AD-A261 597

ıENTATION PAGE

Form Approved
OMB No. 0704-0188

ⓓ

| | |
|---|---|
| ɪᴇᴘᴏʀᴛ ᴅᴀᴛᴇ: 18 December 1992 | 3. REPORT TYPE AND DATES COVERED: Final, 9/1/89--8/31/92 |
| 4. TITLE AND SUBTITLE: "Using Modular Neural Networks with Local Representations to Control Dynamic Systems" (U) | 5. FUNDING NUMBERS: C (AFOSR-89-0500) 2305/B3 61102F |
| 6. AUTHOR(S): Professor Christopher G. Atkeson | |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES): Massachusetts Institute of Technology<br>Dept. of Brain & Cognitive Sciences, and<br>The Artificial Intelligence Laboratory<br>545 Technology Square, Room NE43-771<br>Cambridge, MA 02139 | 8. PERFORMING ORGANIZATION REPORT NUMBER: AFOSR-TR- 93 0062 |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES): Air Force Office of Sponsored Research/NE<br>Electronic and Material Sciences Directorate<br>Building 410<br>Bolling AFB, Washington, DC 20332-6448 | 10. SPONSORING/MONITORING AGENCY REPORT NUMBER: AFOSR-89-0500 |
| 11. SUPPLEMENTARY NOTES | |
| 12a. DISTRIBUTION/AVAILABILITY STATEMENT: Approved for Public Release Distribution Unlimited: | 12b. DISTRIBUTION CODE: UL |

DTIC
ELECTE
MAR 2 1993
S C D

13. ABSTRACT (Maximum 200 words)

The objective of the research was to develop an artificial neural network with very fast learning. MAny areas of activity and approaches have convinced us that we can perform training and access sufficiently quickly to allow real-time learning.

| | |
|---|---|
| 14. SUBJECT TERMS: Neural networks, memory-based learning, motor control | 15. NUMBER OF PAGES: 10 |
| | 16. PRICE CODE |

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| unclassified | unclassified | unclassified | SAR |

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

300' 19Ed

83:51 16, 15 NEC

| Accesion For | |
|---|---|
| NTIS CRA&I | ✓ |
| DTIC TAB | ☐ |
| Unannounced | ☐ |
| Justification | |
| By | |
| Distribution / | |
| Availability Codes | |
| Dist | Avail and / or Special |
| A-1 | |

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Brain and Cognitive Sciences and
The Artificial Intelligence Laboratory
Rm. NE43-771 • 545 Technology Square, Cambridge, MA 02139
(617) 253-0788 • cga@ai.mit.edu

December 18, 1992

Steven Suddarth
Program Manager, Neural Networks
Electronic and Material Sciences Directorate
AFOSR/NE
Bldg. 410
Bolling AFB, DC 20332-6448

Dear Dr. Suddarth,

The following is the Final Technical Report for the grant "Using Modular Neural Networks With Local Representations To Control Dynamic Systems" (AFOSR-89-0500) for the period 9/1/89 - 8/31/92. I also enclose several papers that cover the work done in that period.

The objective of this research was to develop an artificial neural network with very fast learning. Major areas of activity included developing cross validation methods to refine parameters such as the distance metric, developing parallel versions of the learning algorithms which allow implementations to be scaled up by simply adding additional processing hardware, with a negligible penalty in processing time, and performing numerical experiments on simulated data to test the approach. We have also compared our approach with other neural network approaches, and found that it provides equal or better performance. We have implemented versions of this approach on several platforms: serial computers (standard Sun workstations), digital signal processors (Intel i860), a parallel computer (Connection Machine), and using special purpose digital circuitry. We showed that we can perform training and access sufficiently quickly to allow real-time learning, with a real-time implementation of memory-based learning on a juggling task. We have also explored alternative methods such as radial basis functions and projection pursuit, and developed a more demanding experimental task: helicopter control.

The funds provided by the Air Force had a large impact on education as well as on research, providing partial support for several graduate students and undergraduates. Ying Zhao (Mathematics, a woman) received her PhD from the MIT Mathematics Department for work done under this grant. She was also supervised by Prof. Peter Huber in the Mathematics Department. Sherif Botros (Brain and Cognitive Sciences) explored using radial basis functions for motor learning and optimization. Gerrie Van Zyl (Mechanical Engineering) built the juggling robot robot testbed for learning methods. Gideon Stein (Electrical Engineering and Computer Science) explored digital implementation technologies for our learning approach and programmed the Intel i860. Chiqian Xie (Mechanical Engineering) worked on the learning control of flexible objects, an important and difficult class of learning problems since the order (dimensionality of the state vector) is not known. Stefano D'Aquino (Electrical Engineering and Computer Science) explored analog implementation technologies. Several undergraduates received prizes for their thesis work under this grant, including Peter Gordon ("An Associative Content Addressable Memory", S.B. in Electrical Engineering and Computer Science, May 1990. Awarded Second Prize in the David Adler Memorial Thesis Competition by the Undergraduate Thesis Committee of the Department of Electrical Engineering and Computer Science), and Paul Sajda, ("Machine Implementation of a Human Motor Task: The Yo-Yo Robot", S.B. in Electrical Engineering and Computer Science, May 1989. Awarded Second Prize in the David Adler Memorial Thesis Competition by the Undergraduate Thesis Committee of the Department of Electrical Engineering and Computer Science). The students played a major role in the research, and also learned a great deal. This research has also been incorporated in several courses, including courses on motor learning, computational approaches to motor learning, and nonparametric regression applied to learning and optimization.

Sincerely,

[illegible]

Christopher G. Atkeson
Associate Professor

cc: Dr. Harry Klopf

# Final Report: Using Modular Neural Networks With Local Representations To Control Dynamic Systems

Christopher G. Atkeson

The objective of this research was to explore an artificial neural network architecture that simply remembers experiences and builds local models to answer particular queries. The reason we are interested in a memory-based approach is that it offers the possibility of fast training and minimal interference. The approach is to model complex systems using many simple local models. This approach avoids the difficult problem of finding an appropriate structure for a global model. To implement this approach a two-part artificial neural network is designed. One part uses a local representation to remember the training data set, and the second part is trained on selected portions of the training set to form local models as needed. This network architecture can be simulated using k-d tree data structures on standard serial computers and also using parallel search on a massively parallel computer such as the Connection Machine. The performance of the network was initially evaluated by using it to control simulated dynamic systems. The ultimate goal is to demonstrate successful control of actual dynamic systems such as a robot helicopter.

One scientific contribution was to develop and demonstrate the utility of non-parametric methods for adaptive control. Memory-based control provides a new and possibly better method for solving control problems such as the robot trajectory following problem. Another contribution is to the study of learning in intelligent systems. Demonstrating that learning a local representation is fast and efficient and that the problems of limited memory capacity can be overcome is an important contribution to the continuing debate between proponents of local and distributed representations. This is of great relevance to neuroscientists who study biological systems, as well as to computer designers. This research will also contribute to our understanding of human learning. We currently do not understand how humans learn motor tasks. Comparing the behavior of memory-based learning algorithms to actual human learning may give us insight into how humans learn and how they might be taught more effectively.

Memory-based learning provides one approach to fast training, in that the representation is trained by storing experiences in a large memory. This corresponds to winner take all networks which use a local representation, and can be used to perform nearest neighbor search for relevant experiences. Locally weighted regression is a form of memory-based learning in which a model is fit to relevant experiences in order to make a prediction. Locally weighted regression approximates complex functions using simple local models, as does a Taylor series. During training, experiences are stored in a memory. When a query must be answered, experiences relevant to the query are found and combined to form a local model. Examples of types of local models include nearest neighbor, weighted average, and locally weighted regression. Each of these local models combine points near to a query point to estimate the appropriate output. Locally weighted regression uses a relatively complex regression procedure to form this model, and is thus more expensive than nearest neighbor and weighted average memory-based learning procedures. For each query a new local model is formed. The rate at which local models can be formed and evaluated limits the rate at which queries can be answered. However, we have found that locally weighted regression can be implemented in real time, and it has been implemented for online robot learning of a challenging control task (a juggling task known as "devil sticking"). We used commercially available microprocessors (Intel i860 and Texas Instruments TMS320C30). We have also found that memory-based learning avoids interference between new and old data by retaining and using all the data to answer each query.

To illustrate how the local model is formed in locally weighted regression we will first consider a global model formed using unweighted regression. The inputs of each training data point form a row in the matrix $\mathbf{X}$, and the outputs form a corresponding row in the vector $\mathbf{y}$. The structure of the model is chosen so it is linear in the unknown parameters, which appear in the vector $\beta$ and are related by the set of linear equations

$$\mathbf{X}\beta = \mathbf{y}$$

Since there are more equations (data points) than unknown parameters, the parameters $\beta$ are chosen by minimizing the sum of the squared fitting errors.

$$\min_{\beta} (\mathbf{X}\beta - \mathbf{y})^T (\mathbf{X}\beta - \mathbf{y})$$

This sum is minimized by the solution of the normal equations:

$$\left(\mathbf{X}^T\mathbf{X}\right)\beta = \mathbf{X}^T\mathbf{y}$$

We can think of this process as minimizing the energy of a set of identical springs connecting the data points to the model surface. A problem with unweighted regression is that points distant to the query point have as much influence on the answer to the query as nearby points (for equally spaced data).

Locally weighted regression reduces the influence of distant points on the query answer by weighting the data according to its distance from the query point. In order to do this one needs to know what the query point is and to have a distance metric and a weighting function that transforms the calculated distance into a weight:

$$d_i^2 = \sum_j m_j(\mathbf{X}_{ij} - \mathbf{q}_j)^2$$

$$w_i = d_i^{-p}$$

Each row of $\mathbf{X}$ and $\mathbf{y}$ is multiplied by the weight for that point, $w_i$. We can think of locally weighted regression as minimizing the energy of a set of springs whose spring constants decrease with distance from the query point.

Often there is not enough data in all directions, which leads to an ill-conditioned regression problem. The estimates are stabilized by adding small positive numbers to the diagonal of the $\mathbf{X}^T\mathbf{X}$ matrix. This technique is known as ridge regression in statistics. It is equivalent to adding fake data in each direction that has a small weight and a zero output value. The ridge regression constants can also be thought of as Bayesian priors on the variance of the estimated parameter vector $\beta$.

$$\left(\mathbf{X}^T\mathbf{X} + \Lambda\right)\beta = \mathbf{X}^T\mathbf{y}$$

We use off-line cross validation to estimate reasonable values for the fit parameters: the distance metric $m_j$, weighting function $w_i = d_i^{-p}$, and ridge regression parameters $\lambda_j$. Since we are using a local model that is linear in the unknown parameters $\beta$, we can compute derivatives of the cross validation error $e_i = \hat{y}_i - y_i$ with respect to the fit parameters:

$$\frac{\partial e_i}{\partial m_j} \quad \frac{\partial e_i}{\partial p} \quad \frac{\partial e_i}{\partial \lambda_j} \tag{1}$$

and minimize the sum of the squared cross validation error using a Levenberg-Marquardt procedure.

Lookup has three stages: forming weights, forming the regression matrix, and solving the normal equations. Let us examine how the cost of each of these stages grows with the size of the data set and dimensionality of the problem. We will assume a linear local model.

Forming and applying the weights involves scanning the entire data set, so it scales linearly with the number of data points in the database ($n$). For each of $d$ input dimensions there are a constant number of operations, so the number of operations scales linearly with the number of input dimensions.

$$w_i = f\left(\sum_j m_j(\mathbf{X}_{ij} - \mathbf{q}_j)^2\right)$$

Note that we can eliminate points whose distance is above a threshold, reducing the number of points considered in subsequent stages of the computation.

Each element of $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}^T\mathbf{y}$ is the inner (dot) product of two columns of $\mathbf{X}$ or $\mathbf{y}$. The architecture of digital signal processors is ideally suited for this computation, which consists of repeated multiplies and accumulates. The computation is linear in the number of rows $n$ and quadratic in the number of columns $(d^2 + d * o)$, where $d$ is the number of input dimensions and $o$ is the number of output dimensions.

Solving the normal equations is done using a $LDL^T$ decomposition, which is cubic in the number of input dimensions, and independent of the number of data points. Other more sophisticated and more expensive decompositions, such as the singular value decomposition, do not need to be used since the ridge regression procedure guarantees that the normal equations will be well-conditioned and this cost is small compared with forming $\mathbf{X}^T\mathbf{X}$.

The most straightforward parallel implementation of locally weighted regression would distribute the data points among several processors. Queries can be broadcast to the processors, and each processor can weight its data set and form its contribution to $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}^T\mathbf{y}$. These contributions can be summed and the full normal equations solved on a single processor. The communication costs are logarithmic in the number of processors and quadratic in the number of columns ($d^2 + d * o$), and independent of the total number of points.

We have implemented the local weighted regression procedure on a 33MHz Intel i860 microprocessor. The peak computation rate of this processor is 66 MFlops. We have achieved effective computation rates of 20 MFlops on a learning problem with 10 input dimensions and 5 output dimensions, using a linear local model. This leads to a lookup time of approximately 20 milliseconds on a database of 1000 points.

A question that often arises with memory-based models is the effect of memory limitations. We have not yet needed to address this issue in our experiments. However, we plan to explore how memory use can be minimized based on several approaches. One approach is to only store "surprises". The system would try to predict the outputs of a data point before trying to store it. If the prediction is good, it is not necessary to store the point. Another approach is to forget data points. Points can be forgotten or removed from the database based on age, proximity to queries, or other criteria. Because memory-based learning retains the original training data, forgetting can be explicitly controlled.

## 1 Performance Comparisons

Two methods, CMAC (Albus 1975ab) and sigmoidal feedforward neural networks, were compared to the approach explored in this paper. The parameters for the CMAC approach were taken from Miller, Glanz, and Kraft (1987) who used the CMAC to model arm dynamics. The architecture for the sigmoidal feedforward neural network was taken from Goldberg and Pearlmutter (1988, section 6) who also modeled arm dynamics.

The ability of each of these methods to predict the torques of the simulated two joint arm at 1000 random points was compared. Figure 1 plots the normalized RMS prediction error. The points were sampled uniformly using ranges comparable to those used in (Miller et al 1987). Initially, each method was trained on a training set of 1000 random samples of the two joint

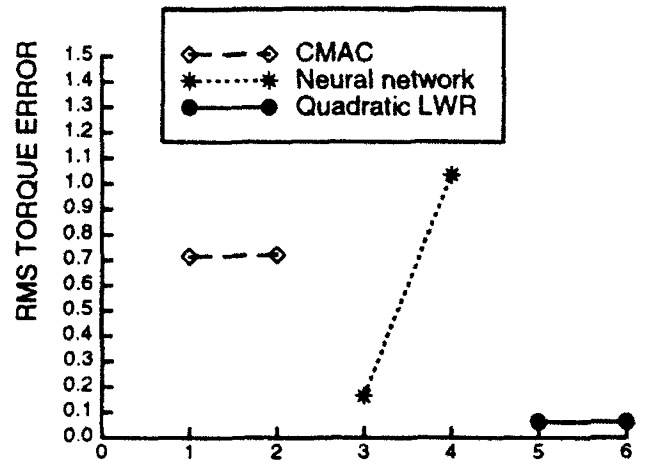

Figure 1: Performance of various methods on two joint arm dynamics.

arm dynamics function, and then the predictions of the torques on a separate test set of 1000 random samples of the two joint arm dynamics function were assessed (points 1, 3, and 5). Each method was then trained on 10 attempts to make a particular desired movement. Each method successfully learned the desired movement. After this second round of training, performance on the random test set was again measured (points 2, 4, and 6).

The data indicate that the locally weighted regression approach (filled in circles) and the sigmoidal feedforward network approach (asterisks) both generalize well on this problem (points 3 and 5 have low error). The CMAC (diamonds) did not generalize well on this problem (point 1 has a large error), although it represented the original training set with a normalized RMS error of 0.000001. A variety of CMAC resolutions were explored, ranging from a basic CMAC cell size covering the entire range of data to a cell size covering a fifth of the data range in each dimension. A cell size covering one half the data ranges in each dimension generalized best (the data shown here).

After training on a different training set (the attempts to make a particular desired movement), the sigmoidal feedforward neural network lost its memory of the full dynamics (point 4), and represented only the dynamics of the particular movements being learned in the second training set. This interference between new and previously learned data was not prevented by increasing the number of hidden units in the single layer network from 10 up to 100. The other methods explored did not show this interference effect (points 2 and 6).

## 2 Other Methods Explored

In the pursuit of fast training methods we explored a variety of techniques for fast or real time function approximation, including radial basis functions and projection pursuit regression. A graduate student, Sherif Botros, looked at ways to speed up radial basis function based learning, and make it more effective (Botros and Atkeson, 1991). Radial basis functions are a form of neural network model in which the hidden units are multidimensional "bumps". The function to be learned is approximated by summing the bumps:

$$f(\mathbf{x}) = \sum c_i g(||\mathbf{x} - \mathbf{x}_i||)$$

For fixed bump locations and shapes, estimating $c_i$ is a linear regression problem, making this approach attractive for linear adaptive control methods. In our research we have found that the choice of distance metric is critical. We have found heuristics for estimating good initial metrics, which can be refined using nonlinear parameter estimation techniques. Using these techniques we have found that radial basis functions are currently the most effective neural network approach to modeling robot arm rigid body dynamics. A remaining challenge is that the RBF approach generally leads to a large regression problem, which is difficult to implement in real time.

Another graduate student, Ying Zhao, tried to understand learning based on projection pursuit regression (Zhao and Atkeson, 1991a, 1991b). Projection pursuit regression is a form of neural network model in which the hidden units are general one dimensional functions rather than sigmoids:

$$f(\mathbf{x}) = \sum g_i(\theta_i^T \mathbf{x})$$

One can view one hidden layer sigmoidal neural networks as a specialization of projection pursuit networks:

$$f(\mathbf{x}) = \sum c_i \sigma(\theta_i^T \mathbf{x} + \delta_i)$$

We are exploring heuristics to choose good initial directions ($\theta_i$) for hidden units. We have also found that projection pursuit learning networks work better on angular smooth functions than on Laplacian smooth functions.

Here "work better" means that for fixed complexities of hidden unit functions and a certain approximation accuracy requirement fewer hidden units are required; or given a fixed number of hidden units a better accuracy can be achieved. As of yet we have no real time implementation of projection pursuit learning networks.