AFRL-RY-WP-TR-2019-0089

DEEP LEARNING ARCHITECTURES FOR ROBUST CLASSIFICATION UNDER ADVERSARIAL NOISE

Yaron Singer
Harvard University

AUGUST 2019
Final Report

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

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 nationals.

Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RY-WP-TR-2019-0089 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

DEMANGE.ASHLEY.M.1513253580 Digitally signed by DEMANGE.ASHLEY.M.1513253580 Date: 2019.08.01 11:09:28 -04'00'

ASHLEY M. DEMANGE
Program Manager
Sensor Subsystems Branch
Aerospace Components & Subsystems Division

JOHNSON.TIMOTHY.R.1067537501 Digitally signed by JOHNSON.TIMOTHY.R.1067537501 Date: 2019.08.02 10:45:14 -04'00'

TIMOTHY R. JOHNSON, Chief
Sensor Subsystems Branch
Aerospace Components & Subsystems Division

JUSTIN W. CLEARY
Deputy (Acting)
Aerospace Components & Subsystems Division
Sensors 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.

*Disseminated copies will show "//Signature//" stamped or typed above the signature

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
August 2019	Final	15 February 2018 – 15 February 2019

4. TITLE AND SUBTITLE
DEEP LEARNING ARCHITECTURES FOR ROBUST CLASSIFICATION UNDER ADVERSARIAL NOISE

5a. CONTRACT NUMBER
FA8650-18-1-7811

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
61101E

6. AUTHOR(S)
Yaron Singer

5d. PROJECT NUMBER
1000

5e. TASK NUMBER
N/A

5f. WORK UNIT NUMBER
Y1QT

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Harvard University
Office for Sponsored Programs
1033 Massachusetts Ave, 5th Floor
Cambridge, MA 02138-5369

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory
Sensors Directorate
Wright-Patterson Air Force Base, OH 45433-7320
Air Force Materiel Command
United States Air Force

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/RYDR

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-RY-WP-TR-2019-0089

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
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 material is based on research sponsored by Air Force Research laboratory (AFRL) and the Defense Advanced Research Agency (DARPA) under agreement number FA8650-18-1-7811. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation herein. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies of endorsements, either expressed or implied, of AFRL and DARPA or the U.S. Government. Report contains color.

14. ABSTRACT
This report focuses on the problem of designing robust classifiers to images that are distorted by noise. The approach taken was robust optimization where the goal was to optimize in the worst case over a class of objective functions. A theoretical framework with strong guarantees was developed. In particular it was shown that given a classifier that has α accuracy over a finite number of attacks, one can develop a robust classifier that is an arbitrarily close to be an α approximation to the optimal robust classifier. These results were applied to robust neural network training and approach was evaluated experimentally on corrupted character classification.

15. SUBJECT TERMS
deep neural networks, noise distortion, deep learning architectures

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. EPORT	b. ABSTRACT	c. THIS PAGE			Ashley DeMange
Unclassified	Unclassified	Unclassified	SAR	21	19b. TELEPHONE NUMBER *(Include Area Code)* N/A

Table of Contents

Section **Page**

List of Figures

Figure **Page**

List of Tables

Table **Page**

1. Overview

In recent years, classification using deep neural networks (DNNs) has produced state-of-the-art performance on visual classification problems, achieving near-human-level performance on image recognition tasks. Despite their empirical success, however, there are many open questions related to our theoretical and intuitive understanding of deep learning methods.

One important open question relates to the robustness of deep neural networks to noise. In a recent paper (https://arxiv.org/pdf/1312.6199.pdf), Szegedy et al. show that standard deep learning classifiers are extremely sensitive to noise. In particular, the authors show that one can take an image which is correctly classified by a DNN, and make the same image become completely misclassified by perturbing the pixels of the image in a manner that is not detectable to the human eye (see Figure 1). In a world which develops an ever-growing dependence on automatic classification using neural networks, with applications ranging from self-driving cars to face recognition, such sensitivity to noise can have dire consequences.

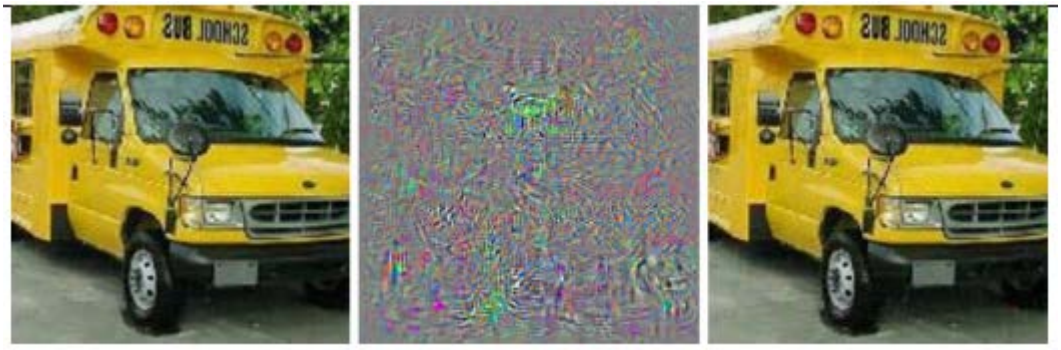

Figure 1: Example of Misclassification of Bus via Noise from Szegedy et al
The image on the left is a bus, correctly classified by a standard DNN; the figure in the middle depicts the perturbation of pixels added to the image on the left; the image on the right is the image of the bus after noise has been applied, where the DNN classified the bus as an ostrich.

Towards noise robust DNNs. Motivated by this problem, the goal of this project was to design deep learning methods that are *probably* robust to adversarial noise. This project combined modeling, mathematical, and experimental challenges. At a high level, the modeling challenge is that we are interested in providing provable guarantees for the methods we suggest. The problem however, is that training a deep learning classifier requires solving an intractable optimization problem. We used robust and combinatorial optimization and overcame various interesting theoretical challenges that arose when designing these algorithms. Most importantly, we developed algorithms that work well in *practice*. This required designing architectures and training large-scale DNNs.

1.1 Deliverables

The deliverables for this project are outlined below. We refer to the Modified National Institute of Standards and Technology (MNIST) handwritten digits data set containing 55000 training images.

Table 1. Outline of Deliverables

DESCRIPTION	APPROACH	CRITERIA	DELIVERABLES
Milestone 1: DNN architecture for every noise type	Design and train DNN on TensorFlow	Empirical test accuracy α on MNIST test data	Software of DNN classifier
Milestone 2: Algorithm for max-min robust guarantees	Robust optimization	Worst-case bounds on max-min accuracy	Report with details on max-min algorithm and analysis
Milestone 3: DNN architecture or oracle and boosting	Design and train DNN on TensorFlow	Test on corrupted MNIST data sets	Software of robust DNN classifier
Milestone 4: Classification from robust guarantees	Boosting framework	Worst-case bounds on classification accuracy	Report with details on classification algorithm and analysis
Milestone 5: Hybrid and Composite methods	Design and train DNN on TensorFlow	Test on corrupted MNIST data sets	Software of robust DNN classifier
Milestone 6: Experiments	Design and train DNN on TensorFlow	Test on corrupted MNIST data sets	Report with analysis of robust DNN on adversarial noise

2. Milestones

2.1 Milestone 1: DNN Architecture for every Noise Type

If we have access to the noise generator and train a classifier on noisy images, the classifier will classify images well. Thus, in this case, we can imagine that we have a set of noise types $N_1, \ldots, N_k$ and for each $i \in [k]$ we are able to find a set of weights $x_i \in \mathbb{R}^n$ s.t. the classification accuracy $f_i(x_i)$ is at least some α, where $\alpha \in [0, 1]$[1]. If we know which noise type the adversary uses we can trivially train a classifier on that noise type and obtain accuracy of α. The problem is that we do not know which noise type the adversary uses, and hence without constructing a robust classifier, even if an adversary chooses a noise type at random, our expected accuracy is α/k, which is poor since we think of k as a large number that tends to infinity.

[1]For example, for most noise distributions we tested, simple neural network architectures are able to produce solutions s.t. their accuracy is over 90%.

Training with Adversarial Noise
In general, we can imagine a space $X \in \mathbb{R}^n$ to optimize over, and a collection of k potential objective functions that might arise: $F = \{f_1, \ldots, f_k\}$. For each function $f_i \in F$ we assume that we do not know how to optimize f_i *exactly*, but rather know how to obtain a solution $x_i \in \mathbb{R}^n$ s.t. $(f(x_i) \geq \alpha f_i(x_i^*)$, where x_i^* is the optimal solution, i.e., $x_i^* \in \arg\max_x f_i(x)$. In our context, each f_i measures the *accuracy* that a DNN can achieve when trained with noise generator $i \in [k]$. Since we may not be able to train the DNN to have perfect or even optimal accuracy, we resort to this black-box model: if one assumes that the DNN can correctly classify at least α fraction of the inputs, our goal is to return a robust classifier whose accuracy is a function of α. Indeed, in this report we show how to design such a robust classifier whose accuracy is arbitrarily close to an α approximation of the optimal robust classifier. Observe that by using this black-box approach where our only assumption about the oracle is that it has α accuracy, we are able to circumvent the issue that there are no guarantees for training DNNs.

2.1.1 Corruption Set Details

Background Corruption Set consists of images with (i) an unperturbed white background–the original images, (ii) a light gray tint background, (iii) a gradient background, (iv) and a checkerboard background.
Shrink Corruption Set consists of images with (i) no distortion–the original images, (ii) a 25% shrinkage along the horizontal axis, (iii) a 25% shrinkage along the vertical axis, and (iv) a 25% shrinkage in both axes.
Pixel Corruption Set consists of images that (i) remain unaltered–the original images, (ii) have *Unif*[-0.15, -0.05] perturbation added i.i.d. to each pixel, (iii) have *Unif*[-0.05, 0.05] perturbation added i.i.d. to each pixel, and (iv) have *Unif*[0.05, 0.15] perturbation added i.i.d. to each pixel.
Mixed Corruption Set consists of images that (i) remain unaltered–the original images, and one corruption type from each of the previous three corruption sets (which were selected at random), namely that with (ii) the checkerboard background, (iii) 25% shrinkage in both axes, and (iv) i.i.d. *Unif*[-0.15, -0.05] perturbation.

2.1.2 Neural Network Results

We use the MNIST handwritten digits data set containing 55000 training images. Table 2 shows the individual bottleneck loss results. Mean is over 10 independent runs and a 95%confidence interval for the mean with T = 50 on all four Corruption Sets. Composite Method outperforms Hybrid Method, and both outperform baselines, with such differences being statistically significant.

Table 2. Individual Bottleneck Loss Results on all four Corruption Sets

	Background Set	*Shrink Set*	*Pixel Set*	*Mixed Set*
Best Individual Baseline	**8.85** *(8.38,9.32)*	**7.19** *(7.09,7.28)*	**1.82** *(1.81,1.82)*	**8.75** *(8.50,9.00)*
Even Split Baseline	**28.35** *(26.81,29.89)*	**11.54** *(11.25,11.83)*	**1.93** *(1.91,1.95)*	**9.92** *(9.78,10.06)*
Uniform Distribution Baseline	**2.06** *(2.05,2.08)*	**1.74** *(1.72,1.76)*	**1.30** *(1.30,1.31)*	**1.46** *(1.45,1.47)*
Hybrid Method	**1.38** *(1.37,1.39)*	**1.48** *(1.47,1.49)*	**1.29** *(1.28,1.30)*	**1.36** *(1.35,1.36)*
Composite Method	**1.31** *(1.30,1.31)*	**1.30** *(1.29,1.31)*	**1.25** *(1.24,1.25)*	**1.25** *(1.24,1.26)*

Code. In the submitted code we include a design of a DNN implemented in TensorFlow that for any noise type.

2.2 Milestone 2: Algorithm for Max-Min Robust Guarantees

Our goal is to find a solution x ∈ *X* that achieves good quality for all f_i. In particular, we seek the max-min optimal solution:

$$\mathbf{x}_{best} = \operatorname*{argmax}_{\mathbf{x}\in X} \min_{i\in[k]} f_i(\mathbf{x}) \tag{1}$$

We are given access to an oracle which computes an approximate optimum for a distribution over objectives. We call the oracle an α-*approximate distributional oracle*. Specifically, given any distribution *D* on *F*, the oracle *M*(*D*) computes an α-approximate solution *x** = *M*(*D*) to the distributional problem, i.e.:

$$E_{f\sim D}\left[f(M(D))\right] \geq \frac{1}{\alpha} \max_{x\in X} E_{f\sim D}\left[f(x)\right] \tag{2}$$

Given access to an α-approximate distributional oracle, we seek to find a distribution *P* over solutions *X* such that for any realization of *f* ∈ *F*, the expected value of *f* (in expectation over the randomized solutions in *P*) is an approximation to the optimal value τ.

We first show that, given access to an α-approximate stochastic oracle, it is possible to efficiently implement improper α-approximate robust optimization, subject to a vanishing additive loss term.

2.2.1 Theorem

Given access to an α-approximate stochastic oracle, Algorithm 1 with $\eta = \sqrt{\frac{\log(m)}{2T}}$ *computes a distribution P over solutions, defined as a uniform distribution over a*

Algorithm 1. Oracle Efficient Improper Robust Optimization

Input: Objectives $\mathcal{L} = \{L_1, \ldots, L_m\}$, Approximate stochastic oracle M, parameters T, η for each time step $t \in [T]$ do

Set

$$\mathbf{w}_t[i] \propto \exp\left\{\eta \sum_{\tau=1}^{t-1} L_i(x_\tau)\right\} \tag{3}$$

Set $x_t = M(w_t)$

end for

Output: the uniform distribution over $\{x_1, \ldots, x_T\}$

---

set $\{x_1, \ldots, x_T\}$*, so that*

$$\max_{i \in [m]} \mathsf{E}_{x \sim \mathcal{P}} \left[L_i(x)\right] \leq \alpha\tau + \sqrt{\frac{2\log(m)}{T}}. \tag{4}$$

Moreover, for any $\eta > 0$*, the distribution P computed by Algorithm 1 satisfies:*

$$\max_{i \in [m]} \mathsf{E}_{x \sim \mathcal{P}} \left[L_i(x)\right] \leq \alpha(1+\eta)\tau + \frac{\log(m)}{\eta T}. \tag{5}$$

Proof. We begin by establishing (4), and will show how to extend our analysis to yield (5) at the end of the proof. We can interpret Algorithm 1 in the following way: We define a zero-sum game between a learner and an adversary. The learner's action set is equal to X and the adversary's action set is equal to $[m]$. The loss of the learner when he picks $x \in X$ and the adversary picks $i \in [m]$ is defined as $L_i(x)$. The corresponding payoff of the adversary is $L_i(x)$.

We will run no-regret dynamics on this zero-sum game, where at every iteration $t = 1, \ldots, T$, the adversary will pick a distribution over functions and subsequently the learner picks a solution x_t. For simpler notation we will denote with w_t the probability density function on $[m]$ associated with the distribution of the adversary. That is, $w_t[i]$ is the probability of picking function $L_i \in \mathcal{L}$. The adversary picks a distribution w_t based on some arbitrary no-regret learning algorithm on the m actions in $\mathcal{L}$. For concreteness consider the case where the adversary picks a distribution based on the multiplicative weight updates algorithm, i.e.,

$$w_t[i] \propto \exp\left\{\sqrt{\frac{\log(m)}{2T}}\sum_{\tau=1}^{t-1} L_i(x_\tau)\right\}. \tag{6}$$

Subsequently the learner picks a solution x_t that is the output of the α-approximate stochastic oracle on the distribution selected by the adversary at time-step t. That is,

$$x_t = M\left(\mathbf{w}_t\right). \tag{7}$$

Write $\epsilon(T) = \sqrt{\frac{2\log(m)}{T}}$. By the guarantees of the no-regret algorithm for the adversary, we have that

$$\frac{1}{T}\sum_{t=1}^{T} \mathsf{E}_{I\sim\mathbf{w}_t}\left[L_I(x_t)\right] \geq \max_{i\in[m]} \frac{1}{T}\sum_{t=1}^{T} L_i(x_t) - \epsilon(T). \tag{8}$$

Combining the above with the guarantee of the stochastic oracle we have

$$\begin{aligned}
\tau = \min_{x\in\mathcal{X}} \max_{i\in[m]} L_i(x) \geq & \min_{x\in X} \frac{1}{T}\sum_{t=1}^{T} \mathsf{E}_{I\sim\mathbf{w}_t}\left[L_I(x)\right] \geq \frac{1}{T}\sum_{t=1}^{T} \min_{x\in X} \mathsf{E}_{I\sim\mathbf{w}_t}\left[L_I(x)\right] \\
& \geq \frac{1}{T}\sum_{t=1}^{T} \frac{1}{\alpha}\cdot \mathsf{E}_{I\sim\mathbf{w}_t}\left[L_I(x_t)\right] \qquad \text{(By oracle guarantee for each } t\text{)} \\
& \geq \frac{1}{\alpha}\cdot\left(\max_{i\in[m]} \frac{1}{T}\sum_{t=1}^{T} L_i(x_t) - \epsilon(T)\right). \qquad \text{(By no-regret of adversary)}
\end{aligned}$$

Thus, if we define with P to be the uniform distribution over $\{x_1,\ldots, x_T\}$, then we have derived

$$\max_{i\in[m]} \mathsf{E}_{x\sim\mathcal{P}}\left[L_i(x)\right] \leq \alpha\tau + \epsilon(T) \tag{9}$$

as required.

2.3 Milestone 3: DNN Architecture or Oracle and Boosting

In order to apply the robust optimization algorithm we need to construct a neural network architecture that facilitates it. In each iteration t, such an architecture receives a distribution over corruption types $w_t = [w_t[1], ..., w_t[m]]$ and produces a set of weights θ_t.

We take each training data image (Figure 2) and perturb it by exactly one corruption, with corruption i being selected with probability $w_t[i]$. Then apply mini-batch gradient descent, picking mini-batches from the perturbed data set, to train a classifier θ_t. Note that the resulting classifier will take into account corruption i more when $w_t[i]$ is larger.

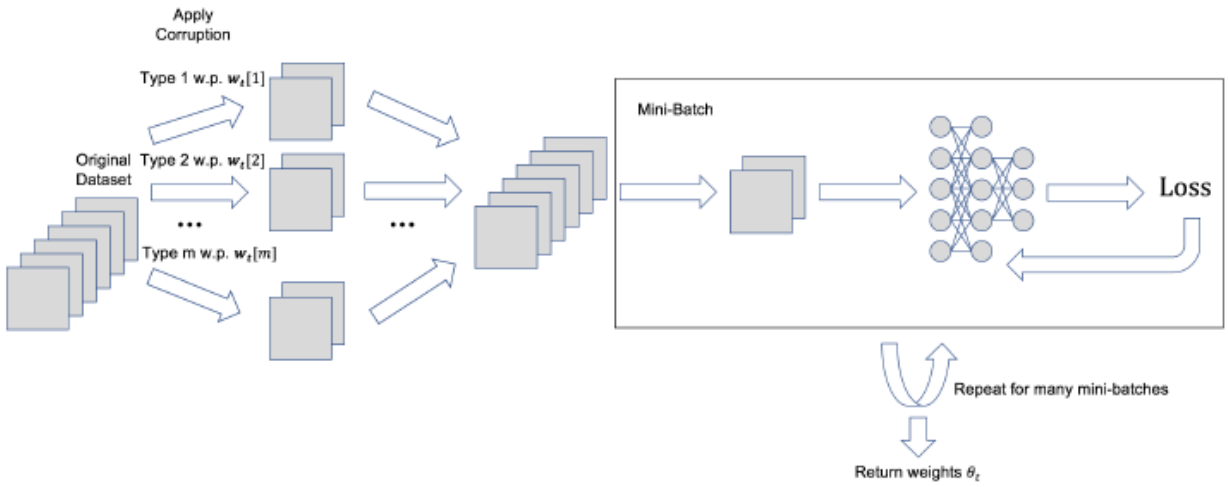

Figure 2: The Stochastic Oracle, Training on a Sample of Images Drawn from the Mixture of Corruptions

2.3.1 Robust Classification with Neural Networks

We have a data set Z of pairs (z, y) of an image $z \in \mathcal{Z}$ and label $y \in \mathcal{Y}$ that can be corrupted in m different ways which produces data sets $Z_1, \ldots, Z_m$. The hypothesis space H is the set of all neural nets of some fixed architecture and for each possible assignment of weights. We denote each such hypothesis with $h(\cdot; \theta) : \mathcal{Z} \to \mathcal{Y}$ for $\theta \in \mathrm{R}^d$, with d being the number of parameters (weights) of the neural net. If we let D_i be the uniform distribution over each corrupted data set Z_i, then we are interested in minimizing the empirical cross-entropy (aka multinomial logistic) loss in the worst case over these different distributions D_i. The latter is a special case of our robust statistical learning framework from Section 2.4.

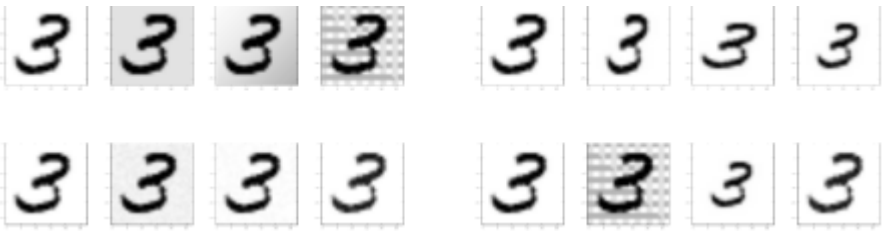

Figure 3: Sample MNIST Image with each of the Corruptions applied to it
Background Corruption Set & Shrink Corruption Set (top) and Pixel Corruption Set & Mixed Corruption Set (bottom)

Training a neural network is a non-convex optimization problem and we have no guarantees on its performance. We instead assume that for any given distribution D over pairs (z, y) of images and labels and for any loss function $\ell(h(z; \theta), y)$, training a neural net with *stochastic gradient descent* run on images drawn from D can achieve an α approximation to the optimal expected loss, i.e. $\min_{\theta \in \mathrm{R}^d} \mathrm{E}_{(z,y)\sim D} [\ell(h(z; \theta), y)]$. Notice that this implies an α-approximate stochastic oracle for the corrupted dataset robust training problem: for any distribution w over the different corruptions $[m]$, the stochastic oracle asks to give an α-approximation to the minimization problem:

$$\min_{\theta \in \mathbb{R}^d} \sum_{i=1}^{m} \mathbf{w}[i] \cdot \mathsf{E}_{(z,y)\sim D_i} \left[\ell(h(z;\theta), y)\right] \tag{10}$$

The latter is simply another expected loss problem with distribution over images being the mixture distribution defined by first drawing a corruption index *i* from w and then drawing a corrupted image from distribution D_i. Hence, our oracle assumption implies that Stochastic Gradient Descent (SGD) on this mixture is an α-approximation. By linearity of expectation, an alternative way of viewing the stochastic oracle problem is that we are training a neural net on the original distribution of images, but with loss function being the weighted combination of loss functions $\Sigma^m_{i=1}$ w[*i*] · $\ell(h(c_i(z); \theta), y)$, where $c_i(z)$ is the *i*-th corrupted version of image *z*. In our experiments we implemented both of these interpretations of the stochastic oracle when designing our neural network training scheme.

Finally, because we use the cross-entropy loss, which is convex in the prediction of the neural net, we can also apply Theorem 2.4.1 to get that the ensemble neural net, which takes the average of the predictions of the neural nets created at each iteration of the robust optimization, will also achieve good worst-case loss (we refer to this as *Ensemble Bottleneck Loss*).

Code. The code for the DNN is included with this report.

2.4 Milestone 4: Classification from Robust Guarantees

In the previous milestone we defined the algorithm for robust optimization. The performance of the algorithm was proven and we now apply our main theorem to statistical learning. Consider regression or classification settings where data points are pairs (*z*, *y*), $z \in Z$ is a vector of features, and $y \in Y$ is the dependent variable. The solution space *X* is then a space of hypotheses *H*, with each $h \in H$ a function from Z to Y We also assume that Y is a convex subset of a finite-dimensional vector space.

We are given a set of loss functions $\mathcal{L} = \{L_1, \ldots, L_m\}$, where each $L_i \in \mathcal{L}$ is a functional $L_i : H \rightarrow [0, 1]$. Theorem 2.2.1 implies that, given an α -approximate stochastic optimization oracle, we can compute a distribution over *T* hypotheses from *H* that achieves an α-approximate minimax guarantee. If the loss functionals are convex over hypotheses, then we can compute a single ensemble hypothesis *h** (possibly from a larger space of hypotheses, if *H* is non-convex) that achieves this guarantee. We state this as Theorem 2.4.1.

2.4.1 Theorem

Suppose that $\mathcal{L} = \{L_1, \ldots, L_m\}$, *are convex functionals. Then ensemble hypothesis* $h^* = \frac{1}{T}\sum_{t=1}^{T} h$, *where* $\{h_1, \ldots, h_T\}$ *are the hypotheses output by Algorithm 1 given an α-approximate stochastic oracle, satisfies*

$$\max_{i\in[m]} L_i(h^*) \leq \alpha \min_{h\in H} \max_{i\in[m]} L_i(h) + \sqrt{\frac{2\log(m)}{T}}. \tag{11}$$

We emphasize that the convexity condition in Theorem 2.4.1 is over the class of hypotheses, rather than over features or any natural parameterization of H (such as weights in a neural network). This is a mild condition that applies to many examples in statistical learning theory. For instance, consider the case where each loss $L_i(h)$ is the expected value of some ex-post loss function $\ell_i(h(z), y)$ given a distribution D_i over $Z \times Y$:

$$L_i(h) = \mathsf{E}_{(z,y)\sim D_i}\left[\ell_i(h(z), y)\right]. \tag{12}$$

In this case, it is enough for the function $\ell_i(\cdot, \cdot)$ to be convex with respect to its first argument (i.e., the predicted dependent variable). This is satisfied by most loss functions used in machine learning, such as multinomial logistic loss (cross-entropy loss) $\ell(\hat{y}, y) = -\Sigma_{c \in [k]} y_c \log(\hat{y}_c)$ from multi-class classification or squared loss $\ell(\hat{y}, y) = \| \hat{y} - y \|^2$ as used in regression. For all these settings, Theorem 2.4.1 provides a tool for improper robust learning, where the final hypothesis h^* is an ensemble of T base hypotheses from H. Again, the under-lying optimization problem can be arbitrarily non-convex in the natural parameters of the hypothesis space; in Section 2.3.1 we will show how to apply this approach to robust training of neural networks, where the stochastic oracle is simply a standard network training method. For neural networks, the fact that we achieve improper learning (as opposed to standard learning) corresponds to training a neural network with a single extra layer relative to the networks generated by the oracle.

2.5 Milestone 5: Hybrid and Composite Methods

2.5.1 Hybrid Method

In order to apply the robust optimization algorithm we need to construct a neural network architecture that facilitates it. In each iteration t, such an architecture receives a distribution over corruption types $w_t = [w_t[1], ..., w_t[m]]$ and produces a set of weights θ_t.

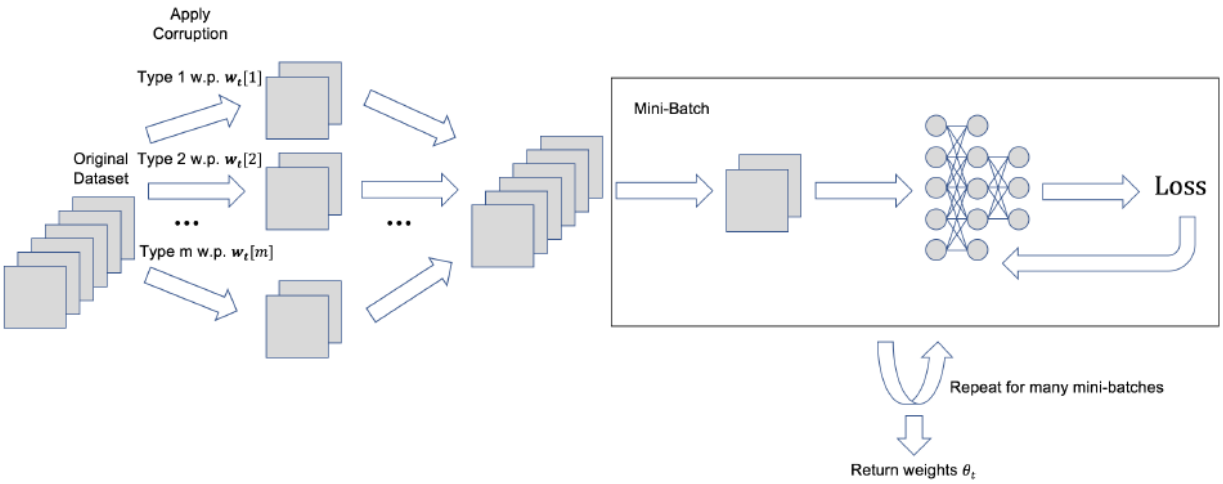

Figure 4: First Interpretation of Bayesian Oracle, Training on a Sample of Images Drawn from the Mixture of Corruptions

In the Hybrid Method, our first oracle, we take each training data image and perturb it by exactly one corruption, with corruption *i* being selected with probability $w_t[i]$. Then apply mini-batch gradient descent, picking mini-batches from the perturbed data set, to train a classifier θ_t. Note that the resulting classifier will take into account corruption *i* more when $w_t[i]$ is larger.

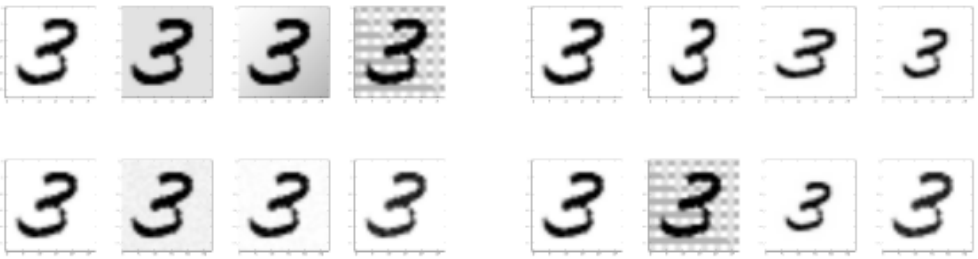

Figure 5: Sample MNIST Image with each of the Corruptions applied to it
Background Corruption Set & Shrink Corruption Set (top). Pixel Corruption Set & Mixed Corruption Set (bottom)

2.5.2 Composite Method

In the Composite Method, at each iteration, we use *m* copies of the training data, where copy *i* has Corruption Type *i* applied to all training images. The new neural network architecture has *m* sub-networks, each taking in one of the *m* training data copies as input. All sub-networks share the same set of neural network weights. During a step of neural network training, a mini-batch is selected from the original training image set, and the corresponding images in each of the *m* training set copies are used to compute weighted average of the losses $\Sigma^m_{i=1} w_t[i] Loss_{t,i}$, which is then used to train the weights.

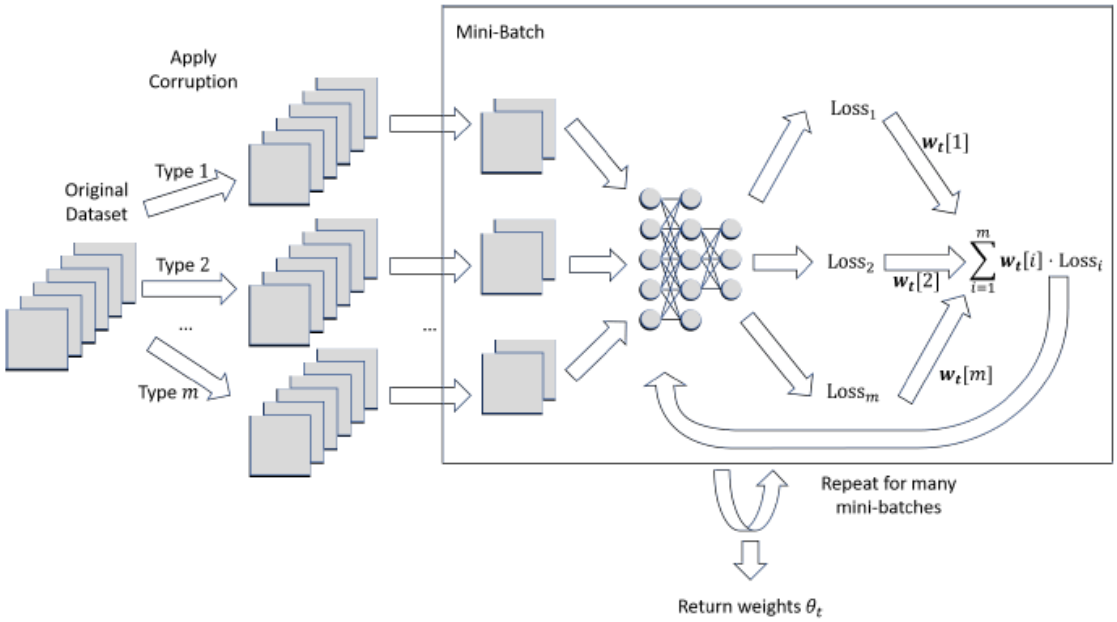

Figure 6: Second Interpretation of Bayesian Oracle
By creating m coupled instantiations of the net architecture (one for each corruption type), with the i-th instance taking as input the image corrupted with the i-th corruption and then defining the loss as the convex combination of the losses from each instance.

2.6 Milestone 6: Experiments

The purpose of our robust optimization framework is classification with neural networks for corrupted or perturbed datasets. We have a data set Z of pairs (z, y) of an image $z \in \mathcal{Z}$ and label $y \in Y$ that can be corrupted in m different ways which produces data sets $Z_1, \ldots, Z_m$. The hypothesis space H is the set of all neural nets of some fixed architecture and for each possible assignment of weights. We denote each such hypothesis with $h(\cdot; \theta) : \mathcal{Z} \to Y$ for $\theta \in \mathrm{R}^d$, with d being the number of parameters (weights) of the neural net. If we let D_i be the uniform distribution over each corrupted data set Z_i, then we are interested in minimizing the empirical cross-entropy (aka multinomial logistic) loss in the worst case over these different distributions D_i. The latter is a special case of our robust statistical learning framework from Section 2.4.

Training a neural network is a non-convex optimization problem and we have no guarantees on its performance. We instead assume that for any given distribution D over pairs (z, y) of images and labels and for any loss function $\ell(h(z; \theta), y)$, training a neural net with stochastic gradient descent run on images drawn from D can achieve an α approximation to the optimal expected loss, i.e. $\min_{\theta \in \mathrm{R}^d} \mathrm{E}_{(z,y)\sim D} [\ell(h(z; \theta), y)]$. Notice that this implies an α-approximate Bayesian Oracle for the corrupted dataset robust training problem: for any distribution w over the different corruptions $[m]$, the Bayesian oracle asks to give an α-approximation to the minimization problem:

$$\min_{\theta \in \mathrm{R}^d} \sum_{i=1}^{m} \mathbf{w}[i] \cdot \mathsf{E}_{(z,y)\sim D_i} \left[\ell\left(h\left(z;\theta\right), y\right)\right] \tag{13}$$

The latter is simply another expected loss problem with distribution over images being the mixture distribution defined by first drawing a corruption index i from w and then drawing a corrupted image from distribution D_i. Hence, our oracle assumption implies that SGD on this mixture is an α-approximation. By linearity of expectation, an alternative way of viewing the Bayesian oracle problem is that we are training a neural net on the original distribution of images, but with loss function being the weighted combination of loss functions $\Sigma^m_{i=1} \mathrm{w}[i] \cdot \ell(h(c_i(z); \theta), y)$, where $c_i(z)$ is the i-th corrupted version of image z. In our experiments we implemented both of these interpretations of the Bayesian oracle, which we call the Hybrid Method and Composite Method, respectively, when designing our neural network training scheme. Finally, because we use the cross-entropy loss, which is convex in the prediction of the neural net, we can also apply Theorem 2.4.1 to get that the ensemble neural net, which takes the average of the predictions of the neural nets created at each iteration of the robust optimization, will also achieve good worst-case loss (we refer to this as Ensemble Bottleneck Loss).

Experiment Setup. We use the MNIST handwritten digits data set containing 55000 training images, 5000 validation images, and 10000 test images, each image being a 28 × 28 pixel grayscale image. The intensities of these 576 pixels (ranging from 0 to 1) are used as input to a neural network that has 1024 nodes in its one hidden layer. The output layer uses the softmax function to give a distribution over digits 0 to 9. The activation function is a Rectified Linear Unit (ReLU) and the network is trained using Gradient Descent with learning parameter 0.5 through 500 iterations of mini-batches of size 100.

In general, the corruptions can be any black-box corruption of the image. In our experiments, we consider four types of corruption ($m = 4$):

- **Background Corruption Set** consists of images with (i) an unperturbed white background – the original images, (ii) a light gray tint background, (iii) a gradient background, (iv) and a checkerboard background.
- **Shrink Corruption Set** consists of images with (i) no distortion – the original images, (ii) a 25% shrinkage along the horizontal axis, (iii) a 25% shrinkage along the vertical axis, and (iv) a 25% shrinkage in both axes.
- **Pixel Corruption Set** consists of images that (i) remain unaltered – the original images, (ii) have *Unif*[-0.15, -0.05] perturbation added i.i.d. to each pixel, (iii) have *Unif*[-0.05, 0.05] perturbation added i.i.d. to each pixel, and (iv) have *Unif*[0.05, 0.15] perturbation added i.i.d. to each pixel.
- **Mixed Corruption Set** consists of images that (i) remain unaltered–the original images, and one corruption type from each of the previous three corruption sets (which were selected at random), namely that with (ii) the checkerboard background, (iii) 25%shrinkage in both axes, and (iv) i.i.d. *Unif*[-0.15, -0.05] perturbation.

Baselines. We consider three baselines: (i) *Individual Corruption*: for each corruption type $i \in [m]$, we construct an oracle that trains a neural network using the training data perturbed by corruption i, and then returns the trained network weights as θ_t, for every $t = 1, \ldots, T$. This gives m baselines, one for each corruption type; (ii) *Even Split*: this baseline alternates between training with different corruption types between iterations. In particular, call the previous m baseline oracles $O_1, ..., O_m$. Then this new baseline oracle will produce θ_t with O_{i+1}, where $i = t \mod m$, for every $t = 1, ..., T$; (iii) *Uniform Distribution*: This more advanced baseline runs the robust optimization scheme with the Hybrid Method, but without the distribution updates. Instead, the distribution over corruption types is fixed as the discrete uniform $\left[\frac{1}{m}, \ldots, \frac{1}{m}\right]$, over all T iterations. This allows us to check if the multiplicative weight updates in the robust optimization algorithm are providing benefit.

Results. The Hybrid and Composite Methods produce results far superior to all three baseline types, with differences both substantial in magnitude and statistically significant. The more sophisticated Composite Method outperforms the Hybrid Method. Increasing T improves performance, but with diminishing returns – largely because for sufficiently large T, the distribution over corruption types has moved from the initial uniform distribution to some more optimal stable distribution. All these effects are consistent across the 4 different corruption sets tested. The Ensemble Bottleneck Loss is empirically much smaller than Individual Bottleneck Loss. For the best performing algorithm, the Composite Method, the mean Ensemble Bottleneck Loss (mean Individual Bottleneck Loss) with $T = 50$ was 0.34 (1.31) for Background Set, 0.28 (1.30) for Shrink Set, 0.19 (1.25) for Pixel Set, and 0.33 (1.25) for Mixed Set. Thus combining the T classifiers obtained from robust optimization is practical for making predictions on new data.

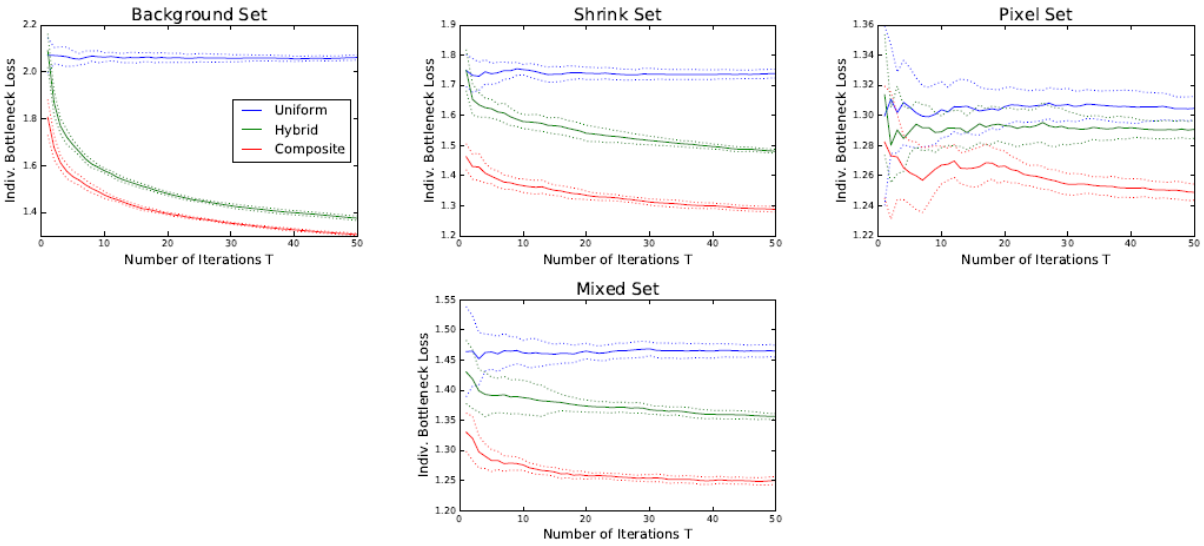

Figure 7: Comparison of methods, showing mean of 10 independent runs and a 95% confidence band

The criterion is *Individual Bottleneck Loss*: $\min_{[m]} \mathrm{E}_{\theta \sim P}\,[\ell(h(z;\theta), y)]$, where P is uniform over all solutions θ_i for that method. Baselines (i) and (ii) are not shown as they produce significantly higher loss.

2.6.1 Neural Network Results

Table 3 shows the individual bottleneck loss results (mean over 10 independent runs and a 95% confidence interval for the mean) with $T = 50$ on all four Corruption Sets. Composite Method outperforms Hybrid Method, and both outperform baselines, with such differences being statistically significant.

Table 3. Individual Bottleneck Loss Results on all four Corruption Sets

	Background Set	*Shrink Set*	*Pixel Set*	*Mixed Set*
Best Individual Baseline	**8.85** *(8.38,9.32)*	**7.19** *(7.09,7.28)*	**1.82** *(1.81,1.82)*	**8.75** *(8.50,9.00)*
Even Split Baseline	**28.35** *(26.81,29.89)*	**11.54** *(11.25,11.83)*	**1.93** *(1.91,1.95)*	**9.92** *(9.78,10.06)*
Uniform Distribution Baseline	**2.06** *(2.05,2.08)*	**1.74** *(1.72,1.76)*	**1.30** *(1.30,1.31)*	**1.46** *(1.45,1.47)*
Hybrid Method	**1.38** *(1.37,1.39)*	**1.48** *(1.47,1.49)*	**1.29** *(1.28,1.30)*	**1.36** *(1.35,1.36)*
Composite Method	**1.31** *(1.30,1.31)*	**1.30** *(1.29,1.31)*	**1.25** *(1.24,1.25)*	**1.25** *(1.24,1.26)*

2.6.2 Analysis of Multiplicative Weights Update

Consider the robust optimization algorithm using the Hybrid and Composite Methods, but parameterizing η as $\eta = c \cdot T^{-\gamma}$ (for constant $c = \sqrt{\frac{\log m}{2}}$) to alter the multiplicative weights update formula. In this project, we have been using $\gamma = 0.5 \implies \eta = \frac{c}{\sqrt{T}}$. Lower values of γ leads to larger changes in the distribution over corruption types between robust optimization iterations. Here we rerun our experiments from Section 2.3.1 using γ = 0.1 (see Figure 8); we did not tune γ – the only values of γ tested were 0.1 and 0.5.[2]

[2]A possible future step would be to use cross-validation to tune γ or design an adaptive parameter algorithm for γ.

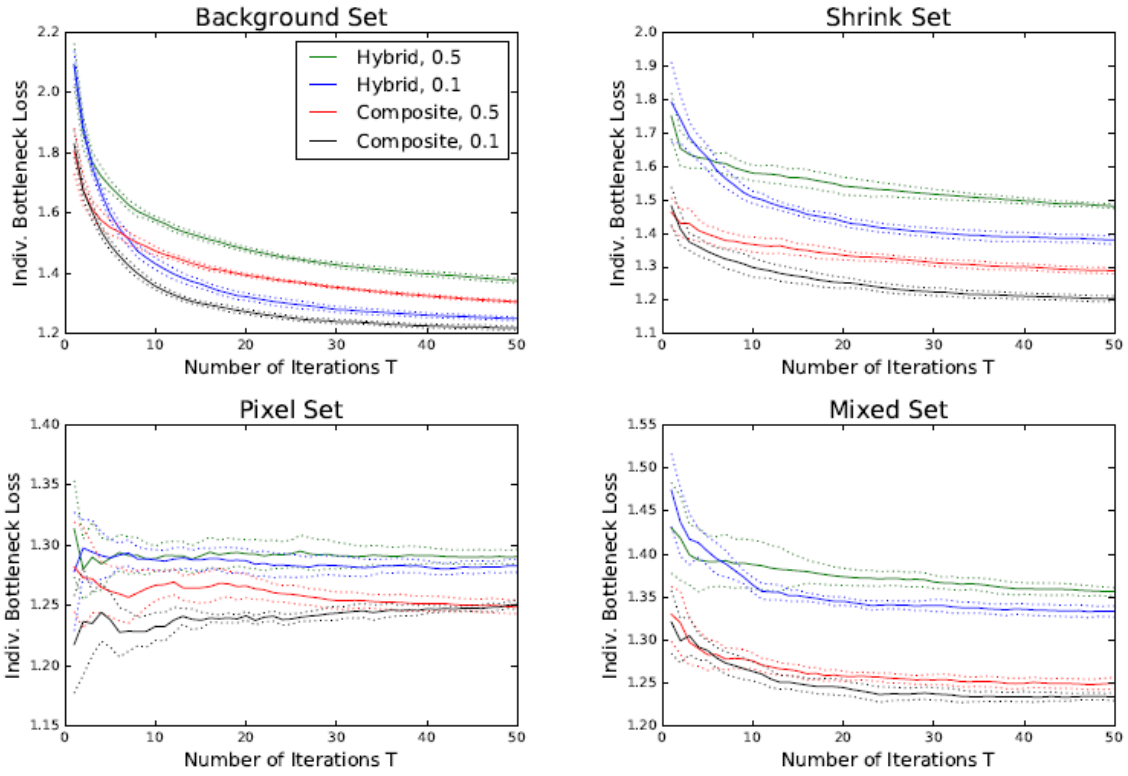

Figure 8: Comparison of Individual Bottleneck Loss between using γ = 0.5 vs. γ = 0.1 in the Multiplicative Weights update, for both the Hybrid and Composite Methods
The γ = 0.1 setting yields lower loss.

The improved performance with γ = 0.1 compared to γ = 0.5 is related to an important property of our robust optimization algorithm in practice – namely that w stabilizes for sufficiently large *T*. Over the course of iterations of the algorithm, w moves from the initial discrete uniform distribution to some optimal *stable distribution*, where the stable distribution is consistent across independent runs. The γ = 0.1 setting yields to better Individual Bottleneck Loss than the γ = 0.5 setting for finite *T* because it converges more rapidly to the stable distribution.

The left plot in Figure 9 shows 16 time series, corresponding to results for each combination of ({Hybrid, Composite},{γ = 0.5, = γ 0.1},{Background, Shrink, Pixel, Mixed}), using the mean over 10 runs. The right plot shows the difference between pairs of time series from the previous figure (thus there are 16/2 = 8 times series shown). Values are positive for small *t* and near 0 for larger *t*, showing that the γ = 0.1 setting yields faster changes in w initially, thereby allowing w to more quickly approach the stable distribution.

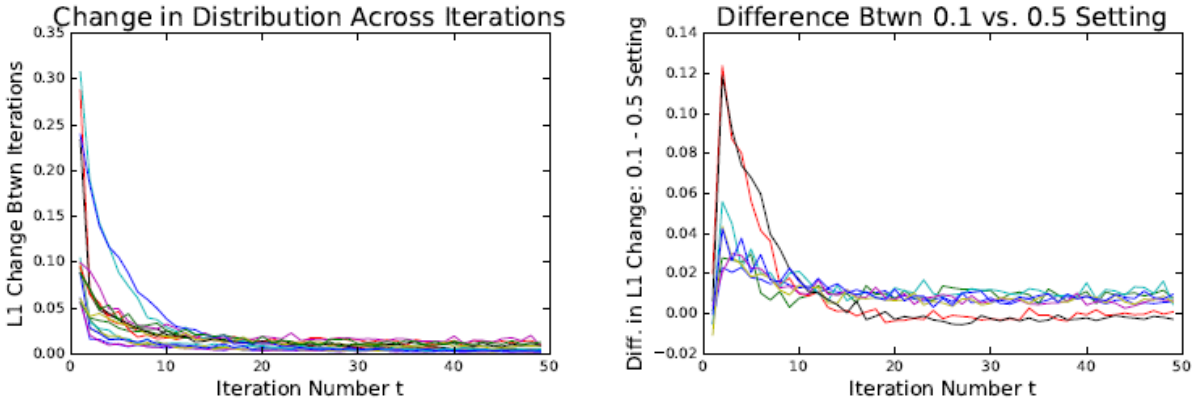

Figure 9: Amount that the Distribution over Corruption Types w changes between Iteration *t* & *t* + 1 Decays Rapidly as *t* Increases and the Distribution Stabilizes (left) and Difference between γ = 0.1 & γ = 0.5 in the Amount that w changes between Iterations (right)

List of Abbreviations, Acronyms, and Symbols

ACRONYM	DESCRIPTION
AFRL	Air Force Research Laboratory
DNN	Deep Neural Network
MNIST	Modified National Institute of Standards and Technology
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent