Using A Neural Network to Learn the Dynamics of the CMU Direct-Drive Arm II

Ken Goldberg and Barak Pearlmutter
Carnegie Mellon University

August 1988

CMU-CS-88-160

Abstract

Computing the inverse dynamics of a robot arm is an active area of research in the control literature. We apply a backpropagation network to this problem and measure its performance on the first 2 links of the CMU Direct-Drive Arm II for a family of "pick-and-place" trajectories. Trained on a random sample of actual trajectories, the network is shown to generalize with a root mean square error/standard deviation (RMSS) of 0.10. The resulting weights can be interpreted in terms of the velocity and acceleration filters used in conventional control theory. We also report preliminary results on learning a larger subset of state space for a simulated arm.

This research was sponsored in part by National Science Foundation grants DMC-8520475, EET-8716324, and by the Office of Naval Research under contract number N00014-86-K-0678. Barak Pearlmutter is a Fannie and John Hertz Foundation fellow.

1. Introduction

Existing robot arms sacrifice speed for flexibility. To maintain control under increased speeds, an accurate model of arm dynamics is desired. Since coupled degrees of freedom are often used to orient tools in the workspace, arm dynamics are highly non-linear. Analytic models have improved in recent years [Hollerbach 80], but unmodelled effects such as friction or torque non-linearity can be significant. By training a neural network on the measured response of a physical arm, we hope to compensate for dynamic effects that are difficult to model analytically.

1.1. Motivation

From a control perspective, a robot arm is a filter; we put in torques at one end and at the other end we observe positions, velocities and accelerations (state). By *dynamics*, we mean the transfer function that relates input to output. Typically, we measure the current state of the arm and want to achieve some desired state of the arm. The inverse dynamics tell us what torques to apply.

In the absence of external disturbances, a perfect model of the dynamics generates a perfect torque signal to achieve perfect arm motion. In the presence of noise and uncertainty, an approximate model generates an approximate torque while feedback is used to compensate for small errors in joint motion. This is known as the *computed torque* method, or the *feedforward* method if the torques are generated off-line. A typical approach is to employ a a set of independent linear PID (Position Integral Derivative) controllers at each joint and a second-order model to compensate for inter-joint coupling [Asada 82]. Yet finding an accurate model for arm dynamics has proven difficult.

The formulation of robot models in terms of classical Lagrangian-Euler (L-E) dynamics has been an area of research for the past 20 years [Hollerbach 80]. This formulation can give physical insight into the relative contributions of inertial, centrifugal, Coriolis, gravitational and actuation torques, but it faces two essential problems: the computational complexity of the model requires several hundred multiplications per cycle, and the idealized model does not necessarily reflect the true response of the arm.

One way to speed up the computations is to simplify the equations by ignoring certain terms [Bejczy 74] or using recursive methods of computation [Hollerbach 80]. [Khosla 86] customizes the L-E model for a particular arm and uses a floating-point processor to achieve a sampling period of 1.2 ms (830 Hz).

The problem of accuracy remains. The L-E equations include terms for joint dimensions, mass, and inertia. The latter is often difficult to measure although methods have been developed to attack the so-called *identification problem* [An 88, Khosla 86].

Most importantly, the L-E equations do not attempt to model such real-world effects as

- friction [Canudas 87]
- backlash [An 88]
- torque non-linearity (especially dead zone and saturation) [An 88]
- high-frequency dynamics [An 88]
- sampling effects [Khosla 86]
- sensor noise [Khosla 86]

A way to address these effects is to model the arm empirically using a *model-based* control scheme. One class of model-based schemes is the *adaptive* controllers, where coefficients in the L-E equations are modified on-line to minimize a performance/stability index. (See [Craig 86] for a bibliography, [Slotine 87] and [Han 87] for more recent work). Adaptive controllers often compensate for the unmodeled effects by treating them as variations in the L-E terms, but why should these unmodeled effects be squeezed into the Procrustean bed of the L-E formulation?

Another approach is to scrap the L-E model and treat the arm as an unknown transfer function. The function can be represented as a table of input/output pairs gathered by running the arm with a feedback controller. Input torques can later be indexed by desired output state to generate feedforward torque for subsequent control. The central problems with this approach are the *data generation problem:* how to evenly sample the state space, and the *generalization* problem: how to interpolate new values from those in the table.

It is possible to avoid both problems by tailoring the controller to a specific trajectory; if the state space is sampled along a particular trajectory it can be sampled densely enough to minimize interpolation effects. This is the approach described in [Raibert 78], where performance is shown to degrade sharply outside the sampled trajectory. Similarly, a tabular method combined with a novel hashing scheme was applied to the control problem with good results in simulation [Miller 87].

To generate torques for a general class of trajectories, the tabular approach requires storing a vast amount of data. One way to minimize the data storage is to fit a polynomial to the data. Unfortunately, one is placed on the horns of a dilemma. If the polynomial is of low order it cannot represent higher-order functions. On the other hand, if one allows high order terms, the solution tends to oscillate in the unconstrained areas in an effort to hit the data points exactly. One approach [Zhang 87] is to attempt to balance off these conflicting goals and find the happiest medium possible. The polynomial-fitting approach has been applied to the problem of control with good results [Yen 87] and warrants further investigation.

Another way to minimize data storage is to use a neural network representation. The relative power of neural networks vs. polynomials is an open area of research. One way to compare these representations is by the number of coefficients needed to represent a given function. For example, the *parity* function, which is 1 or 0 depending on whether the number of 1 bits in an input string of length n is even or odd, requires $O(2^n)$ coefficients if represented as a polynomial but $O(n \log n)$ coefficients if represented with a backpropagation network.

1.2. Backpropagation Networks

The term "neural network" applies to a variety of parallel schemes consisting of *units* and *connections* where each unit performs a weighted sum of its input connections and uses this sum to determine an activity level, which other units see as an input. The weights are either set externally or, more commonly, learned by some learning procedure.

Currently, the bread and butter connectionist learning procedure is back-propagation [Rumelhart 86], which repeatedly adjusts the weights in a network so as to minimize a measure of the difference between the actual output vector of the network and a desired output vector given the current input vector. The output of the network is taken from the last layer of units after all unit operations are complete, and the connections and flow of activity in the network are unidirectional. The simple weight adjusting rule is derived by propagating partial derivatives of the error backwards through the net using the chain rule. Experiments have shown that back-propagation can learn non-linear functions and make fine distinctions between input patterns in the presence of noise [Lang 87, Lapedes 87, Waibel 88]. Moreover, starting from random initial states, back-propagation networks can learn to use their intermediate layers, or ***hidden*** units, to efficiently represent structure, such as cascaded filters, that is inherent in the desired transfer function. Although the backpropagation procedure imposes few constraints on the transfer function used, in this paper units compute their activity level as a function of their total input using the sigmoid function, $output = (1+e^{-input})^{-1}$.

It is thus tempting to apply neural networks to the domain of robot arm control. [Kawato 87] shows some evidence that a network can learn the inverse dynamics of a real robot arm; after training on a single trajectory, they claim their network can generalize to a "faster and quite different" trajectory, although details are omitted. In this paper we measure the ability of a neural network to generalize within a specific family of trajectories on a physical arm.

2. Problem Definition

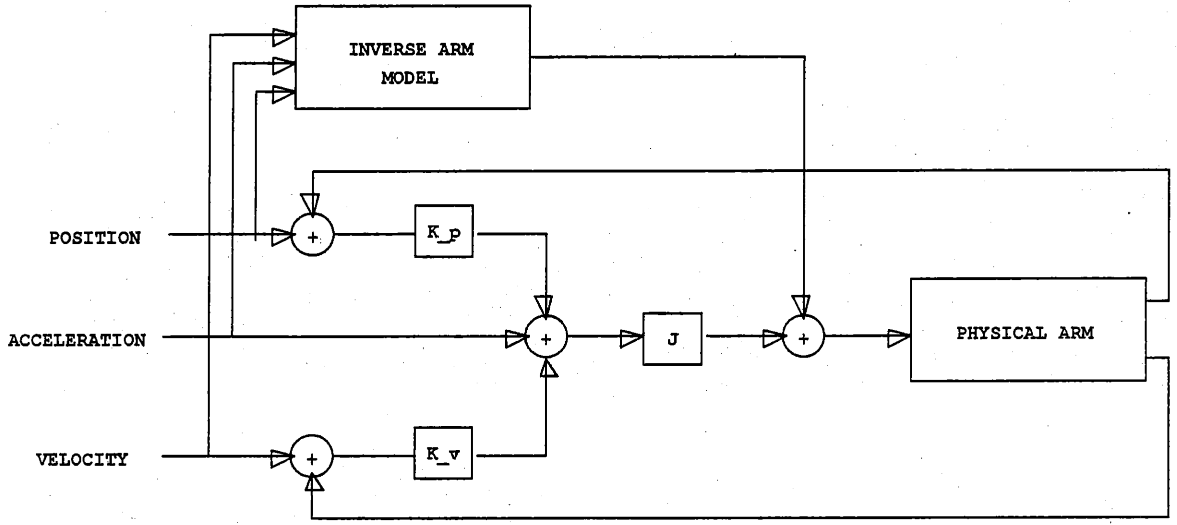


Figure 2-1: Block diagram of a typical *feedforward torque* controller, after [Khosla 86].

A typical dynamics control structure for a robot arm is shown in figure 2-1. In this method the feedforward torques for a desired trajectory are computed off-line using a model of the inverse arm and applied to the joints at every cycle in an effort to linearize the resulting system. An independent feedback loop at each joint is used to correct for errors in the model and external disturbances.

We propose to use a backpropagation network to fill the box marked ''inverse arm'' in the diagram. We will treat the arm as an unknown non-linear transfer function to be represented by weights of the network.

Since the state of the arm is represented by a window of positions (angles) for joint 0 and joint 1, our state space is a 2n-dimensional hypercube of size π. Due to the temporal structure of the window, much of this space is unreachable. We sample a subset of the space traversed by a family of "pick-and-place" trajectories. The family is characterized by a common initial and final position (0 degrees) and an intermediate position that can vary between 0 and 45 degrees for each joint independently. Between these three points, the reference trajectory (position and velocity) is a sinusoid scaled so that the period of the trajectory is 3 seconds. Thus any member of the family can be specified by a pair of intermediate joint positions; we draw random samples from this family by generating pairs of random numbers.

We focus on the generalization problem: trained on a random sample of this family, how well does the neural network generalize to other trajectories in the family?

2.1. The CMU DD Arm II

We tested our approach on the CMU Direct-Drive Arm II [Khosla 86]. Direct-drive arms have the capacity to be driven much faster than geared arms, but their ability to be backdriven exacerbates dynamic effects. DD arms are thus a popular target for dynamics-based control.

3. Network Architecture

The network architecture used in this study is shown in figure 3-1. The input to the network consists of a temporal "window" of desired position values $x(t-n\Delta t), \ldots, x(t), \ldots, x(t+m\Delta t)$ scaled from 0.0 to 1.0. The output is $\tau(t)$, the torque vector applied at time t. The input units are connected to a set of hidden units which are in turn connected to the output units. In addition, there are direct connections from the input units to the output units.

A neural network is used in two phases. In the learning phase, the network is exposed to samples of input/output

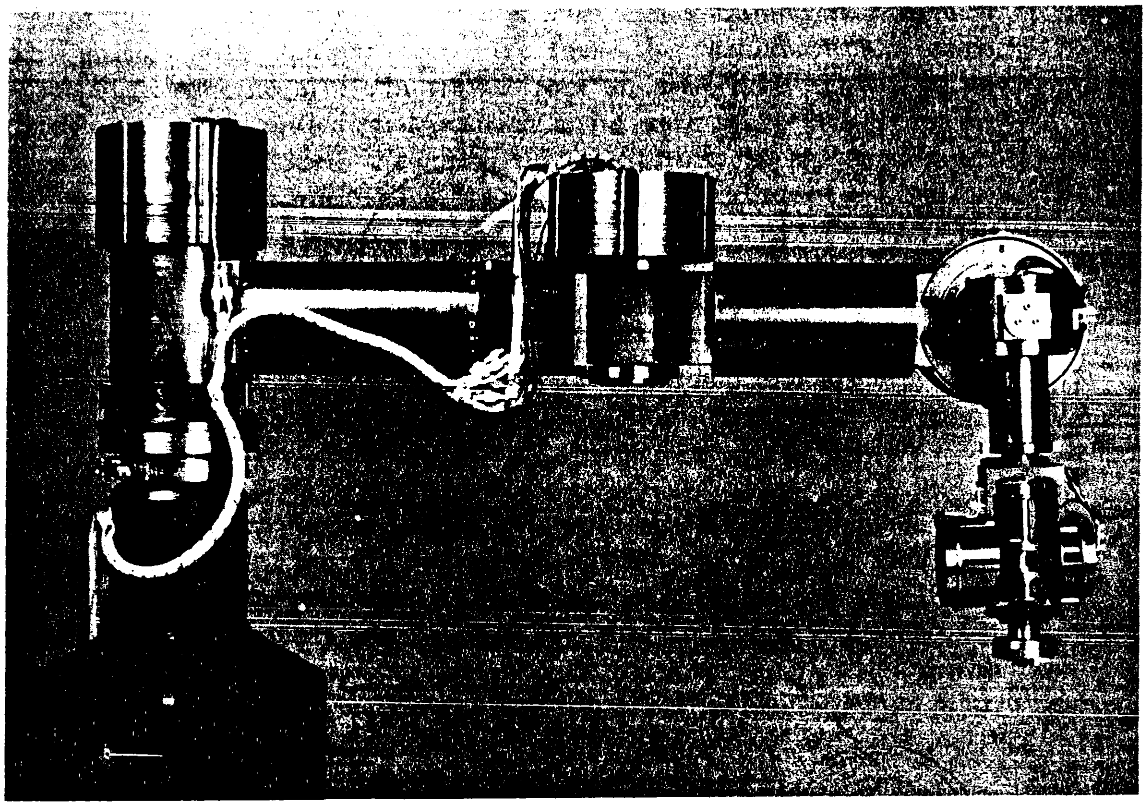


Figure 2-2: A photograph of DD II.

behavior and the weights are adjusted so that network output approximates desired output. When the weights are stable, the network is run in performance mode on a new set of input data. If the weights capture the underlying structure of the sample data, the output generated for the new set will be close to the desired output.

3.1. Temporal Windows

Rather than feeding the network position, velocity and acceleration data from a single point in time, the networks sees a window of positions. We chose this approach because of the conceptual elegance of using only one type of state information; velocity and acceleration can be determined by filtering the window of position values. The time delay introduced by such filtering in real time is avoided because learning occurs off-line.

Hardware cost is another reason to eschew explicit velocity and acceleration measurements. Tachometers at each joint increase the cost of a robot. The alternative technique of simple differencing introduces noise into the estimates. In the neural network, the state filter is part of the model, and so is tailored to both the particular arm and the sensor characteristics (noise, sampling delay, etc).

Independent of our choice to deny the network the output of the velocity sensors, we can justify the use of temporal windows instead of a single time slice because we do not know *a priori* what state information is relevant to generating feedforward torques. For example, it is possible that higher-order terms such as jerk and snap are relevant if the arm has some elasticity. By providing a window of position values, higher-order terms can be extracted by the network.

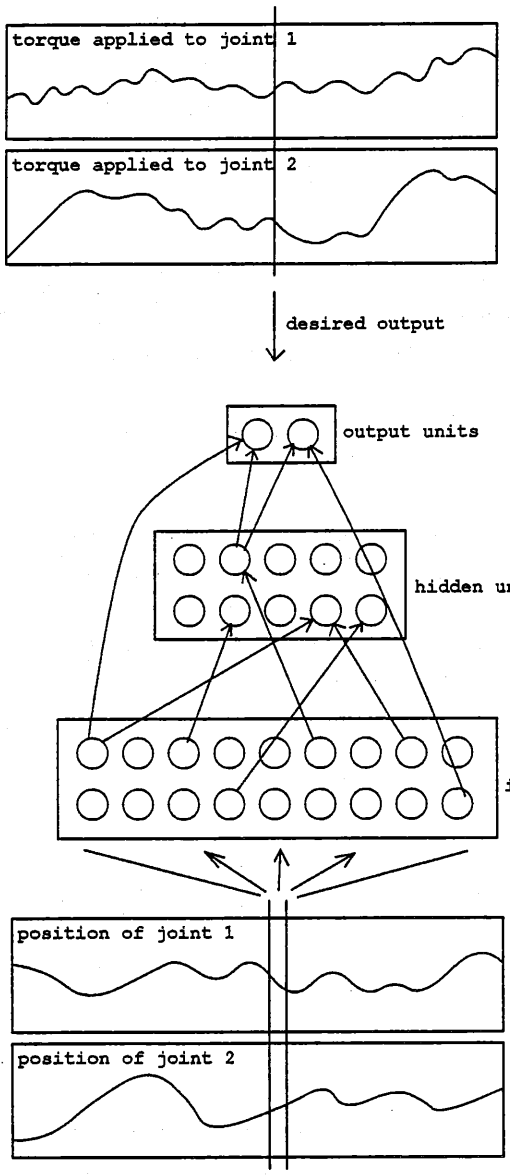


Figure 3-1: Backpropagation Network.

3.2. Network Topology

If the number of joints being controlled is *n*, the window size is *w*, the number of hidden units *h*, and we assume that there are many more hidden units than output units, the number of weights in the network is approximately *nwh*.

In performance mode, each weight requires a single multiplication and and single addition, and for reasonably large networks these computations dominate the sigmoid computations (which are typically implemented as table lookups). Thus, there is a linear relationship between the amount of computation required and the window size, the number of joints, and the number of hidden units.

One of our objectives was to explore the relationship between window size and generalization. Although considerable experience has been gained on choosing the appropriate numbers of hidden units and I/O encodings for backpropagation networks being applied to discrete binary tasks, these issues are largely unexplored in continuous domains.

3.3. Learning Parameters

All of these networks had ten hidden units, as experimentation showed performance to be insensitive to increasing the number of hidden units beyond this point. Each was trained to an asymptote, i.e. to the point where the derivative of the error was nearly zero.[1] We made extensive use of the method of acceleration, which seemed particularly effective in this domain.

4. Results

A proportional controller was used to drive the first 2 links of the DD arm along 6 random trajectories (see Appendix I) from the specified family. Joint torque and position were measured at the actuator. Five trajectories were used to train the network. The network was then run in performance mode on the remaining trajectory to test generalization.

Table 4-1 shows the performance of networks with window sizes of 11, 21, and 41. As in [Lapedes 87], we use root mean square error divided by the dataset's standard deviation (RMSS) as an error measure. This cancels out the effects of the translation and scaling needed to fit desired output data within the range of backpropagation units. A window of size 11 refers to a window centered at time *t* that includes 5 time steps before and 5 time steps after *t*, yeilding a total of eleven samples. As expected, the network performs better on the training set than on the new trajectories. The lowest error rate was achieved with a window size of 21.

Figure 4-1 shows measured vs. predicted torque for joint 1 (the shoulder) on the test trajectory.

	RMSS errors on:	
Window Size	Training Data	Trajectory 6
11	0.078	0.22
21	0.027	0.10
41	0.024	0.13

Table 4-1: RMSS errors of networks with different window sizes on training and test data.

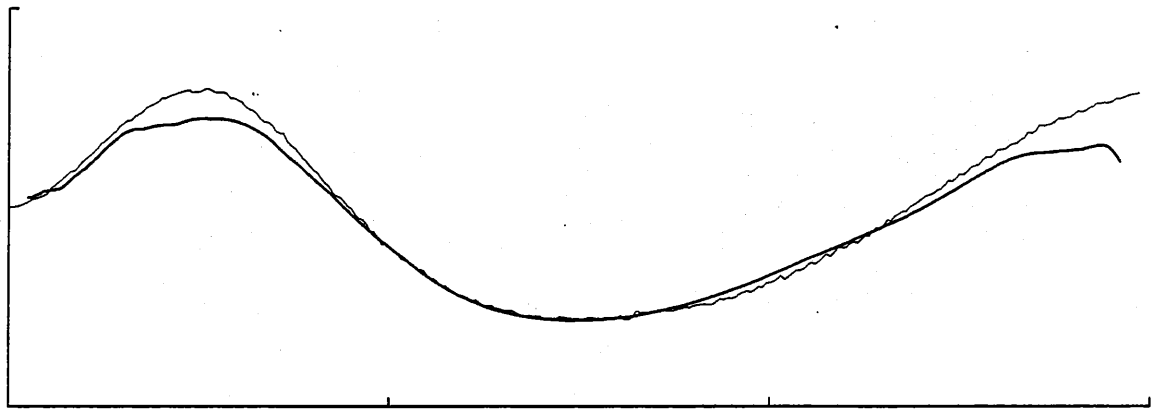
5. Analysis

5.1. Weight Displays

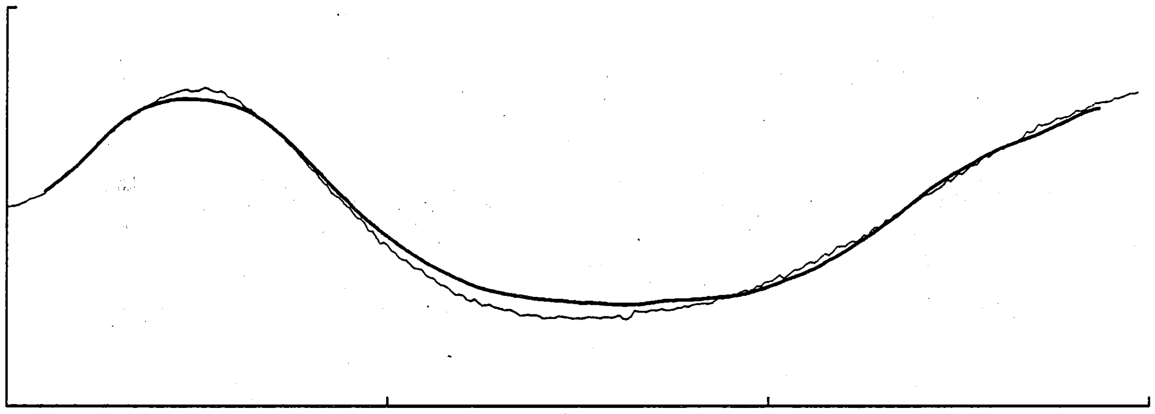
Figures 5-1, 5-2, 5-3 and 6-1 show the weights developed by the networks after training. These "Hinton diagrams" show the weights in a somewhat recursive fashion. Each of the large-scale blobs is a unit; here, the top two are the output units, with the shoulder torque on the left and the elbow torque on the right. The rest are hidden units. Within each unit, the two stripes on the bottom (or, in the case of figure 6-1, the single bottom stripe) show the weights of the incoming connections from the input units. The top two dots on each of the hidden units are the weights of its outgoing connections to the output units. These hidden-to-output connections are also displayed in the middle portion of each of the output units. The single remaining unexplained blob on the upper left is each unit's bias, the strength of a connection from a unit which is always on, which is equivalent to the negative of the threshold. The white blobs are positive and black ones are negative.

We can interpret the weights between the first and third layers (shown in the upper right-hand part of the figure) as temporally smooth filters shaped to detect a linear combination of position, velocity, and acceleration. At first

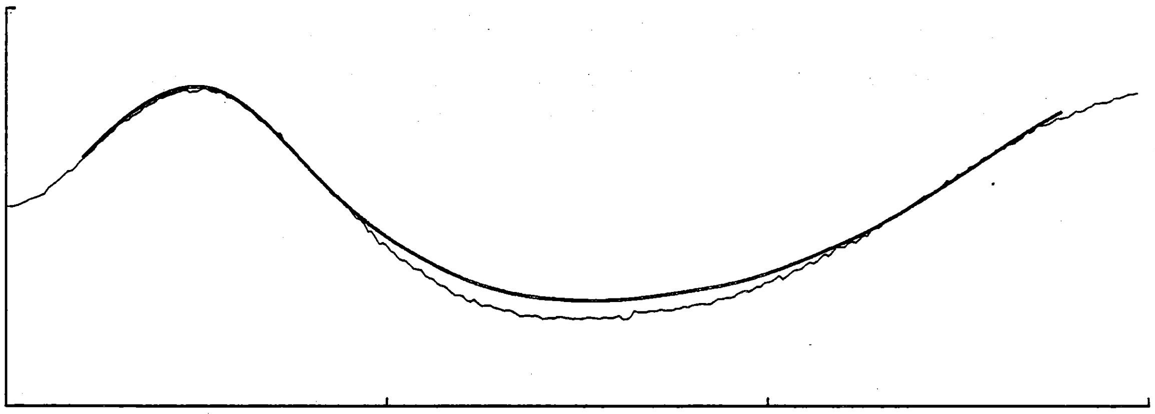
[1]In general, when training on a training corpus and testing using a different testing corpus, performance on the training corpus rises monotonically to an asymptote while performance on the test corpus first rises to a maximum and then falls to an asymptote. Many researchers, quite reasonably, use the best test corpus performance achieved as the generalization rate. In our work, we have used the more pessimistic asymptotic test corpus performance metric.



Window Size 11 (see figure 5-1)



Window Size 21 (see figure 5-2)



Window Size 41 (see figure 5-3)

Figure 4-1: These graphs show torque profiles that are to drive joint 1 (the shoulder) through trajectory 6. The measured torque profile is drawn with a fine line, while the torque profiles generated by networks are drawn with bold lines.

glance most of the units appear to respond almost solely to acceleration, but on closer examination one sees that the zero crossings are frequently a little asymmetric, an indication that velocity is also being responded to. The linear combination of acceleration and velocity stands out in the network of figure 5-3, in which some of the filters are strikingly asymmetric. It should be remembered that the networks have no built in notion of temporal adjacency. Because these filters are convolved with the position in every possible place, we think of them as convolution functions and attempt to analyze them in these terms.

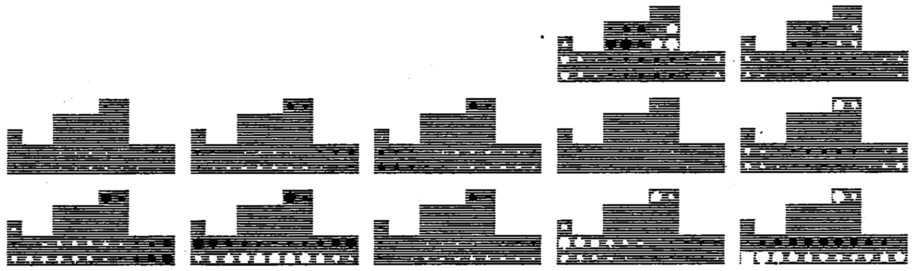


Figure 5-1: This network has a window size of 11. The largest weight has a magnitude of 18.9.

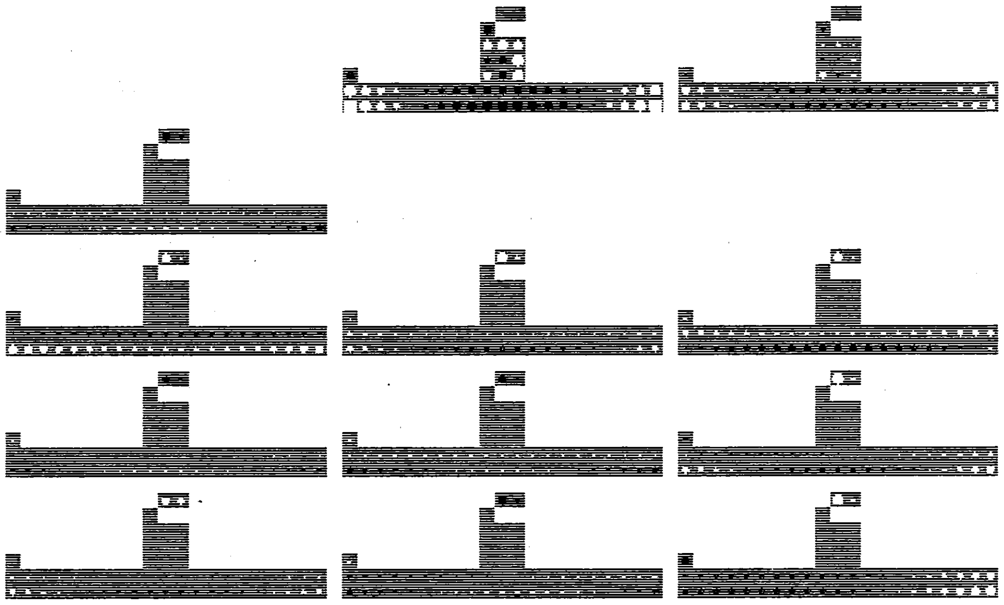


Figure 5-2: This network has a window size of 21. The largest weight has a magnitude of 11.7.

If we assume that the weights are samples from a continuous function that is convolved with the position of the joint, we can decompose this function $f(x), -w \leq x \leq w$ into a constant part, an odd part, and an even part using the equations

$$const = \frac{1}{2w}\int_{-w}^{w} f(x)dx$$

$$even(x) = \frac{f(x)+f(-x)}{2}$$

$$odd(x) = \frac{f(x)-f(-x)}{2} - const.$$

Figure 5-4 shows that the weights on the connections from the elbow joint to a unit taken from the network with window size 21 can be understood as a simple velocity filter. Figure 5-5 shows another unit from that network whose weights from the shoulder joint form a simple acceleration filter, and figure 5-6 shows a unit which is activated by a linear combination of velocity and acceleration.

The functions we have examined so far have been extremely smooth. This is generally the case in the network with window size 21, but in the network with window size 41 a new phenomenon appears. In figure 5-7 we see some curves with rough edges; in the section below on window size vs. generalization, we advance an explanation for this odd behavior.

In figure 5-7, the odd component crosses zero near the edges of the window, suggesting that this unit responds in part to jerk (the derivative of acceleration).

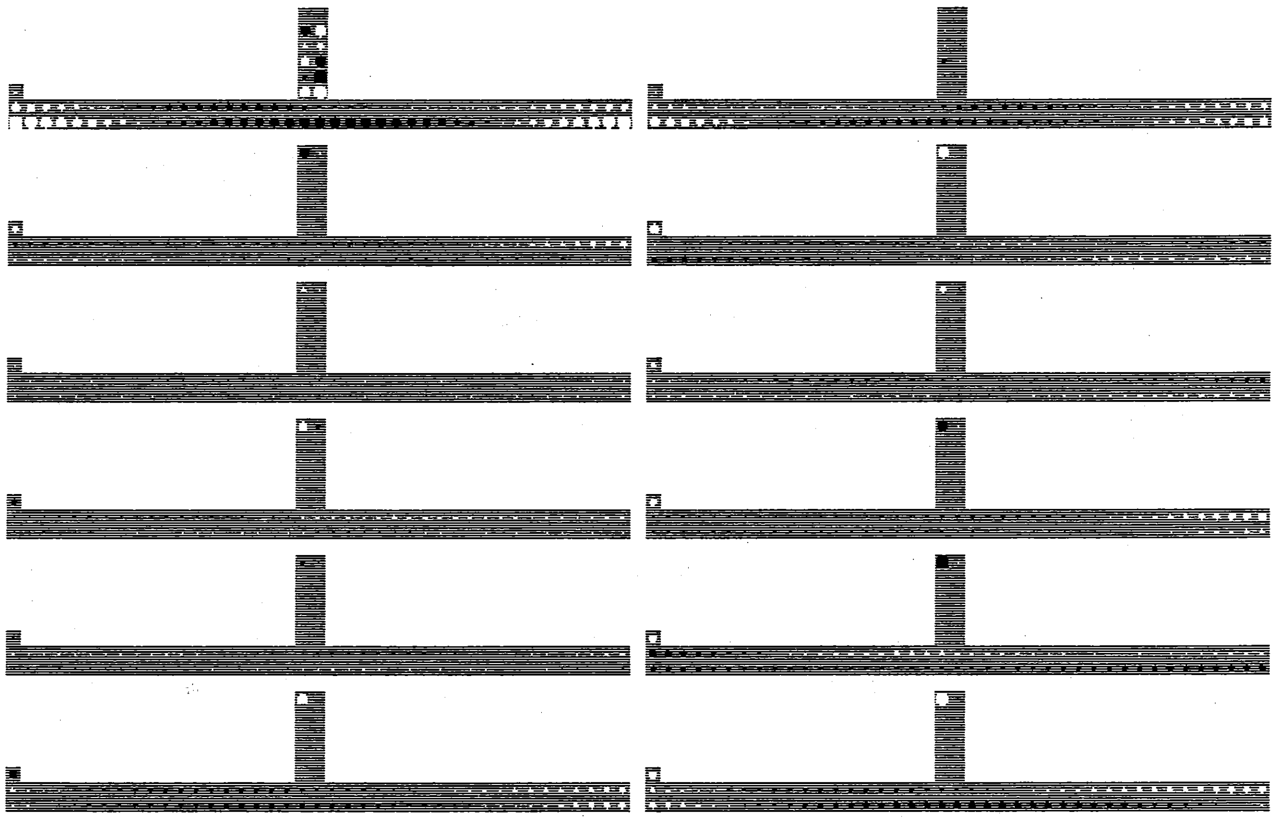


Figure 5-3: This network has a window size of 41. The largest weight has a magnitude of 4.2.

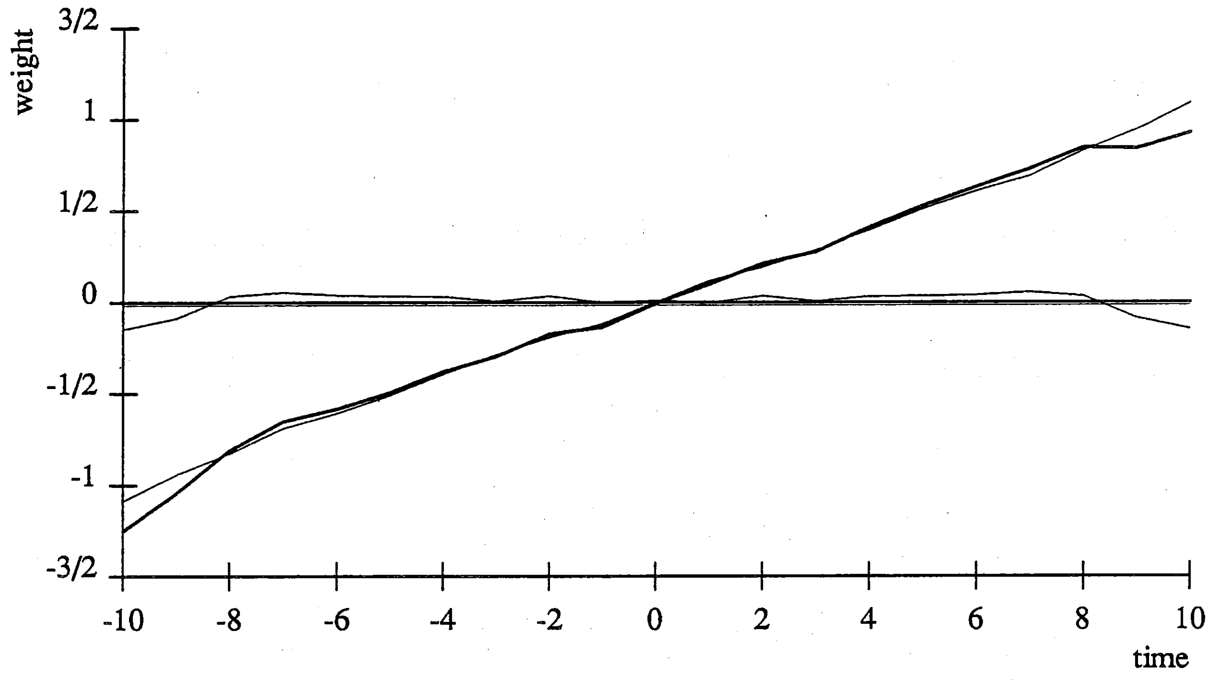
It might be objected that any function can be decomposed into even and odd components, but for arbitrary functions the decomposition would not in general yield smooth curves. For instance, figure 5-8 shows weights a set of weights from the network of window size 41 that do not decompose cleanly. Further, the constant terms in our filter functions were extremely small, suggesting an underlying symmetry.

5.2. A More Global Perspective

The roles of the individual hidden units is more difficult to interpret. Since the weights are quite large, most units are saturated most of the time. Each unit has a transition region in which it is not saturated, but rarely is it in this region. For example, a unit might respond to $x = 3.4a + 1.2da/dt - 2.1b$, being saturated at 0 if x is less than 2.3, at 1 if x is greater than 2.6, and having an intermediate value only within that narrow range. Thus, the input space is chopped up into soft hyperplanes along these dimensions. The use made of the hidden units by the output units provides little information about their roles in the network in intuitive terms, as the values are used in concert, canceling each other out under different input conditions. In a word, the networks are not modular: it is difficult to understand the roles of the various units in isolation.

5.3. Window Size vs. Generalization

Observe from table 4-1 that performance on the training set improves as the window gets larger, while performance on the test set first improves as window size grows, and then worsens, evidence of a tradeoff between window size and generalization. We conjecture that the improved generalization between a window of size five and a window of size ten is caused by the fact that a window of size five simply does not see enough data to make sufficiently accurate estimates of the acceleration. In contrast, the network with a window of size twenty seems to have used a portion of its extra capacity to memorize some of the training set, thus improving performance on the



joint 2 to unit 51

Figure 5-4: A graph of the inputs to a unit taken from the network with a window size of 41 regarded as a convolution function and decomposed into constant, even, and odd components. These weights form a velocity filter.

training set in a way that impairs generalization.

Evidence of this memorization is visible in figure 5-3, where some of the hidden units have receptive fields which have isolated black and white dots, an indication that they respond to some particular pattern of noise that occurred in the training set. Another signature of this memorization is the surprisingly low magnitudes of the weights, which sacrifices accuracy of the acceleration and velocity filters for the ability to detect these patterns of noise.

6. Preliminary Results from Simulation

Due to the practical difficulty of obtaining samples from the entire phase space of an actual robot, we used a simulated arm in order to see if a backpropagation network is able to generalize to a larger subset of state space. We simulated the inverse dynamics of the CMU DD2 Arm using the parameters from [Khosla 86] and the L-E formulation found in [Brady 82], representing the state of the arm as a 6-tuple: position, velocity, and acceleration for each joint. We uniformly sampled the 6-dimensional subspace bounded by a hypercube of radius π. In other words, for each sample point we generated six random numbers between π and $-\pi$.

The analytic model was used to generate the corresponding torques for each sample point. We trained the network in figure 6-1 on this data until it reached an asymptote at 12,000 epochs; these weights are shown in figure 6-1. Due to the analytic nature of the arm simulator, it was neither necessary nor expedient to use a temporal window; rather, the network was presented with the exact position, velocity and acceleration data. To test the network's ability to generalize, we generated another random set of 298 points and a set of 298 points corresponding to a sinusoidal trajectory from our pick-and-place family. The RMSS errors are shown in table 6-1 and overlaid plots of the predicted and analytic torques for the sinusoidal trajectory are shown in figure 6-2.

It is interesting to try to figure out how the network in figure 6-1 is performing its task. The hidden units each

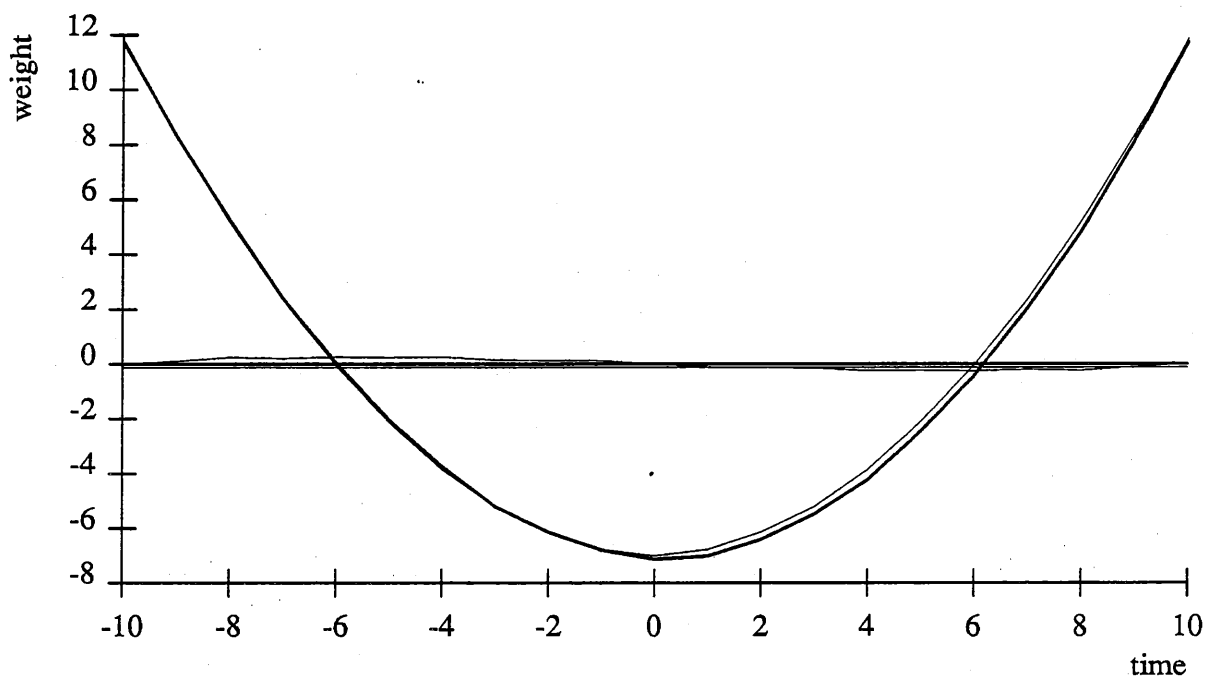


Figure 5-5: A graph of the inputs to a unit taken from the network with a window size of 41 regarded as a convolution function (bold line) and decomposed into constant, even, and odd components (fine lines). These weights form an acceleration filter.

seem to be sensitive to only a particular range of velocities of the shoulder joint; the position of the shoulder is (properly) ignored.

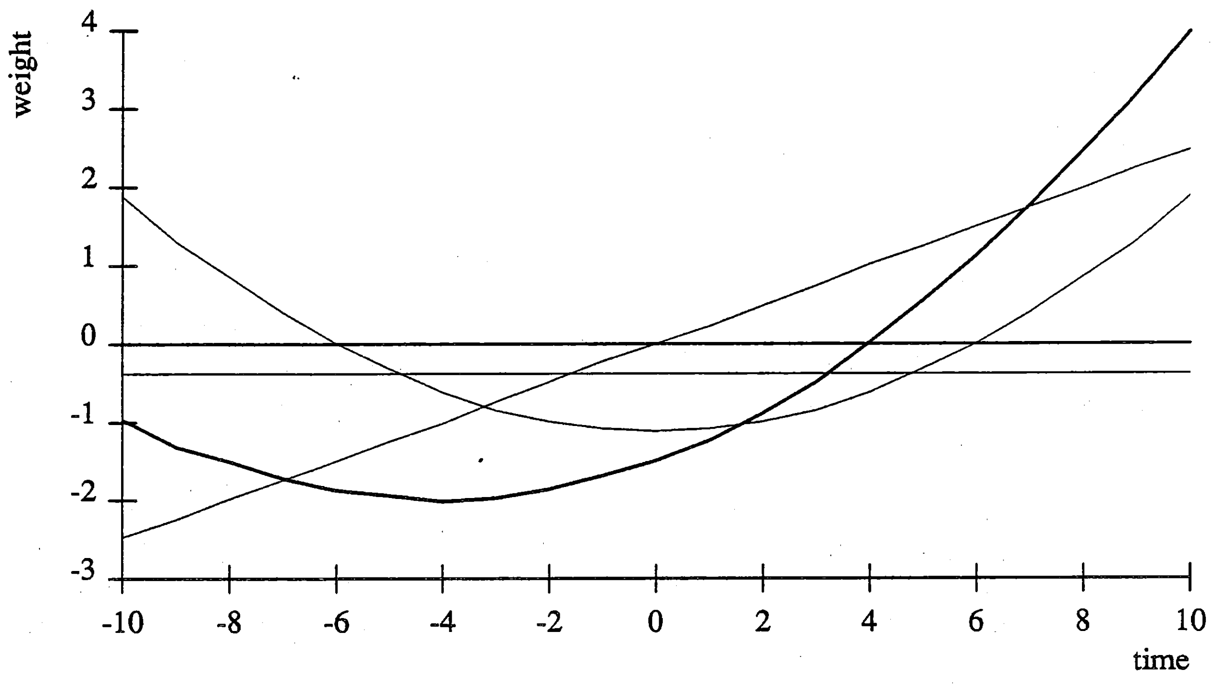
Quantitative measures of the performance of this network on both its training set some test sets is shown in table 6-1. The excellent performance on random set B indicates that the network has generalized well from its sample of 298 points. The performance on trajectory 6, as well as the graphs shown in figure 6-2, show that the network performs well in that portion of phase space which the trajectories lie in. This suggests that neural networks might be able to learn the dynamics of an actual robot arm over the entirety of the arm's state space.

7. Conclusion

We have shown that a three-layer backpropagation network is capable of generating accurate feedforward torques for a limited family of pick-and-place trajectories on a physical robot arm. By using a temporal window of position values as input, the resulting weights can be interpreted as the velocity and acceleration filters used in conventional control theory. The internal units also contribute to the transfer function, but we are unable to analyze their role. We conjecture that these units compensate for effects such as friction and torque non-linearity so that the resulting transfer function accurately reflects the physical arm.

7.1. Future Work

The next step is to use these torques at run-time and evaluate their effect on endpoint error. This will involve minor modifications to the existing control software. We would like to address more general families of trajectories; the family used in this paper has a single via point in the middle of the trajectory. We are beginning studies on a family of circular "stirring" trajectories which explore other regions of state space. We would like to test the neural



joint 2 to unit 52

Figure 5-6: A graph of the inputs to a unit taken from the network with a window size of 41 regarded as a convolution function (bold line) and decomposed into constant, even, and odd components (fine lines). These weights form a filter with responds to a linear combination of velocity and acceleration.

network on the 3rd and 4th joint of the DD arm. The extra links will complicate the dynamic interactions and test the robustness of the neural network architecture in higher-dimensional state spaces. Lastly, we would like to integrate the neural network into the controller and construct on on-line version of the system.

7.2. Acknowledgements

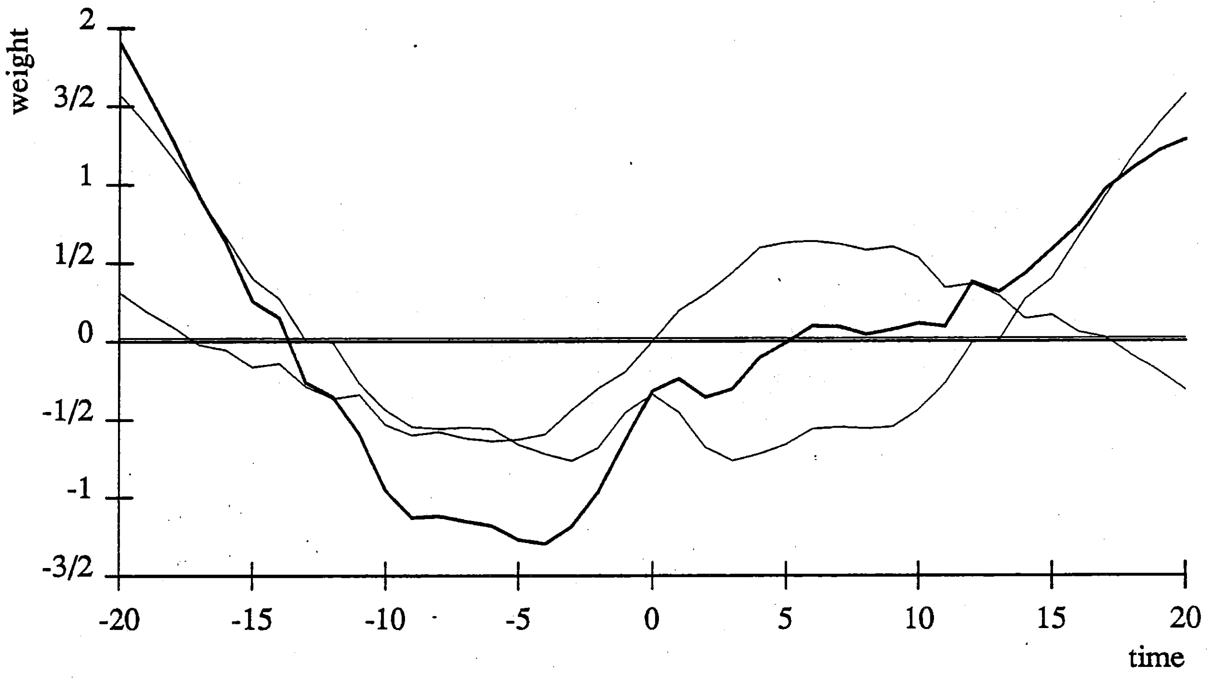
The DD arm was used with the kind and invaluable assistance of Pradeep Khosla. Other experiments were run on Lee Weiss' 2D semi-direct-drive arm, and we express our gratitude to him. Regrettably, logistic difficulties prevented us from including data gathered from the Weiss arm in this document. We also thank Randy Brost and Mark Nagurka for their comments.

I. Trajectories used

The family of three second trajectories we used were parameterized by two values, θ_1 and θ_2, which specify the maximum extent of the shoulder and elbow joints. Other joints were held at 0. The trajectories were generated using the parametric equation

$$\phi_i = -\theta_i \sin\frac{\pi}{3}t$$

where t is the time in seconds. The trajectories mentioned in this report were generated used the following random extents:



joint 2 to unit 93

Figure 5-7: A graph of the inputs to a unit taken from the network with a window size of 81 regarded as a convolution function (bold line) and decomposed into constant, even, and odd components (fine line). These weights seem to form a filter which responds to a linear combination of acceleration and jerk, and also to a particular pattern of noise.

trajectory number	$\theta_{shoulder}$	θ_{elbow}
1	45.0	45.0
2	38.6	32.2
3	5.4	39.4
4	20.9	0.0
5	0.0	45.0
6	39.5	14.4

Trajectories 1 through 5 comprised the training set, and 6 was reserved for testing. We collected 100 samples per second, resulting in 300 time samples per trajectory. Since we trained on the data from 5 trajectories, the environment presented to our networks had approximately 1500 cases. For our largest network, this translates to over 700,000 floating point multiplications per epoch, and training required over 20,000 epochs.

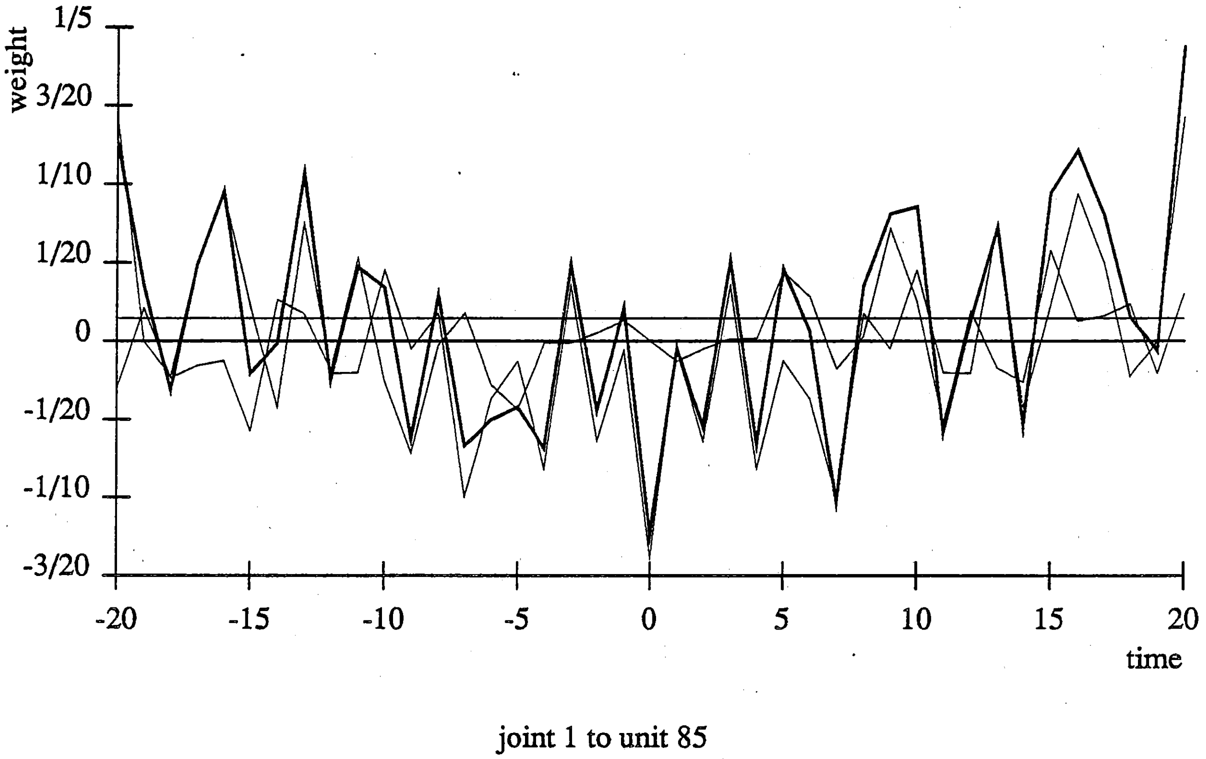


Figure 5-8: A graph of the inputs to a unit taken from the network with window size 81 regarded as a convolution function (bold line) and decomposed into constant, even, and odd components (fine lines). These weights do not seem amenable to this sort of analysis.

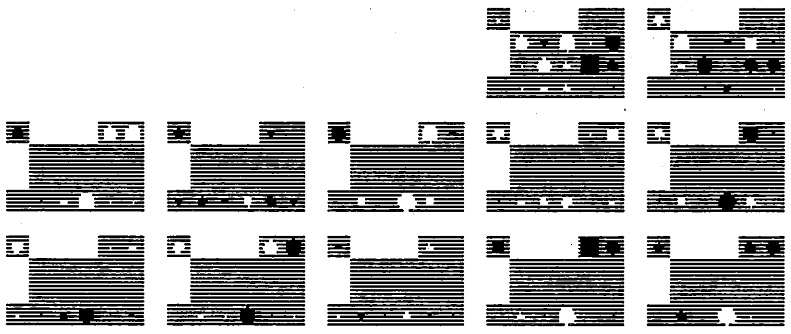
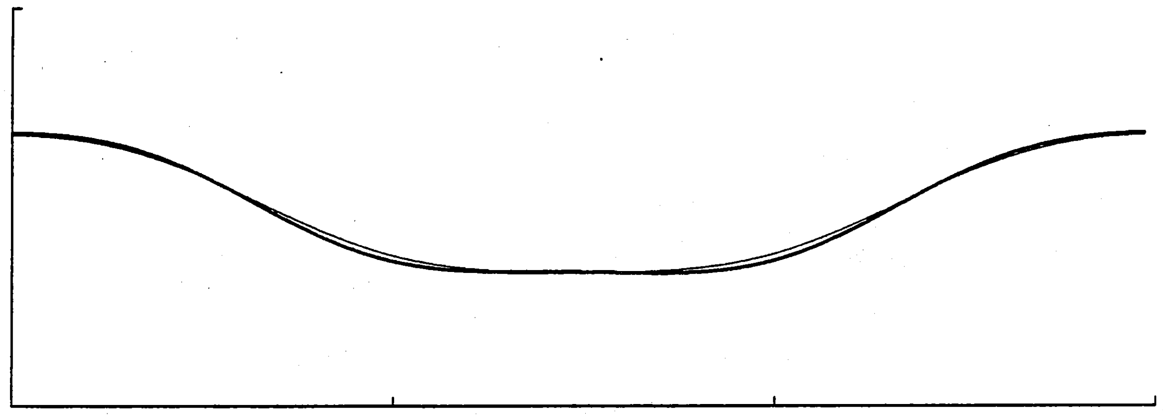


Figure 6-1: This network was trained on 298 samples taken uniformly from the state space of a simulated arm. The inputs (from left to right) are the position, velocity and desired acceleration of the shoulder joint and the position, velocity and desired acceleration of the elbow joint. The magnitude of the largest weight is 9.63.

testing data	RMSS error
random set A	0.068
random set B	0.083
trajectory 6	0.067

Table 6-1: Network performance on various synthetic data sets. The random sets consist of 298 points chosen with a uniform distribution from phase space. Trajectory 6 involves moving both the shoulder and elbow joints through a sinusoid. The network's training set was random set A.


Torque vs. time for trajectory 1 joint 1

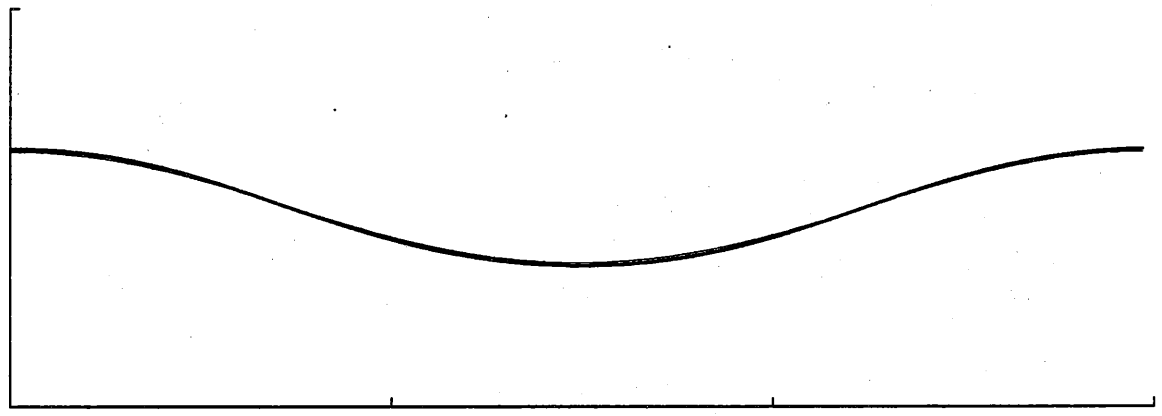

Torque vs. time for trajectory 6 joint 1

Figure 6-2: The network of figure 6-1 was used to generate torques to be applied in some simulated trajectories. The fine line is the actual required torque and the bold line is the torque output by the network.

References

[An 88] An, C., C. Atkeson, and J. Hollerbach.
Model-Based Control of a DD Arm, Part I: Building Models.
IEEE ICRA :1374-1379, 1988.

[Asada 82] Asada, H., T. Kanade, and I Takeyama.
Control of a DD Arm.
Technical Report CMU-RI-TR-82-4, Carnegie-Mellon University, April, 1982.

[Bejczy 74] Bejczy, A. K.
Robot Arm Dynamics and Control.
Technical Report 33-669, JPL, February, 1974.

[Brady 82] Brady, Michael, J. Hollerbach, T. Johnson, T. Lozano-Perez, and M. Mason.
Robot Motion: Planning and Control.
MIT Press, Cambridge, Mass., 1982.

[Canudas 87] Canudas, C. and Astrom, K.J. and Braun, K.
Adaptive friction compensation in DC-motor drives.
IEEE Journal of Robotics and Automation 3(6):681(5), December, 1987.

[Craig 86] Craig, J. J.
Adaptive Control of Mechanical Manipulators.
PhD thesis, Stanford University Dept of EE, 1986.

[Han 87] Han, J-Y., H. Hemami and S. Yurkovich.
Nonlinear Adaptive Control of an N-link Robot with Unknown Load.
IJRR 6(3), Fall, 1987.

[Hollerbach 80] Hollerbach, J. M.
A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity.
IEEE Transactions on Systems, Man and Cybernetics SMC-10(11):730-736, November, 1980.

[Kawato 87] Kawato, Mitsuo, Yoji Uno, Michiaki Isobe and Ryoji Suzuki.
Hierarchical Neural Network Model for Voluntary Movement with Application to Robotics.
In *Proceedings of the IEEE International Conference on Neural Networks.* IEEE, New York, NY, June, 1987.

[Khosla 86] Khosla, Pradeep K.
Real-Time Control and Identification of Direct-Drive Manipulators.
PhD thesis, Carnegie-Mellon University Department of EE, 1986.

[Lang 87] Kevin J. Lang.
Connectionist Speech Recognition.
Unpublished.
1987
CMU CS PhD Thesis Proposal.

[Lapedes 87] Alan Lapedes and Robert Farber.
Nonlinear Signal Processing Using Neural Networks: Prediction and System Modelling.
Technical Report, Theoretical Division, Los Alamos National Laboratory, 1987.

[Miller 87] Miller, W. Thomas III .
Sensor-Based Control of Robotic Manipulators Using a General Learning Algorithm.
IEEE Journal of Robotics and Automation 3, (2):157-166, April, 1987.

[Raibert 78] Raibert, M. H. and B. K. P. Horn.
Manipulator Control using Configuration Space Method.
Industrial Robot 5:69-73, June, 1978.

[Rumelhart 86] Rumelhart, D. E., Hinton, Geoffrey E., and Williams, R. J.
Learning internal representations by error propagation.
In D. E. Rumelhart, J. L. McClelland, & the PDP research group (editors), *Parallel distributed processing: Explorations in the microstructure of cognition.* Bradford Books, Cambridge, MA, 1986.

[Slotine 87] Slotine, J. E. and W. Li.
On the Adaptive Control of Robot Manipulators.
IJRR 6(3), Fall, 1987.

[Waibel 88] Waibel, A., T. Hanazawa, G. Hinton, K. Shikano and K. Lang.
Phoneme Recognition: Neural Networks vs. Hidden Markov Models.
In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*. IEEE, New York, NY, 1988.

[Yen 87] Yen, V. and M. L. Nagurka.
A Fourier-Based Optimal Control Approach for Structural Systems.
Technical Report CMU-RI-TR-87-12, Carnegie-Mellon University, September, 1987.

[Zhang 87] Zhang, Jinxin and Lang, Shijun.
Adaptive Weighted Suboptimal Control for Linear Dynamic Systems having a Polynomial Input.
IEEE Transactions on Automatic Control 32, (12):1106-1111, December, 1987.