IDX, STATE)
- Action Space: six discrete actions (left, right, forward, pickup, drop, toggle)

Task 1 Variant 1
Informal Name
Simple Crossing S9N1

Description
Agent navigates through a 9x9 maze with one crossing to reach the goal.

Extrinsic parameters
Grid size: 9x9
Number of crossings: 1

Intrinsic parameters
Crossing location
Opening location

What constitutes a single LX?
A single LX is one time step for a single agent with 324 maximum time steps.

What constitutes a single EX?
A single EX is one episode for a single agent with 324 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for reaching the goal.

Application-specific metrics
Reward of 1 minus penalty for number of steps taken.

Task 1 Variant 2
Informal Name
Simple Crossing S9N2

Description
Agent navigates through a 9x9 maze with two crossings to reach the goal.

Extrinsic parameters
Grid size: 9x9
Number of crossings: 2

Intrinsic parameters
Crossing location

Opening location

What constitutes a single LX?
A single LX is one time step for a single agent with 324 maximum time steps.

What constitutes a single EX?
A single EX is one episode for a single agent with 324 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for reaching the goal.

Application-specific metrics
Reward of 1 minus penalty for number of steps taken.

Task 1 Variant 3
Informal Name
Simple Crossing S9N3

Description
Agent navigates through a 9x9 maze with three crossings to reach the goal.

Extrinsic parameters
Grid size: 9x9
Number of crossings: 3

Intrinsic parameters
Crossing location
Opening location

What constitutes a single LX?
A single LX is one time step for a single agent with 324 maximum time steps.

What constitutes a single EX?
A single EX is one episode for a single agent with 324 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for reaching the goal.

Application-specific metrics
Reward of 1 minus penalty for number of steps taken.

Task 2 Variant 1
Informal Name
Distribution Shift R2

Description
Agent navigates through the lava maze to reach the goal.

Extrinsic parameters
Lava strip row: 2

Intrinsic parameters
None

What constitutes a single LX?
A single LX is one time step for a single agent with 252 maximum time steps.

What constitutes a single EX?
A single EX is one episode for a single agent with 252 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for reaching the goal. Negative reward of 1 for colliding with lava.

Application-specific metrics
Reward of 1 minus penalty for number of steps taken.

Task 2 Variant 2
Informal Name
Distribution Shift R3

Description
Agent navigates through the lava maze to reach the goal.

Extrinsic parameters
Lava strip row: 3

Intrinsic parameters
None

What constitutes a single LX?
A single LX is one time step for a single agent with 252 maximum time steps.

What constitutes a single EX?
A single EX is one episode for a single agent with 252 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for reaching the goal. Negative reward of 1 for colliding with lava.

Application-specific metrics
Reward of 1 minus penalty for number of steps taken.

Task 2 Variant 3
Informal Name
Distribution Shift R5

Description
Agent navigates through the lava maze to reach the goal.

Extrinsic parameters
Lava strip row: 5

Intrinsic parameters
None

What constitutes a single LX?
A single LX is one time step for a single agent with 252 maximum time steps.

What constitutes a single EX?
A single EX is one episode for a single agent with 252 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for reaching the goal. Negative reward of 1 for colliding with lava.

Application-specific metrics
Reward of 1 minus penalty for number of steps taken.

Task 3 Variant 1
Informal Name
Dynamic Obstacles S6N1

Description
Agent navigates to the goal in a 6x6 grid while avoiding 1 moving obstacle.

Extrinsic parameters
Grid size: 6x6
Number of obstacles: 1
Agent start location/orientation: (1, 1, 0)

Intrinsic parameters
Obstacle start locations

What constitutes a single LX?
A single LX is one time step for a single agent with 144 maximum time steps.

What constitutes a single EX?
A single EX is one episode for a single agent with 144 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for reaching the goal. Negative reward of 1 for colliding with obstacles.

Application-specific metrics
Reward of 1 minus penalty for number of steps taken.

Task 3 Variant 2
Informal Name
Dynamic Obstacles S8N2

Description
Agent navigates to the goal in an 8x8 grid while avoiding two moving obstacles.

Extrinsic parameters
Grid size: 8x8
Number of obstacles: 2
Agent start location/orientation: (1, 1, 0)

Intrinsic parameters
Obstacle start locations

What constitutes a single LX?
A single LX is one time step for a single agent with 256 maximum time steps.

What constitutes a single EX?
A single EX is one episode for a single agent with 256 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for reaching the goal. Negative reward of 1 for colliding with obstacles.

Application-specific metrics
Reward of 1 minus penalty for number of steps taken.

Task 3 Variant 3
Informal Name
Dynamic Obstacles S10N3

Description
Agent navigates to the goal in a 10x10 grid while avoiding three moving obstacles.

Extrinsic parameters
Grid size: 10x10
Number of obstacles: 3
Agent start location/orientation: (1, 1, 0)

Intrinsic parameters
Obstacle start locations

What constitutes a single LX?
A single LX is one time step for a single agent with 400 maximum time steps.

What constitutes a single EX?
A single EX is one episode for a single agent with 400 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for reaching the goal. Negative reward of 1 for colliding with obstacles.

Application-specific metrics
Reward of 1 minus penalty for number of steps taken.

Task 4 Variant 1
Informal Name
Custom Fetch S5T1N2

Description
Agent must pick up the yellow key in a 5x5 grid with two other random objects.

Extrinsic parameters
Grid size: 5x5
Target type: key
Target color: yellow
Number of targets: 1
Number of objects: 2

Intrinsic parameters
Agent start location and orientation
Target locations
Object types: ball or box
Object colors: [red, green, blue, purple, yellow, grey]
Object locations

What constitutes a single LX?
A single LX is one time step for a single agent with 125 maximum time steps.

What constitutes a single EX?
A single EX is one episode for a single agent with 125 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for picking up a key.

Application-specific metrics
Reward of 1 minus penalty for number of steps taken.

Task 4 Variant 2
Informal Name
Custom Fetch S8T1N2

Description
Agent must pick up the yellow key in an 8x8 grid with two other random objects.

Extrinsic parameters
Grid size: 8x8
Target type: key
Target color: yellow
Number of targets: 1
Number of objects: 2

Intrinsic parameters
Agent start location and orientation

Target locations
Object types: ball or box
Object colors: [red, green, blue, purple, yellow, grey]
Object locations

What constitutes a single LX?
A single LX is one time step for a single agent with 320 maximum time steps.

What constitutes a single EX?
A single EX is one episode for a single agent with 320 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for picking up a key.

Application-specific metrics
Reward of 1 minus penalty for number of steps taken.

Task 4 Variant 3
Informal Name
Custom Fetch S10T2N4

Description
Agent must pick up any one of two yellow keys in a 10x10 grid with four other random objects.

Extrinsic parameters
Grid size: 10x01
Target type: key
Target color: yellow
Number of targets: 2
Number of objects: 4

Intrinsic parameters
Agent start location and orientation
Target locations
Object types: ball or box
Object colors: [red, green, blue, purple, yellow, grey]
Object locations

What constitutes a single LX?
A single LX is one time step for a single agent with 500 maximum time steps.

What constitutes a single EX?
A single EX is one episode for a single agent with 500 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for picking up a key.

Application-specific metrics

Reward of 1 minus penalty for number of steps taken.

Task 5 Variant 1
Informal Name
Custom Unlock S5

Description
Agent must pick up a key to unlock a locked door in a 5x5 grid.

Extrinsic parameters
Number of rows: 1
Number of columns: 2
Room size: 5x5

Intrinsic parameters
Key color and location
Door color and location

What constitutes a single LX?
A single LX is one time step for a single agent with 200 maximum time steps.

What constitutes a single EX?
A single EX is one episode for a single agent with 200 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for unlocking the door.

Application-specific metrics
Reward of 1 minus penalty for number of steps taken.

Task 5 Variant 2
Informal Name
Custom Unlock S7

Description
Agent must pick up a key to unlock a locked door in a 7x7 grid.

Extrinsic parameters
Number of rows: 1
Number of columns: 2
Room size: 7x7

Intrinsic parameters
Key color and location
Door color and location

What constitutes a single LX?
A single LX is one time step for a single agent with 392 maximum time steps.

What constitutes a single EX?

A single EX is one episode for a single agent with 392 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for unlocking the door.

Application-specific metrics
Reward of 1 minus penalty for number of steps taken.

Task 5 Variant 3
Informal Name
Custom Unlock S9

Description
Agent must pick up a key to unlock a locked door in a 9x9 grid.

Extrinsic parameters
Number of rows: 1
Number of columns: 2
Room size: 9x9

Intrinsic parameters
Key color and location
Door color and location

What constitutes a single LX?
A single LX is one time step for a single agent with 648 maximum time steps.

What constitutes a single EX?
A single EX is one episode for a single agent with 648 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for unlocking the door.

Application-specific metrics
Reward of 1 minus penalty for number of steps taken.

Task 6 Variant 1
Informal Name
Door Key S5

Description
Agent navigates to goal in a 5x5 grid after unlocking the door with a key.

Extrinsic parameters
Grid size: 5x5

Intrinsic parameters
Agent start location
Vertical wall location

Key color and location
Door color and location

What constitutes a single LX?
A single LX is one time step for a single agent with 250 maximum time steps.

What constitutes a single EX?
A single EX is one episode for a single agent with 250 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for reaching the goal after unlocking the door.

Application-specific metrics
Reward of 1 minus penalty for number of steps taken.

Task 6 Variant 2
Informal Name
Door Key S6

Description
Agent navigates to goal in a 6x6 grid after unlocking the door with a key.

Extrinsic parameters
Grid size: 6x6

Intrinsic parameters
Agent start location
Vertical wall location
Key color and location
Door color and location

What constitutes a single LX?
A single LX is one time step for a single agent with 360 maximum time steps.

What constitutes a single EX?
A single EX is one episode for a single agent with 360 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for reaching the goal after unlocking the door.

Application-specific metrics
Reward of 1 minus penalty for number of steps taken.

Task 6 Variant 3
Informal Name
Door Key S7

Description
Agent navigates to goal in a 7x7 grid after unlocking the door with a key.

Extrinsic parameters
Grid size: 7x7

Intrinsic parameters
Agent start location
Vertical wall location
Key color and location
Door color and location

What constitutes a single LX?
A single LX is one time step for a single agent with 490 maximum time steps.

What constitutes a single EX?
A single EX is one episode for a single agent with 490 maximum time steps where learning is disabled.

What is the learning signal?
Positive reward for reaching the goal after unlocking the door.

Application-specific metrics
Reward of 1 minus penalty for number of steps taken.

For Month 21 too, we relied on differences in the encoded observation space rather than an explicit Task ID signal for the agent to learn and exhibit different task behaviors.

Pre-deployment training of the L2 agent
For Month 21, the System Groups decided not to use any pre-deployment training. For us, this meant the non-inclusion of the Meta-Learned Instinct Network.

Single-Task Experts (STEs) per task variant
We used the same architecture as the Month 21 system to train the STEs for 1e6 time steps, and thereby characterize the ability of the STEs to learn each task variant. We collected 10 STE runs per task variant, which were all initialized with the same "ready-to-deploy" state as the Month 21 system. Performance threshold was set to the terminal performance of the smoothed learning curve averaged across runs for each task, and the number of LXs required to reach it (LX^{th}) was determined accordingly.

Month 21 Scenarios
For Month 21, we collected data for both Condensed and Dispersed Scenarios. We used APL's TELLA interface and common seed values to produce the same task sequences and environment distributions for each lifetime across the System Groups.

Scenario 1 (Complexity=3-high, Scenario Type=Condensed)
LB = learning block, EB = evaluation block.
Label for scenario
"3-high_condensed"

Structure of each run

Algorithm 5 Pseudo-code for each Month 21 scenario:

```
for each run:
    load task variant sequence for scenario (as provided by T&E)
    set L2 agent to Ready-to-deploy state
    evaluate: evaluate L2 agent on all tasks in scenario
    for each learning block in sequence:
        train: present task instances to L2 agent
        evaluate: evaluate L2 agent on all tasks in scenario
```

EB LB_{A1} EB LB_{B1} EB LB_{C1} EB LB_{C2} EB LB_{B2} EB LB_{A2} EB
For a given permutation of task variants (e.g., A1, B1, C1, C2, B2, A2), where LB_{A1} is for task variant A1, etc., a given task variant occurs exactly once during each run. Each EB includes all 18 task variants.

Generation of task sequences across runs
Generate a random permutation of the 18 Month 21 task variants.

Selection of number of runs
There will be 100 runs

Length of learning blocks within each run
LXs = 0.75 of LX^{th} for each of the task variants

Task labeling
No

Task feedback
A reward signal for each LX

Scenario 2 (Complexity=3-high, Scenario Type=Dispersed)
Label for scenario
"3-high_dispersed"

Structure of each run
A superblock should be generated for a permutation of task variants (e.g., A1, B1, C1, C2, B2, A2 ...), with a short learning block, (SLB), where SLB_{A1} is for task variant A1, etc., SLB indicates that this is a short learning block with length one third (1/S for S =3) of the Condensed LB size for that task variant. A given task variant occurs exactly once during each superblock. There will be an EB after each SLB. Each EB includes all 18 task variants. Three superblocks should be generated, each using a random permutation of the 18 task variants.

Generation of task sequences across runs
Within each of the three "superblocks", task ordering will be randomly permuted.

Selection of number of runs
There will be 100 runs

Length of learning blocks within each run

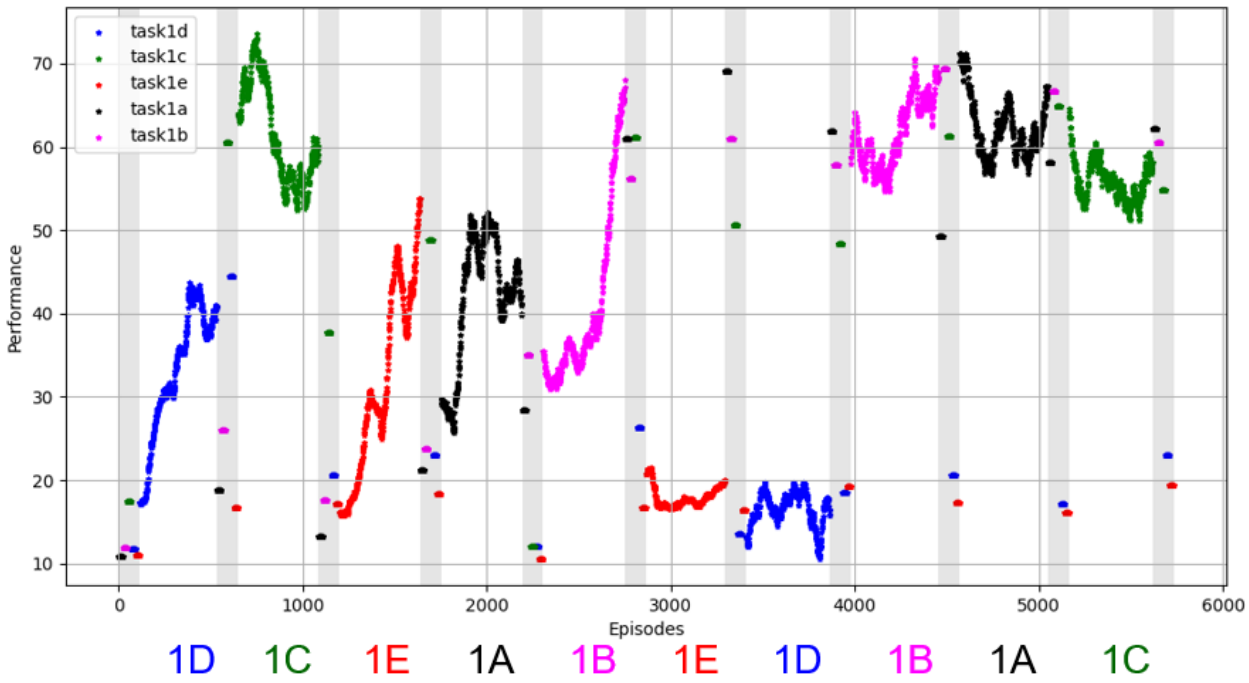


Figure 133: Normalized smoothed performance curves from an illustrative lifetime for our Month 9 system.

LXs = 0.25 of LXth for each of the task variants

Task labeling
No

Task feedback
A reward signal for each LX

2.4 Results and Discussion

2.4.1 Month 9 Evaluation

As our application-specific metric did not have a fixed range, some sort of normalization was required to be able to compute L2 metrics (Figure 133). Our particular method operated over all episode rewards across task variants and lifetimes. Outliers were identified based on quantiles of the global distribution for cumulative probabilities 0.05 and 0.95, and they were truncated to the nearest effective minimum and maximum. Min-max scaling was then applied on the episode rewards to bring them to the 0-100 range. Note in Figure 133, each lifetime comprises 10 learning blocks and 11 evaluation blocks. Each learning block comprises 250K time steps. Each evaluation block comprises 25 episodes per task variant. There are essentially two cycles of exposures to task variants, with random order in each cycle. We collected data for a total of 30 lifetimes with random permutations.

Table 17 provides a summary of Month 9 evaluation results for the 2-component Month 9 system as well as the 6-component Pre-Month-12 system. Note Performance Recovery metric could not be calculated as there were only two learning blocks per task variant in each lifetime. A minimum

Table 17: Summary of Month 9 evaluation results.

L2 Metric	Baseline Model	Month 9 system (2 components)	Pre-Month-12 system (6 components)
Performance Maintenance	-12.28	-3.027	-1.6
Forward Transfer Ratio	1.34	1.33	1.14
Backward Transfer Ratio	0.85	0.99	0.98
Performance Relative to a STE	1.082	0.86	0.81
Sample Efficiency	1.32	1.35	1.92

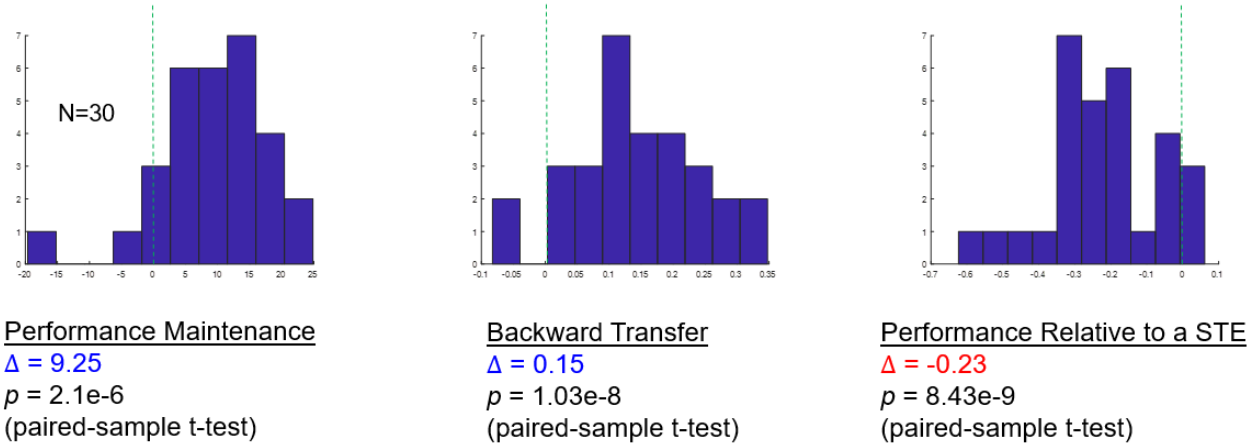


Figure 134: Post-hoc pairwise comparison of the Month 9 system relative to the Baseline Model.

of three is needed. Figure 134 shows pairwise comparison of the 2-component Month 9 system relative to the Baseline Model for L2 metrics where they were statistically different. These results indicate a trade-off between stability and plasticity, suggesting old task variants are being preserved more than needed. There is a need to tune the relative contribution of SCP-based regularization and Self-Preserving World Model replays-based distillation. And Figure 135 shows pairwise comparison of the 6-component Pre-Month-12 system relative to the Baseline Model. Interestingly, the inclusion of more L2 components led to significant differences between the L2 system and the Baseline Model for all L2 metrics.

2.4.2 Month 12 Evaluation

Figures 136 and 137 show normalized smoothed performance curves from illustrative lifetimes for the Month 12 Permuted and Alternating Scenarios, respectively, for the Month 12 system. Note that for the Month 12 Permuted Scenario, each lifetime comprises four learning blocks and five evaluation blocks. Each learning block comprises 25K time steps. Each lifetime had a different random ordering of the four tasks. We collected data for two repetitions of each of the 24 unique permutations of tasks. And for the Month 12 Alternating Scenario, each lifetime comprises six learning blocks and seven evaluation blocks. Each learning block comprises 18K time steps. Each lifetime had a different random pair of tasks, alternating with three repetitions. We collected data for four repetitions of each of the 12 unique permutations of task pairs. In both scenarios, each

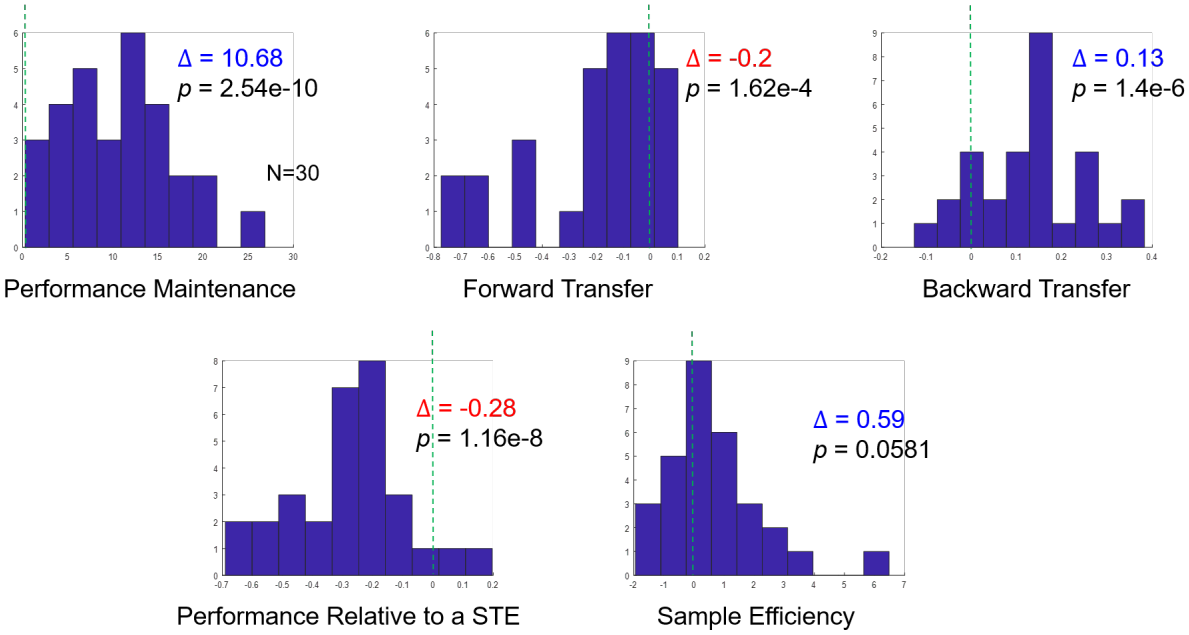


Figure 135: Post-hoc pairwise comparison of the Pre-Month-12 system relative to the Baseline Model.

evaluation block comprises 25 episodes per task. No pre-deployment training was employed, and a common "ready-to-deploy" state was used across all lifetimes.

Table 18 provides a summary of Month 12 evaluation results for the 8-component Month 12 system. Cells are color coded as follows: light red (no L2), light green (L2), and dark green (L2 above the Month 12 target). Note l2metrics v2.8.1 package was used to compute the metrics, and STE-dependent metrics were calculated per STE run and then averaged. We can observe that the Month 12 system exhibits L2 for four of the six metrics in both Permuted and Alternating Scenarios and further exceeds the Month 12 targets set by DARPA for two of them.

Table 19 provides a summary of Month 12 evaluation results for the 10-component Pre-Month-15 system. As above, cells are color coded as follows: light red (no L2), light green (L2), and dark green (L2 above the Month 12 target). Results show that the 10-component Pre-Month-15 system performs better than the 8-component Month 12 system on the Month 12 evaluation protocol (cf., Table 18). Table 20 shows comparative performance of the Month 12 system, the previous versions of the STELLAR system, as well as the Baseline Model on the Month 9 evaluation protocol. Cells are color coded as follows: light red (no L2), light green (L2), and dark green (L2 above the Month 9 target). We can observe that that the 8-component Month 12 system performs better than the previous versions and the Baseline Model, exhibiting L2 for all of the metrics and exceeding the Month 9 targets set by DARPA for two of them.

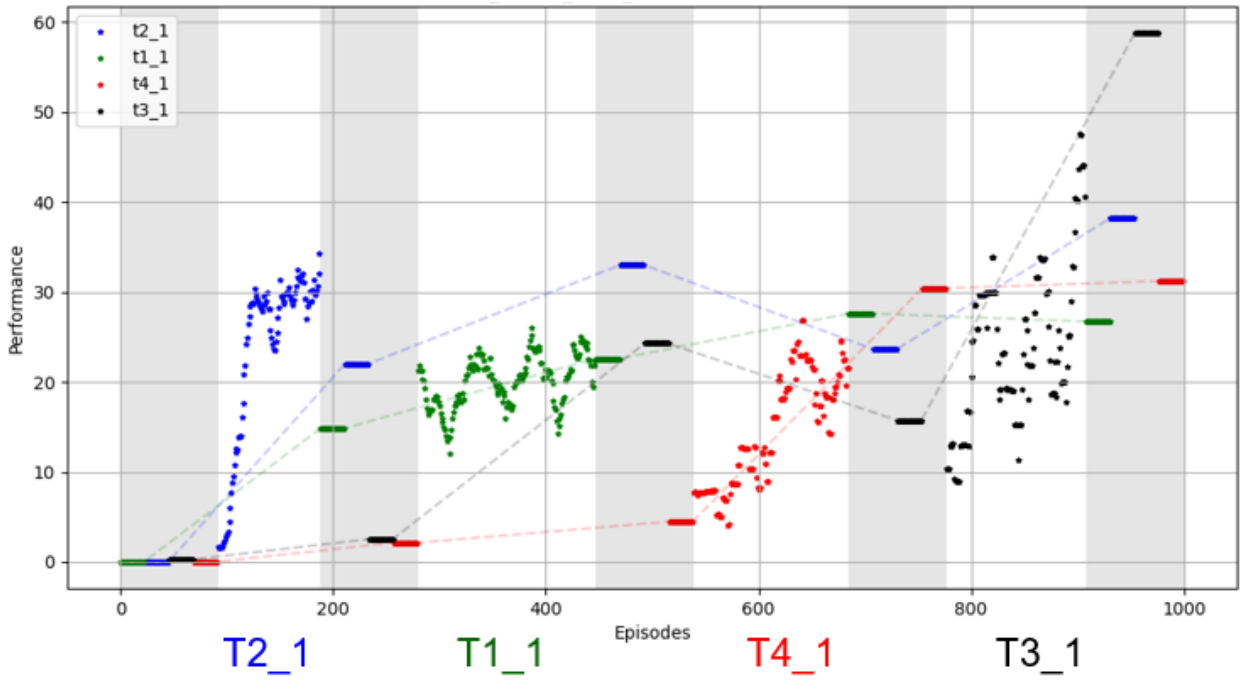


Figure 136: Normalized smoothed performance curves from an illustrative Permuted Scenario lifetime for our Month 12 system.

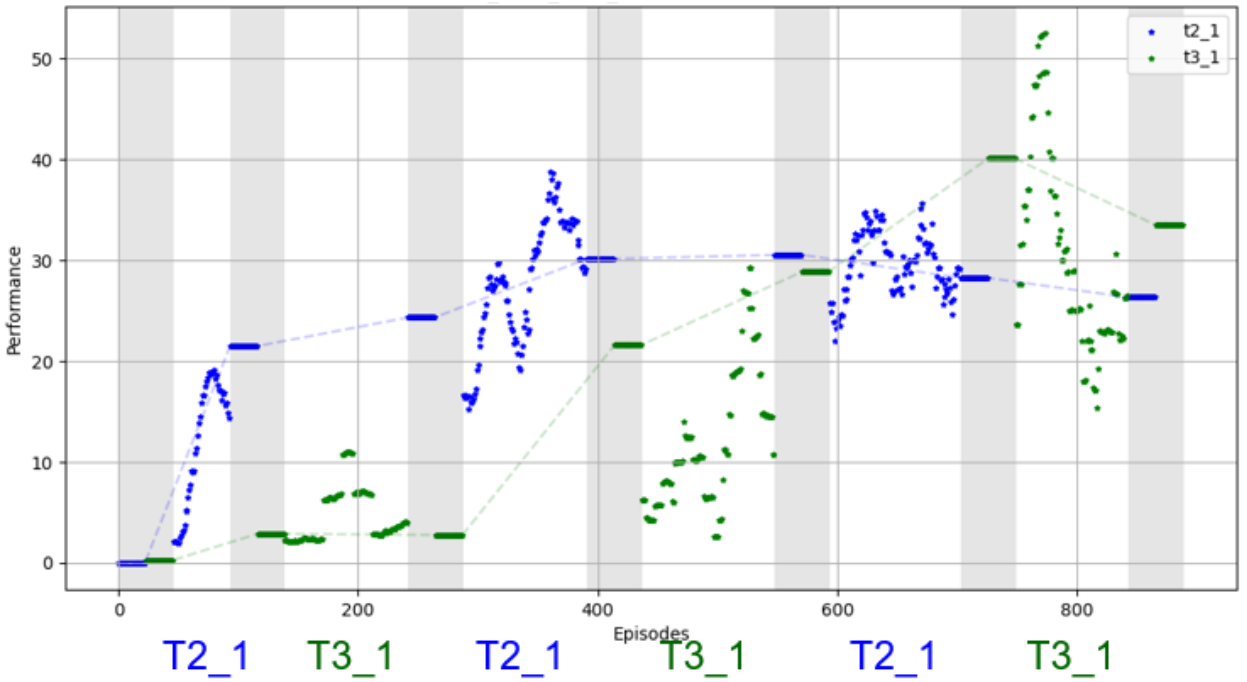


Figure 137: Normalized smoothed performance curves from an illustrative Alternating Scenario lifetime for our Month 12 system.

Table 18: Summary of Month 12 evaluation results for the Month 12 system with eight L2 components.

L2 Metric	Permuted (n=48)	Alternating (n=48)	Combined (n=96)
Performance Maintenance	-0.99	-0.63	-0.81
Forward Transfer Ratio	123.59	382.98	210.05
Backward Transfer Ratio	1.13	1.12	1.12
Performance Relative to a STE	1.15	1.07	1.11
Sample Efficiency	4.71	5.29	5
Performance Recovery	n/a	-0.31	-0.31

Table 19: Summary of Month 12 evaluation results for the Pre-Month-15 system with 10 L2 components.

L2 Metric	Permuted (n=24)	Alternating (n=12)	Combined (n=36)
Performance Maintenance	-0.08	-0.79	-0.31
Forward Transfer Ratio	129.36	433.44	188.76
Backward Transfer Ratio	1.08	0.94	1.04
Performance Relative to a STE	1.17	1.21	1.18
Sample Efficiency	4.54	9.34	4.18
Performance Recovery	n/a	5.29	5.29

Table 20: Summary of Month 9 evaluation results for various versions of the STELLAR system.

L2 Metric	Baseline (n=30)	Month 9 (n=30)	Pre-Month-12 (n=30)	Month 12 (n=30)
Performance Maintenance	-12.28	-3.69	-1.60	0.5
Forward Transfer Ratio	1.34	1.33	1.14	80.15
Backward Transfer Ratio	0.85	0.99	0.98	1.03
Performance Relative to a STE	1.082	0.86	0.81	1.05
Sample Efficiency	1.32	1.37	1.92	8.34

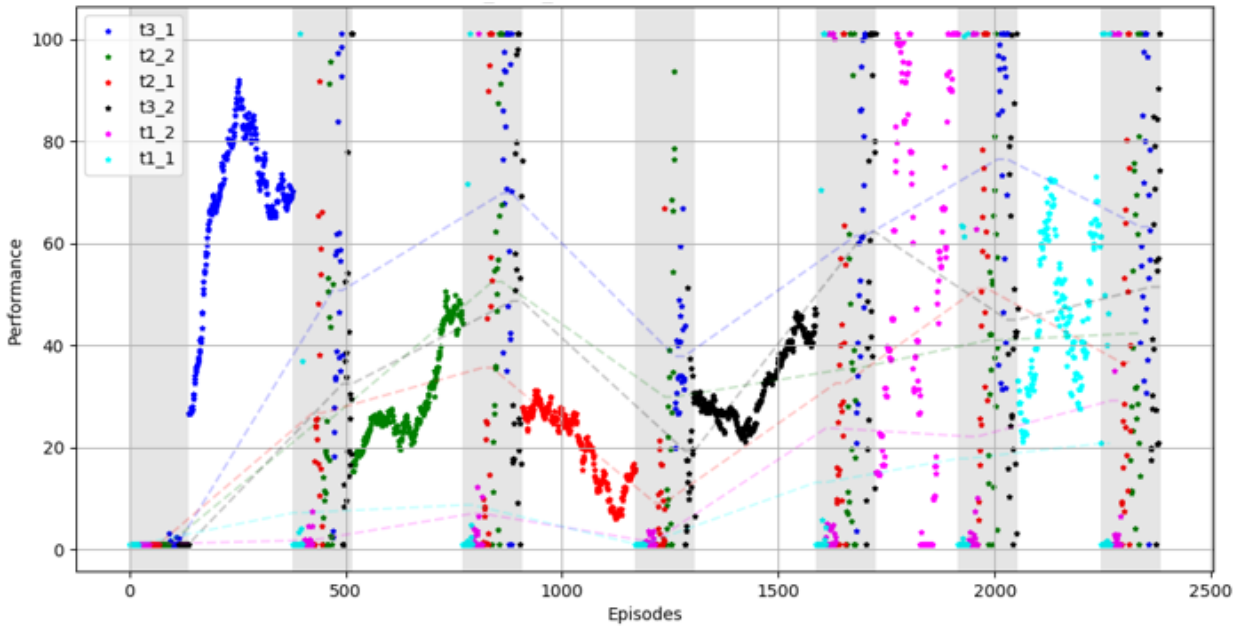


Figure 138: Normalized smoothed performance curves from an illustrative Condensed Scenario lifetime for our Month 15 system.

2.4.3 Month 15 Evaluation

Figures 138 and 139 show normalized smoothed performance curves from illustrative lifetimes for the Month 15 Condensed and Dispersed Scenarios, respectively, for the Month 15 system. Note that for the Month 15 Condensed Scenario, each lifetime comprised six learning blocks and seven evaluation blocks. Each learning block comprised 46,080 time steps. Each lifetime had a different random ordering of the six tasks. We collected data for 30 lifetimes. And for the Month 15 Dispersed Scenario, each lifetime comprised 18 learning blocks and 19 evaluation blocks. Each learning block comprised 15,360 time steps. We collected data for 48 lifetimes. In both scenarios, each evaluation block comprised 25 episodes per task. Pre-deployment training was employed for the Meta-Learned Instinct Network, and a common "ready-to-deploy" state was used across all lifetimes.

Table 21 provides a summary of the evaluation results on the Month 12 Permuted Scenario for the 8-component Month 12 system as well as the 11-component Month 15 system. Cells are color coded as follows: light red (no L2), light green (L2), and dark green (L2 above the Month 12 target). Month 12 targets for each metric are shown in the first column within parentheses. Results show that the Month 15 system was not significantly different from the Month 12 system for the Month 12 Permuted Scenario. But there were numerical improvements in four of the five L2 metrics. In particular, Performance Maintenance was greater than 0 for the Month 15 system compared to a negative value for the Month 12 system. It is possible that increasing the number of lifetimes (from n=24 to n=48) can increase the statistical power and make the improvements over the Month 12 system statistically significant.

Table 22 provides a summary of the evaluation results on the Month 12 Alternating Scenario for

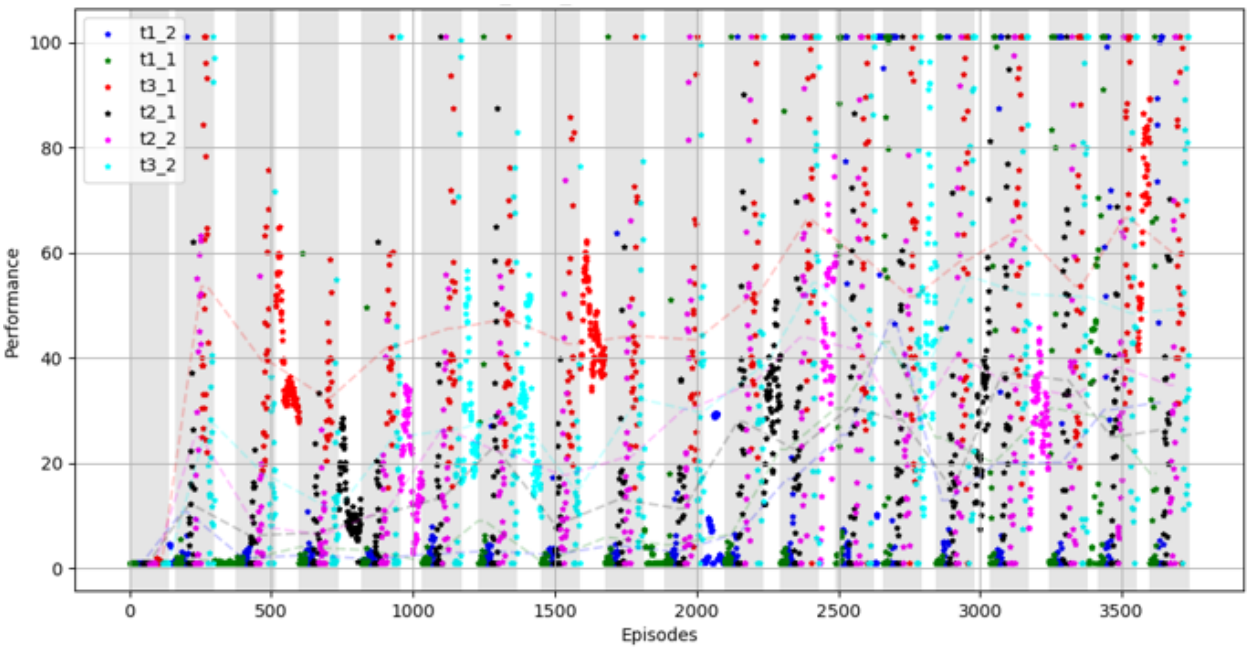


Figure 139: Normalized smoothed performance curves from an illustrative Dispersed Scenario lifetime for our Month 15 system.

the Month 12 and Month 15 systems. As above, cells are color coded as follows: light red (no L2), light green (L2), and dark green (L2 above the Month 12 target). Month 12 targets for each metric are shown in the first column within parentheses. We can observe that the Month 15 system was indeed significantly better than the Month 12 system on the Month 12 Alternating Scenario for two L2 metrics; namely, Forward Transfer (p = 0.0092) and Performance Relative to a STE (p = 2.34e-11). The Month 15 system also exhibited numerical improvements for two of the remaining three L2 metrics, including a positive Performance Maintenance value. Apart from using data from more lifetimes (n=48), another potential explanation for more significant differences in L2 metrics between the Month 12 and Month 15 systems for the Month 12 Alternating Scenario is the non-random task structure involving only two tasks leading to reduced inherent interference between tasks.

Table 23 provides a summary of the evaluation results on the Month 15 Permuted and Alternating Scenarios for the Month 15 system. Cells are color coded as follows: light red (no L2), light green (L2), and dark green (L2 above the Month 15 target). Month 15 targets for each metric are shown in the first column within parentheses. Results show that the performance of the Month 15 system was not significantly different between the Condensed and Dispersed Scenarios. However, all the L2 metrics were numerically lower for the Dispersed Scenario with the decrements being marginally significant for two L2 metrics; namely, Forward Transfer Ratio (p = 0.089) and Performance Relative to a STE (p = 0.038). Potential explanations for the across-the-board numerical decrements in the metrics include the increased cost of switching among tasks in the Dispersed Scenario, greater interference from other tasks in the intervals between learning blocks for a given task, and the lack of any dependence of the strength of the consolidation mechanisms (SCP++, Self-Preserving World Model) on the performance levels acquired in the preceding learning blocks. In the Dispersed

Table 21: Summary of evaluation on the Month 12 Permuted Scenario for the Month 12 and Month 15 systems.

L2 Metric	Month 12 system (n=48)	Month 15 system (n=24)
Performance Maintenance (4)	-1.25	0.003
Forward Transfer Ratio (1.5)	14.76	15.59
Backward Transfer Ratio (1.5)	1.24	1.08
Performance Relative to a STE (1.6)	1.61	1.91
Sample Efficiency (2)	3.28	2.56

Table 22: Summary of evaluation on the Month 12 Alternating Scenario for the Month 12 and Month 15 systems.

L2 Metric	Month 12 system (n=48)	Month 15 system (n=48)
Performance Maintenance (4)	-0.61	0.94
Forward Transfer Ratio (1.5)	27.11	33.72
Backward Transfer Ratio (1.5)	1.22	1.09
Performance Relative to a STE (1.6)	0.97	1.74
Sample Efficiency (2)	5.75	8.05

Scenario, task performances in earlier learning blocks are not expected to be high. In this case, strong preservation of sub-optimal task representations would interfere with subsequent learning blocks. So, the hyperparameters that control the degree of preservation should be reduced to improve all L2 metrics.

2.4.4 Month 18 Evaluation

Table 24 provides a summary of the evaluation results on the Month 18 Condensed Scenario for the Month 15 and Month 18 systems. Cells are color coded as follows: light red (no L2), light green (L2), and dark green (L2 above the Month 18 target). Month 18 targets for each metric are shown in

Table 23: Summary of evaluation on the Month 15 Condensed and Dispersed Scenarios for the Month 15 system.

L2 Metric	Condensed (n=33)	Dispersed (n=30)	Mann-Whitney U Test
Performance Maintenance (6)	-0.24 (± 5.73)	-2.21 (± 3.16)	$p = 0.13$
Forward Transfer Ratio (3)	10.02 (± 4.92)	10.71 (± 2.78)	$p = 0.089$
Backward Transfer Ratio (2)	1.19 (± 0.26)	1.10 (± 0.13)	$p = 0.53$
Performance Relative to a STE (1.9)	2.49 (± 1.31)	1.85 (± 0.71)	$p = 0.038$
Sample Efficiency (2.4)	10.02 (± 13.88)	6.25 (± 3.12)	$p = 0.27$

Table 24: Summary of evaluation on the Month 18 Condensed Scenario for the Month 15 and Month 18 systems.

L2 Metric	Month 15 (n=30)	Month 18 (n=30)	Wilcoxon Signed Rank Test
Performance Maintenance (6)	-0.24 (± 5.92)	-1.92 (± 4.08)	$p = 0.3$
Forward Transfer Ratio (3)	12.06 (± 4.88)	11.62 (± 4.53)	$p = 0.75$
Backward Transfer Ratio (2)	1.19 (± 0.27)	1.06 (± 0.12)	$p = 0.033$
Performance Relative to a STE (1.9)	2.52 (± 1.31)	2.37 (± 1.27)	$p = 0.77$
Sample Efficiency (2.4)	7.43 (± 8.12)	18.17 (± 19.08)	$p = 0.0029$

Table 25: Summary of the effects of SCP++ ablation on the Month 18 Condensed Scenario for the Month 18 system.

L2 Metric	STELLAR (n=33)	SCP++ Ablation (n=33)	Wilcoxon Signed Rank Test
Performance Maintenance (6)	-0.24 (± 5.73)	1.33 (± 8.32)	$p = 0.42$
Forward Transfer Ratio (3)	12.01 (± 4.92)	11.99 (± 5.11)	$p = 0.84$
Backward Transfer Ratio (2)	1.15 (± 0.19)	1.85 (± 1.14)	$p = 0.0005$
Performance Relative to a STE (1.9)	2.49 (± 1.31)	2.37 (± 1.09)	$p = 0.70$
Sample Efficiency (2.4)	8.04 (± 8.33)	7.49 (± 9.21)	$p = 0.74$

the first column within parentheses. Second and third columns show means with standard error of mean in parentheses. And the fourth column provides the p-value of the Wilcoxon Signed Rank Test for differences between the Month 15 and Month 18 systems.

As expected, we found that the Month 18 system was significantly better than the Month 15 system for Sample Efficiency by about 2 .5x. However, we did not observe a significant change in Performance Relative to a STE. This could be due to Reflexive Adaptation slowing learning to maximize safety, weakening the effect of PNN on the Performance Relative to STE. To investigate this further, we will need to compare the effects of ablating Reflexive Adaptation from those of enhancing PNN.

Contrary to hypotheses, SCP++ ablation caused a significant increase in Backward Transfer (by about 60%) and did not cause a drop in Performance Maintenance (Table 25). The former result could be due to how SCP++ aims to preserve the input-to-output mapping, limiting backward transfer. The latter result could be due to Performance Maintenance having a nonlinear (inverted U) dependence on SCP++ stiffness. There is a gradient of importance values across the hierarchy of a deep network with higher importances at downstream layers. When learning a new task, higher stiffness increases pressure on upstream layers to modify weights (due to lower importances) leading to catastrophic forgetting. Note that in Table 25, as above, cells are color coded as follows: light red (no L2), light green (L2), and dark green (L2 above the Month 18 target). Month 18 targets for each metric are shown in the first column within parentheses. Second and third columns show means with standard error of mean in parentheses.

There was a marginally significant effect of increasing the number of replays on Performance

Table 26: Summary of the effects of the number of replays on the Month 18 Condensed Scenario for the Month 18 system with SCP++ ablated.

L2 Metric	1x replays (n=33)	5x replays (n=33)	10x replays (n=33)
Performance Maintenance (6)	1.33 (± 8.32)	-2.01 (± 9.53)	-3.31 (± 8.09)
Forward Transfer Ratio (3)	11.99 (± 5.11)	12.64 (± 5.83)	13.37 (± 5.47)
Backward Transfer Ratio (2)	1.85 (± 1.14)	1.78 (± 1.01)	1.62 (± 1.10)
Performance Relative to a STE (1.9)	2.37 (± 1.09)	2.21 (± 1.18)	2.33 (± 1.17)
Sample Efficiency (2.4)	7.49 (± 9.21)	11.18 (± 12.70)	10.70 (± 10.71)

Table 27: Summary of the effects of PNN ablation on the Month 18 Condensed Scenario for the Month 18 system.

L2 Metric	Month 18 (n=33)	PNN Ablation (n=33)	Wilcoxon Signed Rank Test
Performance Maintenance (6)	-0.24 (± 5.73)	0.41 (± 5.18)	$p = 0.37$
Forward Transfer Ratio (3)	12.01 (± 4.92)	13.31 (± 3.12)	$p = 0.49$
Backward Transfer Ratio (2)	1.15 (± 0.19)	1.09 (± 0.12)	$p = 0.32$
Performance Relative to a STE (1.9)	2.49 (± 1.31)	2.44 (± 1.01)	$p = 0.82$
Sample Efficiency (2.4)	8.04 (± 8.33)	6.79 (± 4.51)	$p = 0.94$

Maintenance (Kruskal-Wallis Test, $p = 0.11$; Table 26). However, we found a significant decrease in Performance Maintenance going from 1x replays to 3x replays (Wilcoxon Signed Rank Test, $p = 0.046$), uncorrected for multiple comparisons. This contrary result could be based on the difference between explicit and generative replays with explicit replays promoting Performance Maintenance, while generative replays promoting Forward Transfer at the cost of Performance Maintenance. The next step would certainly be to investigate the effects of explicit replays. Another factor to look at is the frequency (instead of number) of replay updates. Note that in Table 26, as above, cells are color coded as follows: light red (no L2), light green (L2), and dark green (L2 above the Month 18 target). Month 18 targets for each metric are shown in the first column within parentheses. Second through fourth columns show means with standard error of mean in parentheses.

Contrary to hypothesis, PNN ablation did not cause any significant changes to the metrics (Table 27). This may change with a larger layer size within the PNN component. Further, plastic neuromodulation can be applied to other parts of the STELLAR system such as the Actor-Critic network, Context-Skill Model, and the Encoder. Note that in Table 27, as above, cells are color coded as follows: light red (no L2), light green (L2), and dark green (L2 above the Month 18 target). Month 18 targets for each metric are shown in the first column within parentheses. Second and third columns show means with standard error of mean in parentheses.

As expected, SCP++ stiffness reduction resulted in improvements in three of the five L2 metrics; namely, Performance Maintenance (from -2.73 to 0.26), Backward Transfer Ratio by about 10%, and Performance Relative to a STE by about 30% (Table 28). But the manipulation also caused decrements in the other two metrics; namely, Forward Transfer Ratio by about 4% and Sample Efficiency by about 50%. Of these effects, the improvement in Performance Relative to a STE ($p =$

Table 28: Summary of the effects of reducing SCP++ stiffness on the Month 15 Dispersed Scenario for the Month 18 system.

L2 Metric	Month 18 (n=15)	Reduced SCP++ stiffness (n=15)	Wilcoxon Signed Rank Test
Performance Maintenance (6)	-2.73 (± 2.71)	0.26 (± 3.84)	p = 0.055
Forward Transfer Ratio (3)	9.96 (± 2.16)	9.52 (± 2.97)	p = 1.00
Backward Transfer Ratio (2)	1.15 (± 0.18)	1.27 (± 0.29)	p = 0.23
Performance Relative to a STE (1.9)	1.57 (± 0.49)	2.07 (± 0.44)	p = 0.022
Sample Efficiency (2.4)	7.07 (± 3.44)	3.23 (± 1.42)	p = 0.0026

Table 29: Summary of evaluation results on the Month 21 Condensed Scenario.

L2 Metric	Month 21 (n=100)	Baseline (n=100)	Wilcoxon Signed Rank Test
Performance Maintenance (6)	-9.39 (± 5.45)	-33.51 (± 13.5)	p = 4.22e-17
Forward Transfer Contrast (0)	0.014 (± 0.021)	-0.0094 (± 0.016)	p = 2.81e-13
Backward Transfer Contrast (0)	-0.013 (± 0.015)	-0.06 (± 0.028)	p = 1.8e-16
Performance Relative to a STE (1.9)	2.47 (± 1.61)	2.31 (± 1.36)	p = 0.36
Sample Efficiency (2.4)	17.46 (± 5.79)	16.04 (± 5.71)	p = 0.049

0.022) and the decrement in Sample Efficiency (p=0.0026) were statistically significant, and the improvement in Performance Maintenance (p=0.055) was marginally significant. More work will be needed to understand the dynamics of L2 for task repetitions in the context of the multi-component STELLAR system. It may be the case that the degree of consolidation (structural regularization, interleaving of explicit/generative replays) should be further contingent on task learning. Note that in Table 28, cells are color coded as follows: light red (no L2), light green (L2), and dark green (L2 above the Month 15 target). Month 15 targets for each metric are shown in the first column within parentheses. Second and third columns show means with standard error of mean in parentheses.

In summary, we learned that the effects of the key parameter for each component/algorithm need to be characterized to inform mechanistic understanding. Design choices would then be driven by this understanding. And hyperparameter fine-tuning must be done in the context of the integrated system. SCP and SCP++ may have a nonlinear (inverted U) dependence on stiffness of important weights for performance maintenance; there is a trade-off between performance maintenance and backward transfer. Explicit replays may be better suited for performance maintenance vs. generative replays, which promote generalization and transfer. Consolidation (SCP++, explicit replays) should be contingent on task learning.

2.4.5 Month 21 Evaluation

Table 29 provides a summary of the evaluation results on the Month 21 Condensed Scenario for the Month 21 system and the Baseline Model. Month 21 targets for each metric are shown in the first column within parentheses. The second and third columns show means with standard deviation in parentheses.

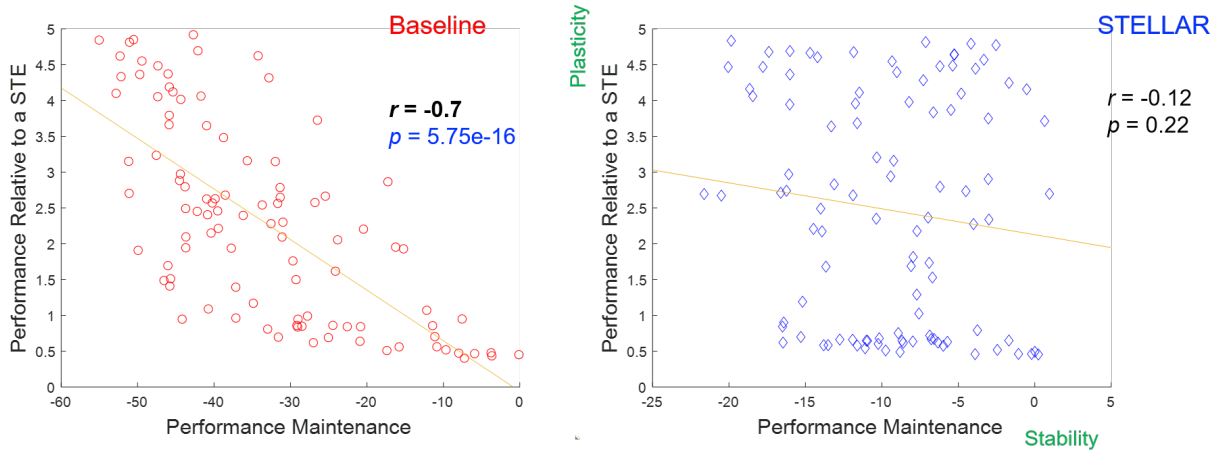


Figure 140: Baseline Model suffers from stability-plasticity dilemma unlike the STELLAR system.

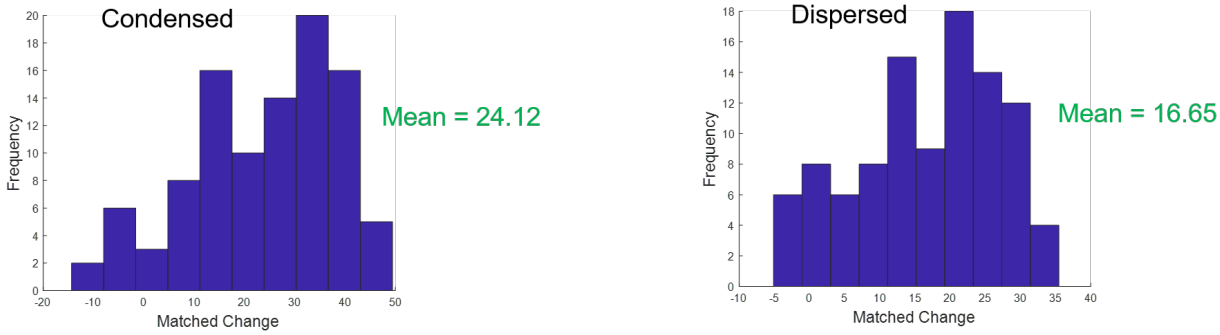


Figure 141: Histograms of matched differences in Performance Maintenance between the STELLAR system and the Baseline Model.

For the Condensed Scenario, we found that the Baseline Model exhibited fast-learning as well as fast-forgetting. In particular, on the Performance Maintenance, it fared very poorly. Month 21 STELLAR system improved Performance Maintenance significantly by about 3.5x and Backward Transfer by about 4.5x. Unlike the Baseline Model, STELLAR did not exhibit the most forgetting on the most learned lifetimes (Figure 140). In particular, only the Baseline Model exhibits a significant negative correlation between the Performance Relative to a STE and Performance Maintenance. Differences of the Performance Maintenance metric for the STELLAR system and the Baseline Model on the matched n=100 lifetimes demonstrate L2 on the Month 12 Condensed and Dispersed Scenarios (Figure 141).

Table 30 provides a summary of the effects of adding Context-Skill Model and Plastic Neuromodulated Network to the Month 21 system for the Month 21 Condensed Scenario. Month 21 targets for each metric are shown in the first column within parentheses. The second and third columns show means with standard deviation in parentheses. We can observe that enabling these components in the STELLAR system improved Performance Maintenance significantly by about 35% and Sample Efficiency by about 20%. There was also a numerical improvement in Performance Relative to a

Table 30: Summary of the effects of adding CSM and PNN to the Month 21 system for the Month 21 Condensed Scenario.

L2 Metric	Month 21 (n=100)	Post-Month-21 (n=100)	Wilcoxon Signed Rank Test
Performance Maintenance (6)	-8.96 (± 5.33)	-6.55 (± 4.1)	$p = 0.0038$
Forward Transfer Contrast (0)	0.011 (± 0.02)	0.0027 (± 0.022)	$p = 0.014$
Backward Transfer Contrast (0)	-0.013 (± 0.016)	-0.013 (± 0.013)	$p = 0.57$
Performance Relative to a STE (1.9)	2.21 (± 1.54)	3.27 (± 3.9)	$p = 0.1$
Sample Efficiency (2.4)	17.3 (± 5.77)	20.77 (± 4.22)	$p = 4.1e\text{-}4$

Table 31: Summary of the effects of ablating SCP++ in the Month 21 system for the Month 21 Condensed Scenario.

L2 Metric	Month 21 (n=10)	SCP++ Ablation (n=10)	Wilcoxon Signed Rank Test
Performance Maintenance (6)	-5.18 (± 3.65)	-40.98 (± 11.89)	$p = 0.002$
Forward Transfer Contrast (0)	0.021 (± 0.023)	-0.0038 (± 0.01)	$p = 0.027$
Backward Transfer Contrast (0)	-0.011 (± 0.01)	-0.064 (± 0.032)	$p = 0.002$
Performance Relative to a STE (1.9)	3.8 (± 1.31)	3.68 (± 1.6)	$p = 0.85$
Sample Efficiency (2.4)	14.53 (± 6.29)	13.75 (± 4.15)	$p = 0.77$

STE by about 40%.

Table 31 provides a summary of the effects of ablating SCP++ in the Month 21 system for the Month 21 Condensed Scenario. Month 21 targets for each metric are shown in the first column within parentheses. The second and third columns show means with standard deviation in parentheses. We can observe that SCP++ ablation caused a significant drop in Performance Maintenance by about 8x. This contrasts with the outcome of the SCP++ ablation experiment in the Month 18 evaluation, highlighting the importance of hyperparameter tuning. SCP++ ablation also caused significant decrements in both Forward Transfer and Backward Transfer.

Table 32 provides a summary of evaluation results on the Month 21 Dispersed Scenario for the Month 21 system and the Baseline Model. Month 21 targets for each metric are shown in the first column within parentheses. The second and third columns show means with standard deviation

Table 32: Summary of evaluation results on the Month 21 Dispersed Scenario for the Month 21 system and the Baseline Model.

L2 Metric	Month 21 (n=100)	Baseline (n=100)	Wilcoxon Signed Rank Test
Performance Maintenance (6)	-4.32 (± 2.37)	-20.97 (± 9.96)	$p = 2.88e\text{-}17$
Forward Transfer Contrast (0)	0.018 (± 0.023)	0.006 (± 0.02)	$p = 2.38e\text{-}5$
Backward Transfer Contrast (0)	-0.0071 (± 0.01)	-0.029 (± 0.018)	$p = 2.4e\text{-}16$
Performance Relative to a STE (1.9)	2.2 (± 1.33)	2.86 (± 1.69)	$p = 7.25e\text{-}4$
Sample Efficiency (2.4)	20.2 (± 4.6)	21.13 (± 3.84)	$p = 0.1$

Table 33: Comparison of the performance of the Month 21 system on the Month 21 Condensed and Dispersed Scenarios.

L2 Metric	Condensed (n=100)	Dispersed (n=100)	Mann-Whitney U Test
Performance Maintenance (6)	-9.39 (± 5.45)	-4.32 (± 2.37)	p = 8.64e-13
Forward Transfer Contrast (0)	0.014 (± 0.021)	0.018 (± 0.023)	p = 0.26
Backward Transfer Contrast (0)	-0.013 (± 0.015)	-0.0071 (± 0.01)	p = 8.12e-4
Performance Relative to a STE (1.9)	2.47 (± 1.61)	2.2 (± 1.33)	p = 0.26
Sample Efficiency (2.4)	17.46 (± 5.79)	20.2 (± 4.6)	p = 8.64e-4

in parentheses. For the Dispersed Scenario as well, we found that the Baseline Model exhibited fast-learning as well as fast-forgetting. It again fared very poorly on the Performance Maintenance metric. The STELLAR system improved Performance Maintenance significantly by about 5x, Forward Transfer by about 3x, and Backward Transfer by about 4x.

Table 33 provides a comparison of the performance of the Month 21 system on the Month 21 Condensed and Dispersed Scenarios. We can observe that the performance of the Month 21 system on the Dispersed Scenario was significantly better compared to that on Condensed Scenario. In particular, the Month 21 system improved Performance Maintenance on Dispersed Scenario significantly by about 2x, Backward Transfer by about 2x, and Sample Efficiency by about 15%. This contrasts with the Month 18 evaluation, where we observed across-the-board decrements in all L2 metrics for the Dispersed Scenario compared to the Condensed Scenario. We hypothesize this is the effect of implementing performance-gated consolidation (SCP++, experience replays). Namely, for the Month 21 evaluation, learning blocks that didn't cross task-specific minimal thresholds were not consolidated. If true, this result suggests that consolidating sub-threshold, noisy task representations impaired L2.

2.5 Conclusions

Given all the evaluations in Phase 2, we have a number of lessons learned. First, the effects of key parameter(s) for each component/algorithm must be characterized to inform mechanistic understanding that must drive design choices. Hyperparameters of individual components should be tuned in the context of integrated system. Second, selective plasticity methods may have a nonlinear (inverted U) dependence on stiffness of important weights for performance maintenance; there is a trade-off between performance maintenance and backward transfer. Third, explicit replays may be better suited for performance maintenance vs. generative replays, which promote generalization and transfer. Fourth, consolidation (selective plasticity, replays) should be contingent on task learning, as consolidating sub-threshold, noisy task representations very likely impairs L2. Fifth, future work needs to adequately solve the continual RL challenge related to how switching to an unknown new task needs guided exploration. Sixth, there is also a need to investigate learning implicit task representations vs. leveraging explicit task information when tasks do not differ in input space. Seventh, L2 metrics cannot be interpreted independently. Performance Maintenance and Relative Performance relative to a STE metrics taken together are the most useful to assess L2. A genuine lifelong learner needs to show learning that improves with more experiences as well as

maintenance of continuously improving performance. And finally, system integration of multiple components should initially leverage lightweight environments to get a better understanding of which configurations of components and hyperparameters do not work well for particular classes of challenges and problems

Both civilian and government customers can benefit immensely from HRL's STELLAR technology to make their autonomous platforms more adaptive and sustainable during deployment. In particular, their autonomous systems will be able to safely and rapidly adapt to new conditions (surprises) during fielded operation in a scalable manner, leading to significantly reduced downtime, repair, and rebuild costs through their lifespans. As HRL's Phase 2 demonstration was geared towards autonomous driving, the team has identified transition opportunities to semi-autonomous driving applications such as (1) the personalization to a user on the daily commute through continual imitation learning over several weeks, and (2) the continual adaptation to new situations (including corner cases) from user interventions or imitating expert drivers. L2 is also critical for human operators to trust autonomous systems. For instance, an autonomous pilot system is expected to quickly adapt its policies in the real world with only pre-deployment training in a simulator. It is expected to fly in new weather conditions or fly a new aircraft. And it must cope with any rapidly changing new conditions. To retain human operator trust, the autonomous pilot must prioritize safety. Collisions and crashes must be avoided while adapting to unexpected situations when the likelihood of errors is high.

Follow-on Development
HRL team has obtained multi-year internal research funding for further development of L2M technology to enable one-shot generalization to corner cases that autonomous systems, such as self-driving cars, invariably encounter during deployment. A corner case is defined as a novel variant or situation where the autonomous system exhibits catastrophic failure due to lack of any or sufficient exposure during pre-deployment training.

The robustness of any ML solution is fundamentally bound by the data it was trained on. One way to generalize beyond the original training is through human informed augmentation of the original dataset, however this augmentation will always be insufficient provided a realistically varied environment in which the solution is deployed. To address this limitation our internal project combines model-based RL and model-interpretability methods to further develop the Self-Preserving World Model that allows learned systems to self-generate data conditioned on low-dimensional *concept* representations of the input space that are sensitive to the agent's actions.

3. References

[1] P. Ladosz, E. Ben-Iwhiwhu, J. Dick, N. Ketz, S. Kolouri, J. L. Krichmar, P. K. Pilly, and A. Soltoggio, "Deep reinforcement learning with modulated Hebbian plus Q-network architecture," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 152, pp. 2045–2056, 2022.

[2] S. Kolouri, N. A. Ketz, A. Soltoggio, and P. K. Pilly, "Sliced Cramer synaptic consolidation for preserving deeply learned representations.," in *Proceedings of the International Conference on Learning Representations*, 2020.

[3] Zou, Kolouri, Pilly, and Krichmar, "Neuromodulated attention and goal-driven perception in uncertain domains," *Neural Networks*, vol. 125, pp. 56–69, 2020.

[4] N. Ketz, S. Kolouri, and P. K. Pilly, "Using world models for pseudo-rehearsal in continual learning," *arXiv 1903.02647*, 2019.

[5] Tutum, Abdulquddos, and Miikkulainen, "Generalization of agent behavior through explicit representation of context," *Proceedings of the IEEE Conference on Games*, 2021.

[6] Maguire, Ketz, Pilly, and Mouret, "An online data-driven emergency-response method for autonomous agents in unforeseen situations," *arXiv 2112.09670*, 2021.

[7] Martin and Pilly, "Probabilistic program neurogenesis," *Proceedings of the Conference on Artificial Life*, 2019.

[8] Grbic and Risi, "Safer reinforcement learning through transferable instinct networks," *Proceedings of the Conference on Artificial Life*, 2021.

[9] Ben-Iwhiwhu, Dick, Ketz, Pilly, and Soltoggio, "Context meta-reinforcement learning via neuromodulation," *Neural Networks*, vol. 152, pp. 70–79, 2022.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," *arXiv 1312.5602*, 2013.

[11] D. Wierstra, A. Fö rster, J. Peters, and J. Schmidhuber, "Solving deep memory POMDPs with recurrent policy gradients," *International Conference on Artificial Neural Networks*, pp. 697–706, 2007.

[12] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," in *AAAI Fall Symposium Series*, 2015.

[13] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, "Memory-based control with recurrent neural networks," *arXiv 1512.04455*, 2015.

[14] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, *et al.*, "Learning to navigate in complex environments," *arXiv 1611.03673*, 2016.

[15] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber, "Recurrent policy gradients," *Logic Journal of the IGPL*, vol. 18, pp. 620–634, 09 2009.

[16] P. Zhu, X. Li, and P. Poupart, "On improving deep reinforcement learning for POMDPs," *arXiv 1804.06309*, 2017.

[17] T. P. Le, N. A. Vien, and T. Chung, "A deep hierarchical reinforcement learning algorithm in partially observable Markov decision processes," *IEEE Access*, vol. 6, pp. 49089–49102, 2018.

[18] J. Beck, K. Ciosek, S. Devlin, S. Tschiatschek, C. Zhang, and K. Hofmann, "AMRL: Aggregated memory for reinforcement learning.," in *International Conference on Learning Representations*, 2020.

[19] T. Stepleton, R. Pascanu, W. Dabney, S. M. Jayakumar, H. Soyer, and R. Munos, "Low-pass recurrent neural networks - a memory architecture for longer-term correlation discovery," *arXiv 1805.04955*, 2018.

[20] E. Parisotto and R. Salakhutdinov, "Neural map: Structured memory for deep reinforcement learning," *International Conference on Learning Representations*, 2018.

[21] J. Oh, V. Chockalingam, S. Singh, and H. Lee, "Control of memory, active perception, and action in Minecraft," in *Proceedings of the International Conference on Machine Learning*, p. 2790–2799, 2016.

[22] M. Zhang, Z. McCarthy, C. Finn, S. Levine, and P. Abbeel, "Learning deep neural network policies with continuous memory states," in *International Conference on Robotics and Automation*, pp. 520–527, 2016.

[23] D. Steckelmacher, D. M. Roijers, A. Harutyunyan, P. Vrancx, and A. Nowé, "Reinforcement learning in POMDPs with memoryless options and option-observation initiation sets," in *AAAI Conference on Artificial Intelligence*, 2018.

[24] D. Precup, *Temporal Abstraction in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst, 2000.

[25] R. McAllister and C. E. Rasmussen, "Data-efficient reinforcement learning in continuous state-action Gaussian-POMDPs," in *Neural Information Processing Systems*, pp. 2040–2049, 2017.

[26] M. P. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proceedings of the International Conference on International Conference on Machine Learning*, p. 465–472, 2011.

[27] M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson, "Deep variational reinforcement learning for POMDPs," in *Proceedings of the International Conference on Machine Learning*, pp. 2117–2126, 2018.

[28] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology Press, 1949.

[29] A. Soltoggio and J. J. Steil, "Solving the distal reward problem with rare correlations," *Neural Computation*, vol. 25, pp. 940–978, 2013.

[30] T. Miconi, K. Stanley, and J. Clune, "Differentiable plasticity: Training plastic neural networks with backpropagation," *International Conference on Machine Learning*, pp. 3559–3568, 2018.

[31] T. Miconi, A. Rawal, J. Clune, and K. O. Stanley, "Backpropamine: Training self-modifying neural networks with differentiable neuromodulated plasticity," *arXiv 2002.10585*, 2020.

[32] E. M. Izhikevich, "Solving the distal reward problem through linkage of STDP and dopamine signaling," *Cerebral Cortex*, vol. 17, no. 10, pp. 2443–2452, 2007.

[33] A. Soltoggio, A. Lemme, F. Reinhart, and J. J. Steil, "Rare neural correlations implement robotic conditioning with delayed rewards and disturbances," *Frontiers in Neurorobotics*, vol. 7, p. 6, 2013.

[34] A. Soltoggio, F. Reinhart, A. Lemme, and J. Steil, "Learning the rules of a game: neural conditioning in human-robot interaction with delayed rewards," in *IEEE Joint International Conference on Development and Learning and Epigenetic Robotics*, pp. 1–6, 2013.

[35] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[36] R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare, "Safe and efficient off-policy reinforcement learning," in *International Conference on Neural Information Processing Systems*, pp. 1054–1062, 2016.

[37] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992.

[38] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, pp. 1928–1937, 2016.

[39] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell, "The Malmo platform for artificial intelligence experimentation," in *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 4246–4247, 2016.

[40] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," *The Psychology of Learning & Motivation*, vol. 24, pp. 104–169, 1989.

[41] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. Torr, "Riemannian walk for incremental learning: Understanding forgetting and intransigence," in *Proceedings of the European Conference on Computer Vision*, pp. 532–547, 2018.

[42] J. Schwarz, J. Luketina, W. M. Czarnecki, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell, "Progress & compress: A scalable framework for continual learning," *arXiv 1805.06370*, 2018.

[43] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, "Overcoming catastrophic forgetting by incremental moment matching," in *Advances in Neural Information Processing Systems*, pp. 4652–4662, 2017.

[44] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[45] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *Proceedings of the European Conference on Computer Vision*, pp. 139–154, 2018.

[46] D. T. Gillespie, "A theorem for physicists in the theory of random variables," *American Journal of Physics*, vol. 51, no. 6, pp. 520–533, 1983.

[47] C. Villani, *Optimal transport: Old and new*, vol. 338. Springer Science & Business Media, 2008.

[48] S. Kolouri, S. R. Park, M. Thorpe, D. Slepcev, and G. K. Rohde, "Optimal mass transport: Signal processing and machine-learning applications," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 43–59, 2017.

[49] H. Cramér, "On the composition of elementary errors: First paper: Mathematical deductions," *Scandinavian Actuarial Journal*, vol. 1928, no. 1, pp. 13–74, 1928.

[50] G. J. Székely and M. L. Rizzo, "Energy statistics: A class of statistics based on distances," *Journal of Statistical Planning and Inference*, vol. 143, no. 8, pp. 1249–1272, 2013.

[51] M. G. Bellemare, I. Danihelka, W. Dabney, S. Mohamed, B. Lakshminarayanan, S. Hoyer, and R. Munos, "The Cramér distance as a solution to biased Wasserstein gradients," *arXiv 1705.10743*, 2017.

[52] J. Dedecker and F. Merlevède, "The empirical distribution function for dependent variables," *ESAIM: Probability and Statistics*, vol. 11, pp. 102–114, 2007.

[53] S. Kolouri, Y. Zou, and G. K. Rohde, "Sliced-Wasserstein kernels for probability distributions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4876–4884, 2016.

[54] S. Kolouri, P. E. Pope, C. E. Martin, and G. K. Rohde, "Sliced Wasserstein auto-encoders," in *International Conference on Learning Representations*, 2019.

[55] S. Kolouri, G. K. Rohde, and H. Hoffmann, "Sliced Wasserstein distance for learning Gaussian mixture models," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3427–3436, 2018.

[56] J. Tabor, S. Knop, P. Spurek, I. Podolak, M. Mazur, and S. Jastrzebski, "Cramér-Wold autoencoder," *arXiv 1805.09235*, 2018.

[57] J. Zhang, S. A. Bargal, Z. Lin, J. Brandt, X. Shen, and S. Sclaroff, "Top-down neural attention by excitation backprop," *International Journal of Computer Vision*, vol. 126, no. 10, pp. 1084–1102, 2018.

[58] L. Itti and C. Koch, "A saliency-based search mechanism for overt and covert shifts of visual attention," *Vision Research*, vol. 40, no. 10-12, pp. 1489–1506, 2000.

[59] J. K. Tsotsos, M. P. Eckstein, and M. S. Landy, "Computational models of visual attention," *Vision Research*, vol. 116, no. B, pp. 93–94, 2015.

[60] M. C. Avery, N. Dutt, and J. L. Krichmar, "Mechanisms underlying the basal forebrain enhancement of top-down and bottom-up attention," *European Journal of Neuroscience*, vol. 39, no. 5, pp. 852–865, 2014.

[61] M. Baxter and A. Chiba, "Cognitive functions of the basal forebrain," *Current Opinion in Neurobiology*, vol. 9, no. 2, pp. 178–183, 1999.

[62] N. Oros, A. A. Chiba, D. A. Nitz, and J. L. Krichmar, "Learning to ignore: a modeling study of a decremental cholinergic pathway and its influence on attention and learning," *Learning & Memory*, vol. 21, no. 2, pp. 105–118, 2014.

[63] A. J. Yu and P. Dayan, "Uncertainty, neuromodulation, and attention," *Neuron*, vol. 46, no. 4, pp. 681–692, 2005.

[64] S. Bouret and S. Sara, "Network reset: A simplified overarching theory of locus coeruleus noradrenaline function," *Trends in Neuroscience*, vol. 28, no. 11, pp. 574–582, 2005.

[65] S. Grella, J. Neil, H. Edison, V. Strong, I. Odintsova, S. Walling, G. Martin, D. Marrone, and C. Harley, "Locus coeruleus phasic, but not tonic, activation initiates global remapping in a familiar environment," *Journal of Neuroscience*, vol. 39, no. 3, pp. 445–455, 2019.

[66] J. Schmidhuber, "On learning to think: Algorithmic information theory for novel combinations of reinforcement learning controllers and recurrent neural world models," *arXiv 1511.09249*, 2015.

[67] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Advances in Neural Information Processing Systems*, pp. 2455–2467, 2018.

[68] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *International Conference on Machine Learning*, pp. 3987–3995, 2017.

[69] J. L. McClelland, B. L. McNaughton, and R. C. O'Reilly, "Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory," *Psychological Review*, vol. 102, no. 3, p. 419, 1995.

[70] R. C. O'Reilly, R. Bhattacharyya, M. D. Howard, and N. Ketz, "Complementary learning systems," *Cognitive Science*, vol. 38, no. 6, pp. 1229–1248, 2014.

[71] A. Robins, "Catastrophic forgetting, rehearsal and pseudorehearsal," *Connection Science*, vol. 7, no. 2, pp. 123–146, 1995.

[72] B. Ans, S. Rousset, R. M. French, and S. Musca, "Self-refreshing memory in artificial neural networks: Learning temporal sequences without catastrophic forgetting," *Connection Science*, vol. 16, no. 2, pp. 71–99, 2004.

[73] D. Kumaran, D. Hassabis, and J. L. McClelland, "What learning systems do intelligent agents need? Complementary learning systems theory updated," *Trends in Cognitive Sciences*, vol. 20, no. 7, pp. 512–534, 2016.

[74] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," in *Advances in Neural Information Processing Systems*, pp. 2990–2999, 2017.

[75] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

[76] N. Kamra, U. Gupta, and Y. Liu, "Deep generative dual memory network for continual learning," *arXiv 1710.10368,*, 2017.

[77] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv 1503.02531*, 2015.

[78] C. Atkinson, B. McCane, L. Szymanski, and A. Robins, "Achieving deep reinforcement learning without catastrophic forgetting," *arXiv 1812.02464*, 2018.

[79] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," *arXiv 1811.04551*, 2018.

[80] J. Schmidhuber, *Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-... hook*. PhD thesis, Institut für Informatik, Technische Universität München, 1987.

[81] S. Thrun and L. Pratt, *Learning to Learn: Introduction and Overview*, pp. 3–17. Kluwer Academic Publishers, 1998.

[82] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the International Conference on Machine Learning*, pp. 1126–1135, 2017.

[83] C. Fernando, J. Sygnowski, S. Osindero, J. Wang, T. Schaul, D. Teplyashin, P. Sprechmann, A. Pritzel, and A. Rusu, "Meta-learning by the Baldwin effect," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018.

[84] D. Grbic and S. Risi, "Towards continual reinforcement learning through evolutionary meta-learning," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 119–120, 2019.

[85] K. Kansky, T. Silver, D. A. Mély, M. Eldawy, M. Lázaro-Gredilla, X. Lou, N. Dorfman, S. Sidor, S. Phoenix, and D. George, "Schema networks: Zero-shot transfer with a generative causal model of intuitive physics," *arXiv 1706.04317*, 2017.

[86] X. Li and R. Miikkulainen, "Evolving adaptive poker players for effective opponent exploitation," in *AAAI Conference on Artificial Intelligence Workshop*, 2017.

[87] X. Li and R. Miikkulainen, "Opponent modeling and exploitation in poker using evolved recurrent neural networks," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018.

[88] A. Rusu, N. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv 1606.04671*, 2016.

[89] T. Draelos, N. Miner, C. Lamb, J. Cox, C. Vineyard, K. Carlson, W. Severa, C. James, and J. Aimone, "Neurogenesis deep learning: Extending deep networks to accommodate new classes," *International Joint Conference on Neural Networks*, pp. 526–533, 2017.

[90] J. Yoon, E. Yang, J. Lee, and S. Hwang, "Lifelong learning with dynamically expandable networks," *arXiv 1708.01547*, 2017.

[91] N. Justesen, P. Bontrager, J. Togelius, and S. Risi, "Deep learning for video game playing," *IEEE Transactions on Games*, 2019.

[92] Y. Li, "Deep reinforcement learning: An overview," *arXiv 1701.07274*, 2017.

[93] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[94] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra, "Benchmarking reinforcement learning algorithms on real-world robots," in *Proceedings of the Conference on Robot Learning*, pp. 561–591, 2018.

[95] J. Gauci, E. Conti, Y. Liang, K. Virochsiri, Y. He, Z. Kaden, V. Narayanan, X. Ye, Z. Chen, and S. Fujimoto, "Horizon: Facebook's open source applied reinforcement learning platform," *arXiv 1811.00260*, 2018.

[96] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[97] A. Ray, J. Achiam, and D. Amodei, "Benchmarking safe exploration in deep reinforcement learning," *arXiv 1910.01708*, 2019.

[98] C. L. Wainwright and P. Eckersley, "Safelife 1.0: Exploring side effects in complex environments," *arXiv 1912.01217*, 2019.

[99] P. A. Ortega, V. Maini, *et al.*, "Building safe artificial intelligence: Specification, robustness, and assurance," *DeepMind Safety Research Blog*, 2018.

[100] S. Hoehl, K. Hellmer, M. Johansson, and G. Gredebäck, "Itsy bitsy spider…: Infants react with increased arousal to spiders and snakes," *Frontiers in Psychology*, vol. 8, p. 1710, 2017.

[101] D. M. Ferrero, J. K. Lemon, D. Fluegge, S. L. Pashkovski, W. J. Korzan, S. R. Datta, M. Spehr, M. Fendt, and S. D. Liberles, "Detection and avoidance of a carnivore odor by prey," *Proceedings of the National Academy of Sciences*, vol. 108, no. 27, pp. 11235–11240, 2011.

[102] D. Grbic and S. Risi, "Safe reinforcement learning through meta-learned instincts," in *Proceedings of the Conference on Artificial Life*, pp. 283–291, 2020.

[103] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, pp. 400–407, 1951.

[104] J. Kiefer, J. Wolfowitz, *et al.*, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952.

[105] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv 1412.6980*, 2014.

[106] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv 1707.06347*, 2017.

[107] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *arXiv 1502.05477*, 2015.

[108] M. Pecka and T. Svoboda, "Safe exploration techniques for reinforcement learning–an overview," in *International Workshop on Modelling and Simulation for Autonomous Systems*, pp. 357–375, 2014.

[109] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[110] E. Altman, *Constrained Markov decision processes*, vol. 7. CRC Press, 1999.

[111] M. Wen and U. Topcu, "Constrained cross-entropy method for safe reinforcement learning," in *Advances in Neural Information Processing Systems*, pp. 7450–7460, 2018.

[112] Z. C. Lipton, A. Kumar, J. Gao, L. Li, and L. Deng, "Combating deep reinforcement learning's sisyphean curse with reinforcement learning," *arXiv 1611.01211*, 2016.

[113] K. Srinivasan, B. Eysenbach, S. Ha, J. Tan, and C. Finn, "Learning to be safe: Deep RL with a safety critic," *arXiv 2010.14603*, 2020.

[114] H. Bharadhwaj, A. Kumar, N. Rhinehart, S. Levine, F. Shkurti, and A. Garg, "Conservative safety critics for exploration," *arXiv 2010.14497*, 2020.

[115] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[116] L. Z. Yuan, M. Hasanbeig, A. Abate, and D. Kroening, "Modular deep reinforcement learning with temporal logic specifications," *arXiv 1909.11591*, 2019.

[117] J. Fan and W. Li, "Safety-guided deep reinforcement learning via online Gaussian process estimation," *arXiv 1903.02526*, 2019.

[118] Z. Kenton, A. Filos, O. Evans, and Y. Gal, "Generalizing from a few environments in safety-critical reinforcement learning," *arXiv 1907.01475*, 2019.

[119] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," *arXiv 1611.05763*, 2016.

[120] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "RL^2: Fast reinforcement learning via slow reinforcement learning," *arXiv 1611.02779*, 2016.

[121] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-reinforcement learning of structured exploration strategies," in *Proceedings of the International Conference on Neural Information Processing Systems*, pp. 5307–5316, 2018.

[122] L. Zintgraf, K. Shiarli, V. Kurin, K. Hofmann, and S. Whiteson, "Fast context adaptation via meta-learning," in *International Conference on Machine Learning*, pp. 7693–7702, 2019.

[123] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *Proceedings of the International Conference on Machine Learning*, pp. 5331–5340, 2019.

[124] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner," in *International Conference on Learning Representations*, 2018.

[125] K. Doya, "Metalearning and neuromodulation," *Neural networks*, vol. 15, no. 4-6, pp. 495–506, 2002.

[126] A. Soltoggio, J. A. Bullinaria, C. Mattiussi, P. Dürr, and D. Floreano, "Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios," in *Proceedings of the International Conference on Artificial Life*, pp. 569–576, 2008.

[127] S. Beaulieu, L. Frati, T. Miconi, J. Lehman, K. O. Stanley, J. Clune, and N. Cheney, "Learning to continually learn," *arXiv 2002.09571*, 2020.

[128] J. Xing, X. Zou, and J. L. Krichmar, "Neuromodulated patience for robot and self-driving vehicle navigation," in *Proceedings of the International Joint Conference on Neural Networks*, pp. 1–8, 2020.

[129] S. Bengio, Y. Bengio, J. Cloutier, and J. Gecsei, "On the optimization of a synaptic learning rule," in *Optimality in Artificial and Biological Neural Networks?*, pp. 6–8, Routledge, 1992.

[130] J. Schmidhuber, J. Zhao, and M. Wiering, "Simple principles of metalearning," *Technical Report IDSIA*, vol. 69, pp. 1–23, 1996.

[131] N. Schweighofer and K. Doya, "Meta-learning in reinforcement learning," *Neural Networks*, vol. 16, no. 1, pp. 5–9, 2003.

[132] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-SGD: Learning to learn quickly for few-shot learning," *arXiv 1707.09835*, 2017.

[133] B. C. Stadie, G. Yang, R. Houthooft, X. Chen, Y. Duan, Y. Wu, P. Abbeel, and I. Sutskever, "Some considerations on learning to explore via meta-reinforcement learning," *arXiv 1803.01118*, 2018.

[134] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel, "ProMP: Proximal meta-policy search," in *International Conference on Learning Representations*, 2019.

[135] E. Z. Liu, A. Raghunathan, P. Liang, and C. Finn, "Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices," in *International Conference on Machine Learning*, pp. 6925–6935, 2021.

[136] L. Zintgraf, K. Shiarlis, M. Igl, S. Schulze, Y. Gal, K. Hofmann, and S. Whiteson, "VariBAD: A very good method for Bayes-adaptive deep rl via meta-learning," in *International Conference on Learning Representations*, 2020.

[137] J. Humplik, A. Galashov, L. Hasenclever, P. A. Ortega, Y. W. Teh, and N. Heess, "Meta reinforcement learning as task inference," *arXiv 1905.06424*, 2019.

[138] E. Marder, "Neuromodulation of neuronal circuits: back to the future," *Neuron*, vol. 76, no. 1, pp. 1–11, 2012.

[139] M. Bear, B. Connors, and M. A. Paradiso, *Neuroscience: Exploring the brain*. Jones & Bartlett Learning, LLC, 2020.

[140] M. C. Avery and J. L. Krichmar, "Neuromodulatory systems and their interactions: A review of models, theories, and experiments," *Frontiers in Neural Circuits*, vol. 11, no. 108, pp. 1662–5110, 2017.

[141] A. Soltoggio, P. Durr, C. Mattiussi, and D. Floreano, "Evolving neuromodulatory topologies for reinforcement learning-like problems," in *IEEE Congress on Evolutionary Computation*, pp. 2471–2478, 2007.

[142] R. Velez and J. Clune, "Diffusion-based neuromodulation can eliminate catastrophic forgetting in simple neural networks," *PloS One*, vol. 12, no. 11, p. e0187736, 2017.

[143] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[144] R. Kempter, W. Gerstner, and J. L. Van Hemmen, "Hebbian learning and spiking neurons," *Physical Review E*, vol. 59, no. 4, p. 4498, 1999.

[145] K. D. Miller and D. J. MacKay, "The role of constraints in Hebbian learning," *Neural Computation*, vol. 6, no. 1, pp. 100–126, 1994.

[146] A. Soltoggio and K. O. Stanley, "From modulated Hebbian plasticity to simple behavior learning through noise and weight saturation," *Neural Networks*, vol. 34, pp. 28–41, 2012.

[147] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[148] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," *Proceedings of the International Conference on Machine Learning*, pp. 807–814, 2010.

[149] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," *Neural Information Processing Systems Workshop*, 2017.

[150] S. Greydanus, "Excitation BP: Visualizing how deep networks make decisions," *https://github.com/greydanus/excitationbp*, 2018.

[151] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv 1408.5093*, 2014.

[152] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in COntext," in *European Conference on Computer Vision*, pp. 740–755, 2014.

[153] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

[154] T. Yamamoto, T. Nishino, H. Kajima, M. Ohta, and K. Ikeda, "Human Support Robot (HSR)," in *ACM SIGGRAPH Emerging Technologies*, p. 11, 2018.

[155] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proceedings of the International Conference on Machine Learning*, 2014.

[156] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[157] C. M. Bishop, "Mixture density networks," tech. rep., Aston University, Birmingham, 1994.

[158] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[159] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv 1606.01540*, 2016.

[160] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, *et al.*, "Model-based reinforcement learning for Atari," *arXiv 1903.00374*, 2019.

[161] Wikipedia, "Flappy bird." https://en.wikipedia.org/wiki/Flappy_Bird, 2020. Accessed 3-February-2020).

[162] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[163] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 1st ed., 2017.

[164] J. D. Knowles, R. A. Watson, and D. W. Corne, "Reducing local optima in single-objective problems by multi-objectivization," in *Evolutionary Multi-Criterion Optimization* (E. Zitzler, L. Thiele, K. Deb, C. A. Coello Coello, and D. Corne, eds.), pp. 269–283, Springer Berlin Heidelberg, 2001.

[165] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[166] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, 2012.

[167] V. Vassiliades, K. Chatzilygeroudis, and J.-B. Mouret, "Using centroidal Voronoi tessellations to scale up the multi-dimensional archive of phenotypic elites algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 4, pp. 623–630, 2017.

[168] K. Chatzilygeroudis, V. Vassiliades, and J.-B. Mouret, "Reset-free trial-and-error learning for robot damage recovery," *Robotics and Autonomous Systems*, vol. 100, pp. 236–250, 2018.

[169] A. Cully and J.-B. Mouret, "Evolving a behavioral repertoire for a walking robot," *Evolutionary Computation*, vol. 24, no. 1, pp. 59–88, 2016.

[170] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, pp. 503–507, 2015.

[171] D. Ha and J. Schmidhuber, "World Models," *arXiv 1803.10122*, 2018.

[172] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv 1504.04909,*, 2015.

[173] V. Vassiliades and J.-B. Mouret, "Discovering the elite hypervolume by leveraging interspecies correlation," *Genetic and Evolutionary Computation Conference*, 2018.

[174] P. Larrañaga and J. Lozano, "Estimation of distribution algorithms: A new tool for evolutionary computation. Vol. 2," *Springer Science & Business Media*, 2001.

[175] M. Hoffman, D. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1303–1347, 2013.

[176] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994.

[177] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*, pp. 1008–1014, 2000.

[178] J. Peters, S. Vijayakumar, and S. Schaal, "Natural actor-critic," in *European Conference on Machine Learning*, pp. 280–291, Springer, 2005.

[179] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," *arXiv 1708.05144*, 2017.

[180] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015.

[181] I. Kostrikov, "PyTorch implementations of reinforcement learning algorithms." https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail, 2018.

[182] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning*, pp. 1861–1870, 2018.

[183] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3234–3243, 2016.

[184] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 234–241, 2015.

[185] K. H. Zou, S. K. Warfield, A. Bharatha, C. M. Tempany, M. R. Kaus, S. J. Haker, W. M. Wells III, F. A. Jolesz, and R. Kikinis, "Statistical validation of image segmentation quality based on a spatial overlap index: Scientific reports," *Academic Radiology*, vol. 11, no. 2, pp. 178–189, 2004.

[186] A. Krizhevsky, V. Nair, and G. Hinton, "CIFAR-10 and CIFAR-100 datasets," *URL: https://www.cs.toronto.edu/kriz/cifar.html*, 2009.

[187] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv 1605.07146*, 2016.

[188] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *arXiv 1708.04552*, 2017.

[189] L. Zaborszky, "The modular organization of brain systems. Basal forebrain: The last frontier," *Progress in Brain Research*, vol. 136, pp. 359–372, 2002.

[190] D. R. Wozny, U. R. Beierholm, and L. Shams, "Probability matching as a computational strategy used in perception," *PLoS Computational Biology*, vol. 6, no. 8, 2010.

[191] A. B. Craig, M. E. Phillips, A. Zaldivar, R. Bhattacharyya, and J. L. Krichmar, "Investigation of biases and compensatory strategies using a probabilistic variant of the wisconsin card sorting test," *Frontiers in Psychology*, vol. 7, no. 17, 2016.

[192] J. Naude, S. Tolu, M. Dongelmans, N. Torquet, S. Valverde, G. Rodriguez, S. Pons, U. Maskos, A. Mourot, F. Marti, and P. Faure., "Nicotinic receptors in the ventral tegmental area promote uncertainty-seeking," *Nature Neuroscience*, vol. 19, no. 3, pp. 471–478, 2016.

[193] F. Baluch and L. Itti, "Mechanisms of top-down attention," *Trends in Neurosciences*, vol. 34, no. 4, pp. 210–224, 2011.

[194] J. Tanner and L. Itti, "Goal relevance as a quantitative model of human task relevance," *Psychological Review*, vol. 124, p. 168, Mar 2017.

[195] J. Tanner and L. Itti, "A top-down saliency model with goal relevance," *Journal of Vision*, vol. 19, pp. 1–16, Jan 2019.

[196] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2921–2929, 2016.

[197] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 618–626, 2017.

[198] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, "Large-scale study of curiosity-driven learning," *arXiv 1808.04355*, 2018.

[199] J. Achiam and S. Sastry, "Surprise-based intrinsic motivation for deep reinforcement learning," *arXiv 1703.01732*, 2017.

[200] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 16–17, 2017.

[201] C. Cao, X. Liu, Y. Yang, Y. Yu, J. Wang, Z. Wang, Y. Huang, L. Wang, C. Huang, W. Xu, *et al.*, "Look and think twice: Capturing top-down visual attention with feedback convolutional neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2956–2964, 2015.

[202] K. Cho, A. Courville, and Y. Bengio, "Describing multimedia content using attention-based encoder-decoder networks," *IEEE Transactions on Multimedia*, vol. 17, no. 11, pp. 1875–1886, 2015.

[203] L. McInnes, J. Healy, and J. Melville, "UMAP: Uniform Manifold Approximation and Projection for dimension reduction," *arXiv 1802.03426*, 2018.

[204] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[205] D. Lopez-Paz *et al.*, "Gradient episodic memory for continual learning," in *Advances in Neural Information Processing Systems*, pp. 6467–6476, 2017.

[206] C. M. Alberini and J. E. LeDoux, "Memory reconsolidation," *Current Biology*, vol. 23, no. 17, pp. R746–R750, 2013.

[207] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *arXiv 1803.02999*, 2018.

[208] R. Miikkulainen, "Subsymbolic case-role analysis of sentences with embedded clauses," *Cognitive Science*, vol. 20, pp. 47–73, 1996.

[209] N. Hansen, S. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003.

[210] E. Coumans and Y. Bai, "PyBullet, a Python module for physics simulation for games, robotics and machine learning," *GitHub Repository*, 2016.

[211] K. Chatzilygeroudis, V. Vassiliades, and J.-B. Mouret, "Reset-free trial-and-error learning for robot damage recovery," *Robotics and Autonomous Systems*, vol. 100, pp. 236–250, 2018.

[212] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," *GitHub Repository*, 2018.

[213] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

[214] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Conference on Robot Learning*, pp. 1094–1100, 2020.

[215] E. Ben-Iwhiwhu, P. Ladosz, J. Dick, W.-H. Chen, P. Pilly, and A. Soltoggio, "Evolving inborn knowledge for fast adaptation in dynamic POMDP problems," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 280–288, 2020.

[216] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton, "Similarity of neural network representations revisited," in *Proceedings of the International Conference on Machine Learning*, pp. 3519–3529, 2019.

[217] A. Morcos, M. Raghu, and S. Bengio, "Insights on representational similarity in neural networks with canonical correlation," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[218] N. Kriegeskorte, M. Mur, and P. A. Bandettini, "Representational similarity analysis-connecting the branches of systems neuroscience," *Frontiers in Systems Neuroscience*, vol. 2, p. 4, 2008.

[219] A. Gretton, O. Bousquet, A. Smola, and B. Schölkopf, "Measuring statistical dependence with Hilbert-Schmidt norms," in *International Conference on Algorithmic Learning Theory*, pp. 63–77, 2005.

[220] T. Goerttler and K. Obermayer, "Exploring the similarity of representations in model-agnostic meta-learning," in *Proceedings of the International Conference on Machine Learning Workshop*, 2021.

[221] A. Raghu, M. Raghu, S. Bengio, and O. Vinyals, "Rapid learning or feature reuse? Towards understanding the effectiveness of MAML," in *Proceedings of the International Conference on Learning Representations*, 2020.

[222] M. Corbetta, G. Patel, and G. L. Shulman, "The reorienting system of the human brain: From environment to theory of mind," *Neuron*, vol. 58, no. 3, pp. 306–324, 2008.

[223] M. Sarter, M. E. Hasselmo, J. P. Bruno, and B. Givens, "Unraveling the attentional functions of cortical cholinergic inputs: Interactions between signal-driven and cognitive modulation of signal detection," *Brain Research Reviews*, vol. 48, no. 1, pp. 98–111, 2005.

[224] M. Goard and Y. Dan, "Basal forebrain activation enhances cortical coding of natural scenes," *Nature Neuroscience*, vol. 12, no. 11, p. 1444, 2009.

[225] J. L. Herrero, M. Roberts, L. S. Delicato, M. A. Gieselmann, P. Dayan, and A. Thiele, "Acetylcholine contributes through muscarinic receptors to attentional modulation in V1," *Nature*, vol. 454, no. 7208, p. 1110, 2008.

[226] M. A. Atkinson, A. A. Simpson, and G. G. Cole, "Visual attention and action: How cueing, direct mapping, and social interactions drive orienting," *Psychonomic Bulletin & Review*, vol. 25, no. 5, pp. 1585–1605, 2018.

[227] A. Mahon, S. Bendžiūtė, C. Hesse, and A. R. Hunt, "Shared attention for action selection and action monitoring in goal-directed reaching," *Psychological Research*, pp. 1–14, 2018.

[228] M. Henaff, W. F. Whitney, and Y. LeCun, "Model-based planning with discrete and continuous actions," *arXiv 1705.07177*, 2017.

[229] M. Rostami, S. Kolouri, and P. K. Pilly, "Complementary learning for overcoming catastrophic forgetting using experience replay," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2019.

[230] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[231] J. Schrum and R. Miikkulainen, "Evolving multimodal behavior with modular neural networks in Ms. Pac-Man," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 325–332, 2014.

[232] S. Narvekar and P. Stone, "Learning curriculum policies for reinforcement learning," *arXiv 1812.00285*, 2018.

[233] R. Wang, J. Lehman, J. Clune, and K. O. Stanley, "Paired Open-Ended Trailblazer (POET): Endlessly generating increasingly complex and diverse learning environments and their solutions," *arXiv 1901.01753*, 2019.

[234] J. Schmidhuber, "POWERPLAY: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem," *arXiv 1112.5309*, 2011.

[235] N. Justesen and S. Risi, "Automated curriculum learning by rewarding temporally rare events," 2018.

[236] S. Risi and J. Togelius, "Procedural Content Generation: From automatically generating game levels to increasing generality in machine learning," *arXiv 1911.13071*, 2019.

[237] G. Parisi, R. Kemker, J. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, 2019.

[238] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[239] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv 1409.1259*, 2014.

[240] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[241] J. X. Wang, M. King, N. Porcel, Z. Kurth-Nelson, T. Zhu, C. Deck, P. Choy, M. Cassin, M. Reynolds, F. Song, *et al.*, "Alchemy: A structured task distribution for meta-reinforcement learning," *arXiv 2102.02926*, 2021.

[242] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning.," *IEEE International Conference on Robotics and Automation*, pp. 4693–4700, 2018.

4. Publications and Presentations

Journal Publications

Pilly, P. K., Howard, M. D., and Bhattacharyya, R. (2018). Modeling contextual modulation of memory associations in the hippocampus. *Frontiers in Human Neuroscience*, 12, 442.

Beyeler, M., Rounds, E. L., Carlson, K. D., Dutt, N., and Krichmar, J. L. (2019). Neural correlates of sparse coding and dimensionality reduction. *PLoS Computational Biology*, 15(6), e1006908.

Zou, X., Kolouri, K., Pilly, P. K., and Krichmar, J. L. (2020). Neuromodulated attention and goal-driven perception in uncertain domains. *Neural Networks*, 125, 56-69.

Dick, J., Ladosz, P., Ben-Iwhiwhu, E., Shimadzu, H., Kinnell, P., Pilly, P. K., Kolouri, S., and Soltoggio, A. (2020). Detecting changes and avoiding catastrophic forgetting in dynamic partially observable environments. *Frontiers in Neurorobotics*, 14, 578675.

McClelland, J. L., McNaughton, B. L., and Lampinen, A. K. (2020). Integration of new information in memory: New insights from a complementary learning systems perspective. *Philosophical Transactions of the Royal Society B*, 375(1799), 20190637.

Belkaid, J. L. and Krichmar, J. L. (2020). Modeling uncertainty-seeking behavior mediated by cholinergic influence on d opamine. *Neural Networks*, 125, 10-18.

Hwu, T. and Krichmar, J. L. (2020). A neural model of schemas and memory encoding. *Biological Cybernetics*, 114(2), 169-186.

Chiba, A. A. and Krichmar, J. L. (2020). Neurobiologically inspired self-monitoring systems. *Proceedings of the IEEE*, 108(7), 976-986.

Ladosz, P., Ben-Iwhiwhu, E., Dick, J., Ketz, N., Kolouri, S., Krichmar, J. L., Pilly, P. K., and Soltoggio, A. (2022). Deep reinforcement learning with modulated Hebbian plus Q-network architecture. *IEEE Transactions on Neural Networks and Learning Systems*, 33(5), 2045-2056.

Kudithipudi, D. et al. (2022). Biological underpinnings for lifelong learning machines. *Nature Machine Intelligence*, 4, 196-210.

Ben-Iwhiwhu, E., Dick, J. Ketz, N. A., Pilly, P. K., and Soltoggio, A. (2022). Context meta-reinforcement learning via neuromodulation. *Neural Networks*, 152, 70-79.

Conference Publications

Rostami, M., Kolouri, S., and Pilly, P. K. (2019). Complementary learning for overcoming catastrophic forgetting using experience replay. *Proceedings of the International Joint Conference on Artificial Intelligence, Macao, China*.

Martin, C. E. and Pilly, P. K. (2019). Probabilistic program neurogenesis. *Proceedings of the*

Conference on Artificial Life, Newcastle-upon-Tyne, United Kingdom.

Grbic, D. and Risi, S. (2019). Towards continual reinforcement learning through evolutionary meta-learning. *Proceedings of the Genetic and Evolutionary Computation Conference, Prague, Czech Republic*.

Kolouri, S., Nadjahi, K., Simsekli, U., Badeau, R., and Rohde, G. K. (2019). Generalized sliced Wasserstein distances. *Proceedings of the Neural and Information Processing Systems, Vancouver, Canada*.

Rostami, M., Kolouri, S., McClelland, J., and Pilly, P. K. (2020). Generative continual concept learning. *Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY*.

Kolouri, S., Ketz, N. A., Soltoggio, A., and Pilly, P. K. (2020). Sliced Cramer synaptic consolidation for preserving deeply learned representations. *Proceedings of the International Conference on Learning Representations, Addis Ababa, Ethiopia*.

Ben-Iwhiwhu, E., Ladosz, P., Dick, J., Chen, W.-H., Pilly, P., and Soltoggio, A. (2020). Evolving inborn knowledge for fast adaptation in dynamic POMDP problems. *Proceedings of the Genetic and Evolutionary Computation Conference, Cancun, Mexico*.

Grbic, D. and Risi, S. (2020) Safe reinforcement learning through meta-learned instincts. *Proceedings of the Conference on Artificial Life, Montreal, Canada*.

Carvelli, C., Grbic, D., and Risi, S. (2020). Evolving hypernetworks for game-playing agents. *Proceedings of the Genetic and Evolutionary Computation Conference, Cancun, Mexico*.

Xing, J., Zou, X., and Krichmar, J. L. (2020). Neuromodulated patience for robot and self-driving vehicle navigation. *Proceedings of the International Joint Conference on Neural Networks, Glasgow, United Kingdom*.

Singh, A. and McClelland, J. L. (2020). Human-like learning framework for frequency-skewed multi-level classification. *Proceedings of the Annual Meeting of Cognitive Science Society, Toronto, Canada*.

Tutum, C. C. and Miikkulainen, R. (2020). Generalization through context. *Proceedings of the Genetic and Evolutionary Computation Conference, Cancun, Mexico*.

Maguire, G. and Mouret, J. B. (2020). Quality diversity for multi-task optimization. *Proceedings of the Genetic and Evolutionary Computation Conference, Cancun, Mexico*.

Xing, J., Zou, X., and Krichmar, J. L. (2020). Neuromodulated patience for robot and self-driving vehicle navigation. *Proceedings of the International Joint Conference on Neural Networks, Glasgow, United Kingdom*.

Hwu, T. J., Kashyap, H., and Krichmar, J. L. (2020). Applying a neurobiological model of schemas and memory consolidation to contextual awareness in robotics. *Proceedings of the International Joint Conference on Neural Networks, Glasgow, United Kingdom*.

Li. H., Krishnan, A., Wu, J., Kolouri, S., Pilly, P. K., and Braverman, V. (2021). Lifelong learning with sketched structural regularization. *Proceedings of the Asian Conference on Machine Learning, Virtual*.

Xing, J., et al. (2021). Domain adaptation in reinforcement learning via latent unified state representation. *Proceedings of the AAAI Conference on Artificial Intelligence, Virtual*.

Tutum, C., Abdulquddos, S., and Miikkulainen, R. (2021). Generalization of agent behavior through explicit representation of context". *Proceedings of the IEEE Conference on Games*.

Grbic, D. and Risi, S. (2021). Safer reinforcement learning through transferable instinct networks. *Proceedings of the Conference on Artificial Life, Virtual*.

Workshop Publication

Li., H., Krishnan, A., Wu, J., Kolouri, S., Pilly, P. K., and Braverman, V. (2021). Lifelong learning with sketched structural regularization. *Proceedings of the Workshop on Theory and Foundation of Continual Learning at the 2021 International Conference on Machine Learning, Virtual*.

Unpublished Preprints

Kolouri, S., Ketz, N., Zou, X., Krichmar, J., and Pilly, P. K. (2019). Attention-based structural plasticity. *arXiv* 1903.06070.

Ketz, N., Kolouri, S., and Pilly, P. K. (2019). Continual learning using world models for pseudo-rehearsal. *arXiv* 1903.02647.

Howard, M. D., Skorheim, S., and Pilly, P. K. (2019). A model of bidirectional interactions between complementary learning systems for memory consolidation of sequential experiences. *bioRxiv*. https://doi.org/10.1101/2019.12.19.882035.

Warner, J., Devaraj, A., and Miikkulainen, R. (2020). Using context to make gas classifiers robust to sensor drift. *arXiv* 2003.07292v2.

Krichmar, J. L., Ketz, N. A., Pilly, P. K., and Soltoggio, A. (2021). Flexible path planning through vicarious trial and error. *bioRxiv*. https://doi.org/10.1101/2021.09.08.459317.

Maguire, G., Ketz, N., Pilly, P., and Mouret, J.-B. (2021). An online data-driven emergency-response method for autonomous agents in unforeseen situations. *arXiv* 2112.09670.

Presentations

Pilly, P. K. (2019). Lifelong learning machines. *2019 DARPA Electronics Resurgence Initiative Summit, Detroit, MI*. (Plenary session on "The Intersection of Automotive, Electronics, and Au-

tomation")

Pilly, P. K. (2021). Autonomous navigation and game play domains. *2021 DARPA Electronics Resurgence Initiative Summit, Virtual*. (Workshop on "New Opportunities for Lifelong Learning Machines")

5. Software Packages

MOdulated Hebbian Network
Repository: https://github.com/pladosz/MOHQA
Publication: https://ieeexplore.ieee.org/document/9547670

Context-Skill Model
Repository: https://github.com/nnrg/ContextSkillFlappyball
Publication: https://ieee-cog.org/2021/assets/papers/paper 29.pdf

Meta-Learned Instinct Network
Repository: https://github.com/djole/IR2L
Publication: https://direct.mit.edu/isal/proceedings/isal/33/107/102974

Plastic Neuromodulated Network
Repository: https://github.com/anon-6994/nm-metarl
Publication: https://www.sciencedirect.com/science/article/pii/S0893608022001368

Detecting changes and avoiding catastrophic forgetting in dynamic partially observable environments
Repository: https://github.com/JupiLogy/adaptive-model-detection
Publication: https://www.frontiersin.org/articles/10.3389/fnbot.2020.578675

Domain adaptation in reinforcement learning via latent unified state representation
Repository: https://github.com/KarlXing/LUSR
Publication: https://ojs.aaai.org/index.php/AAAI/article/view/17251/17058

Configurable Tree graph environment
Repository: https://github.com/soltoggio/CT-graph

6. Intellectual Property

Patents

Martin, C. E., Ketz, N. A., Pilly, P. K., Kolouri, S., Howard, M. D., and Stepp, N. D. (2021). Artificial neural network and method of training an artificial neural network with epigenetic neurogenesis. US Patent No. 11,113,597.

Kolouri, S., Ketz, N. A., Pilly, P. K., Martin, C. E., and Howard, M. D. (2021). Artificial neural networks having attention-based selective plasticity and methods of training the same. US Patent No. 11,210,559.

Pending Patent Applications

Ketz, N. A., Pilly, P. K., Kolouri, S., Martin, C. E., and Howard, M. D. (2019). Autonomous system including a continually learning World Model and related methods. US Patent Application No. 16,548,560.

Kolouri, S., Rostami, M., and Pilly, P. K. (2020). Systems and methods for unsupervised continual learning. US Patent Application No. 17,066,457.

Rostami, M., Kolouri, S., and Pilly, P. K. (2020). System and method for continual learning using experience replay. US Patent Application No. 16,875,852.

Yi, W., Martin, C., Kolouri, S., and Pilly, P. K. (2021). Neuromorphic memory circuit and method of neurogenesis for an artificial neural n etwork. US Provisional Patent Application N o. 63,185,830.

Stepp, N. D. and Pilly, P. K. (2022). Homological representation and recognition of reward-based tasks. HRL Invention Disclosure No. 81315179.

7. List of Symbols, Abbreviations, and Acronyms

2D	Two-Dimensional
a	Action
A2C	Advantage Actor Critic
A3C	Asynchronous Advantage Actor Critic
AI	Artificial Intelligence
ACER	Actor-Critic with Experience Replay
ACh	Cholinergic
ALE	Arcade Learning Environment
AMRL	Aggregated Memory for Reinforcement Learning
BCE	Binary Cross-Entropy
BO	Bayesian Optimization
C	Controller network
C	Context-only network
C^*	Saved copy of *C*
CAM	Class Activation Mapping
c-EB	contrastive Excitation Backpropagation
CARLA	Car Learning to Act (Autonomous driving domain)
CAVIA	Context Adaptation via Meta-Learning
CKA	Centered Kernel Alignment
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CNN	Convolutional Neural Network
COCO	Common Objects in COntext
CS	Context-Skill network
CSM	Context-Skill Model
CT-graph	Configurable Tree graph
CVT	Centroidal Voronoi Tessellation
d	Done state
DQN	Deep Q-Network
E-PNN	Evolving Plastic Neural Network
EB	Excitation Backpropagation
EDA	Estimation of Distribution Algorithm
EWC	Elastic Weight Consolidation
EX	Evaluation Experience
FB	Flappy Ball domain
FIM	Fisher Information Matrix
G	Number of Gaussian mixtures
GA	Genetic Algorithm
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
Grad-CAM	Gradient-weighted Class Activation Mapping

h	Hidden state of Long Short-Term Memory network
HSIC	Hilbert-Schmidt Independence Criterion
HSR	Human Support Robot
IR^{2}L	Instinct-Regulated Reinforcement Learning
JS	Jensen-Shannon
KL	Kullback–Leibler
KNN	K-Nearest Neighbors
$\mathscr{L}$	Loss term
L2	Lifelong Learning
L2M	Lifelong Learning Machines
LB	Learning Block
LC	Linear Classifier
LIDAR	LIght Detection And Ranging
LL	Lunar Lander domain
LSTM	Long Short-Term Memory
LX	Learning Experience
M	Long Short-Term Memory + Mixture Density Networks
M^*	Saved copy of M
MAP-Elites	Multi-dimensional Archive of Phenotypic Elites
MAS	Memory Aware Synapses
ML	Machine Learning
μ	Mean of a given Gaussian
MDN	Mixture Density Network
MDRNN	Mixture Density Recurrent Neural Network
MDP	Markov Decision Process
MLP	Multi-Layer Perception
MOHN	MOdulated Hebbian Network
MOHQA	MOdulated Hebbian plus Q-network Architecture
MNIST	Modified National Institute of Standards and Technology
MSE	Mean Squared Error
MTME	Multi-Task MAP Elites
MWP	Marginal Winning Probability
$\mathscr{N}$	Normal distribution
NE	Noradrenergic
NOOP	No Operation
NPN	Neuromodulated Policy Network
NSGA	Non-dominated Sorting Genetic Algorithm
PEARL	Probabilistic Embeddings for Actor-Critic Meta-RL
π	Policy or action distribution
Π	Gaussian mixture weights/probabilities
PNN	Plastic Neuromodulated Network
POMDP	Partially Observable Markov Decision Process

PPL	Probabilistic Program Learner
PPN	Probabilistic Program Neurogenesis
PPO	Proximal Policy Optimization
PSD	Positive Semi-Definite
PyFLANN	Python Fast Library for Approximate Nearest Neighbors
QD	Quality Diversity
r	Reward
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RePR	Reinforcement-Pseudo-Rehearsal
RGB-D	Red Green Blue-Depth
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RSA	Representation Similarity Analysis
S	Skill-only network
SCP	Sliced Cramer Preservation
SLB	Short Learning Block
σ	Standard deviation of a given Gaussian
SPN	Standard Policy Network
SSIM	Structural Similarity Index Metric
STDP	Spike-Timing-Dependent Plasticity
STE	Single Task Expert
STELLAR	Super Turing Evolving Lifelong Learning ARchitecture
SVHN	Street View House Numbers
T	Episode length
TD	Temporal Difference
UMAP	Uniform Manifold Approximation and Projection
V	Convolutional Variational Auto-Encoder
VAE	Variational Auto-Encoder
WTA	Winner-Take-All
z	Compressed state representation (output of V)