USING BACKPROPAGATION TO LEARN THE DYNAMICS OF A REAL ROBOT ARM

Technical Report AIP - 53

Ken Goldberg & Barak Pearlmutter

Department of Computer Science
Carnegie Mellon Universivery
Pittsburgh, Pa. 15213

July 1988

This research was supported by the Computer Sciences Division, Office of Naval Research and DARPA under Contract Number N00014-86-K-0678, and in part by National Science Foundation grant EET-8716324. Barak Pearlmutter is a Fannie and John Hertz Foundation Fellow. Reproduction in whole or in part is permitted for purposes of the United States Government. Approved for public release; distribution unlimited.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

3 DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release;
Distribution unlimited

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
AIP 53

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
Carnegie-Mellon University

6b. OFFICE SYMBOL (*If applicable*)

7a. NAME OF MONITORING ORGANIZATION
Computer Sciences Division
Office of Naval Research

6c. ADDRESS (*City, State, and ZIP Code*)
Department of Psychology
Pittsburgh, Pennsylvania 15213

7b. ADDRESS (*City, State, and ZIP Code*)
800 N. Quincy Street
Arlington, Virginia 22217-5000

8a. NAME OF FUNDING / SPONSORING ORGANIZATION
Same as Monitoring Organization

8b. OFFICE SYMBOL (*If applicable*)

9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
N00014-86-K-0678

8c. ADDRESS (*City, State, and ZIP Code*)

10 SOURCE OF FUNDING NUMBERS p4000ub201/7-4-86

PROGRAM ELEMENT NO	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO
N/A	N/A	N/A	N/A

11. TITLE (*Include Security Classification*)
USING BACKPROPAGATION TO LEARN THE DYNAMICS OF A REAL ROBOT ARM

12. PERSONAL AUTHOR(S)
Ken Goldberg and Barak Pearlmutter

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Technical	FROM 86Sept15 TO 91Sept14	1988 July	18

16. SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	neural networks, robot control, learning manipulator dynamics

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

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 CMU Direct-Drive Arm II for a family of pick and place trajectories. Trained on a random sample of these trajectories, the network is shown to generalize top new samples drawn from the same family. The weights developed during the learning phase are reminiscent of the velocity and acceleration filters used in standard control theory.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS

21 ABSTRACT SECURITY CLASSIFICATION

22a NAME OF RESPONSIBLE INDIVIDUAL
Dr. Alan L. Meyrowitz

22b TELEPHONE (*Include Area Code*)
(202) 696-4302

22c. OFFICE SYMBOL
N00014

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Unclassified

1. Introduction

Today's robot arms sacrifice speed for flexibility. Coupled degrees of freedom are necessary to orient tools in the workspace, but existing commercial controllers make no attempt to compensate for the highly non-linear dynamics introduced by such coupling. This work applies neural networks, in particular back propagation, to this task. By learning the dynamics of a robot arm de novo, we hope to compensate for dynamic effects that are difficult to model or identify using conventional techniques.

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 dynamics tell us what torques to apply.

In the absence of external disturbances, a perfect model of the system dynamics can generate a perfect torque signal to achieve perfect arm motion. In the presence of noise and uncertainty, a more practical goal is to use an approximate model to generate an approximate torque and then use feedback 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 controllers at each joint and a second-order model to compensate for inter-joint coupling [Asada 82]. Yet finding an approximate 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 actuating torques, but faces two essential problems: the computational complexity of the model requires several hundred multiplications per cycle, and the analytic model does not always accurately 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 to 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 terms 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 we see no reason why unmodeled effects should 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 naive controller. Input torques can later be indexed by desired output state to generate feedforward torque. 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; the state space is only sampled along the desired trajectory and 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. For instance, given some samples $f(x_i,y_i) = z_i$, one can let $g(x,y) = \sum_{0 \le k<m} \sum_{0 \le l<n} a_{k,l} x^k y^l$ and choose the terms $a_{k,l}$ to minimize the sum squared error $E = \sum_i (f(x_i,y_i) - g(x_i,y_i))^2$. Unfortunately, one is placed on the horns of a dilemma. If the polynomial is of low order it does not have the flexibility to represent many 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 z_i's 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 (described below). 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 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 *weights* 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 formula $output = (1+e^{-input})^{-1}$.

It is thus tempting to apply neural networks to the domain of robot arm control. [Kawato 88] 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 explore the ability of a neural network to generalize within a specific family of trajectories, and we report some simulated results on training a neural network on the entire phase space of a manipulator.

2. Problem Definition

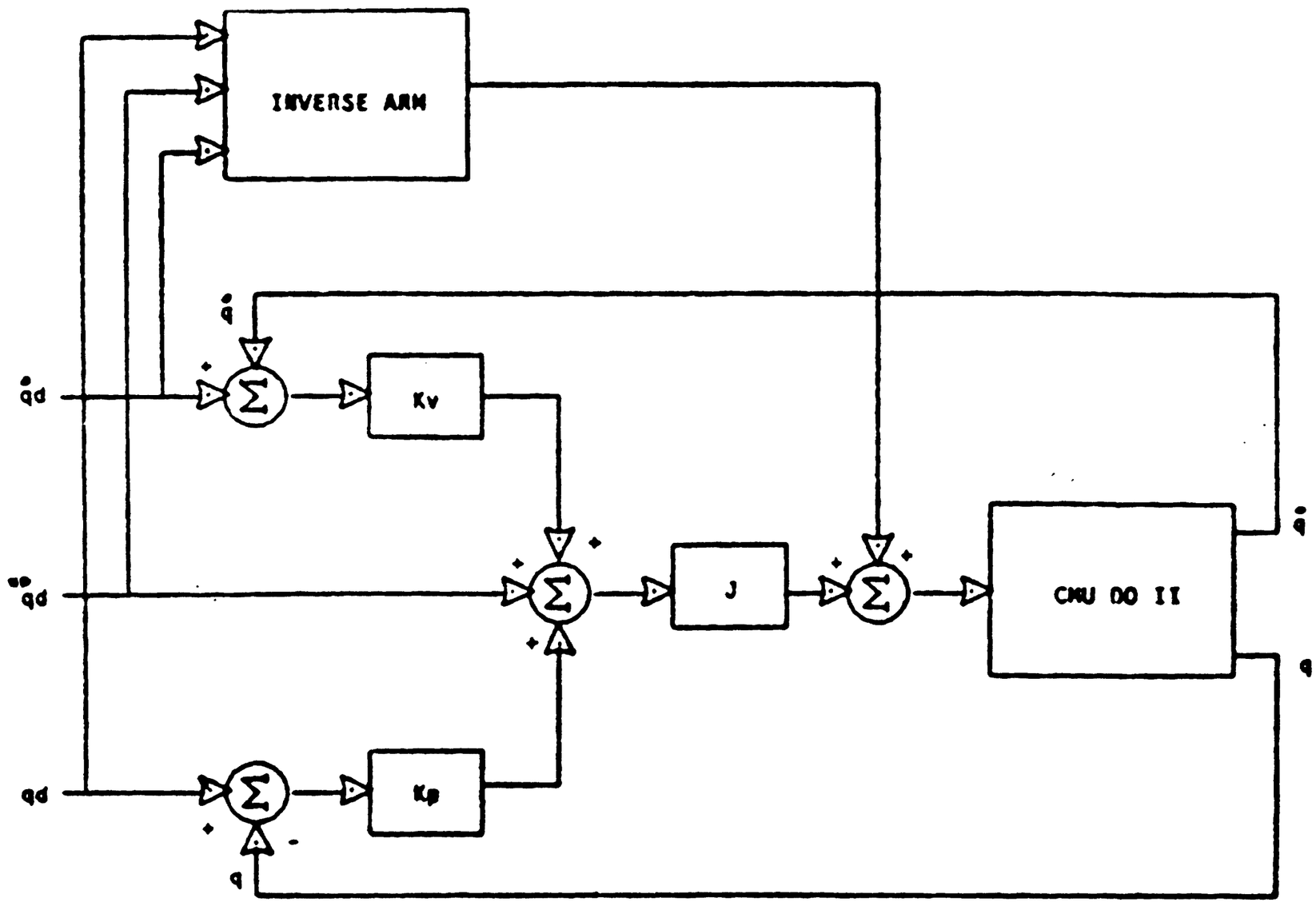


Figure 2-1: Block diagram of a *feedforward torque* controller, taken from [Khosla 86, page 114].

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 arm, or *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 avoid the L-E formulation and treat the arm as an unknown non-linear transfer function to be represented by weights of the network. As mentioned above, we must address the central problems associated with this approach: data generation and generalization.

To address these issues, we use a family of trajectories that is sampled to obtain a set of training trajectories and independently sampled to obtain a set of test trajectories. Specifically, we will focus on the family of pick and place trajectories that can be characterized by a fixed initial and final state and an intermediate or *via* position for each joint that can vary between 0 and 45 degrees. Each joint follows one half period of a scaled sinusoid. The peak amplitude is chosen independently for each joint, and velocities scale accordingly.

This paper focuses on the generalization problem: trained on a sample of this family of trajectories, how well does the neural network generalize to other members of the family?

2.1. Measurement of a Real 2 Link Arm

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. The DD arm was used with the kind and invaluable assistance of Pradeep Khosla. Other experiments were run on Lee Weiss' 2D direct drive arm, and we express our gratitude to him. Regrettably, logistic difficulties prevented us from including data gathered from his arm in this document.

We use a standard proportional controller to drive the first 2 links of the DD arm along 7 trajectories chosen randomly from our family of trajectories, recording the actual torques and positions. The first 5 of these are used to train a neural network to generate the actual torque profiles given the actual state profiles, and the performance of the network is tested on the last two trajectories by comparing the torque profiles that the network generates given the actual state trajectories with the torque profiles that were actually fed to the arm. The DD arm controller runs at an internal sample rate of 2 msec., but torque and state samples are recorded every 10 msec. See appendix I for additional details.

2.2. Measurement of a Simulated Arm

In an effort to see if a network of the sort we are using is able to learn not just a small region of phase space, but all of phase space, we simulated the inverse dynamics of a simple 2 link arm using parameters from the real arm described in appendix I and the L-E formulation found in [Brady 82]. We took 298 samples chosen uniformly from joint position, velocity and acceleration space, used the model to generate the corresponding torques, and trained a network on this data. We then tested the network's generalization on an independent sample of 298 points from phase space.

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 $x(t-n\Delta t), \ldots, x(t), \ldots, x(t+m\Delta t)$ and 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.

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 can add thousands of dollars to 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).

An additional consideration is the simplification of the analysis phase. Backpropagation has demonstrated the ability to utilize multiple noisy sources of information, so we were confident of the ability of the network to assimilate such information were we to make it available. However, we knew that the resultant networks would be more difficult to analyze.

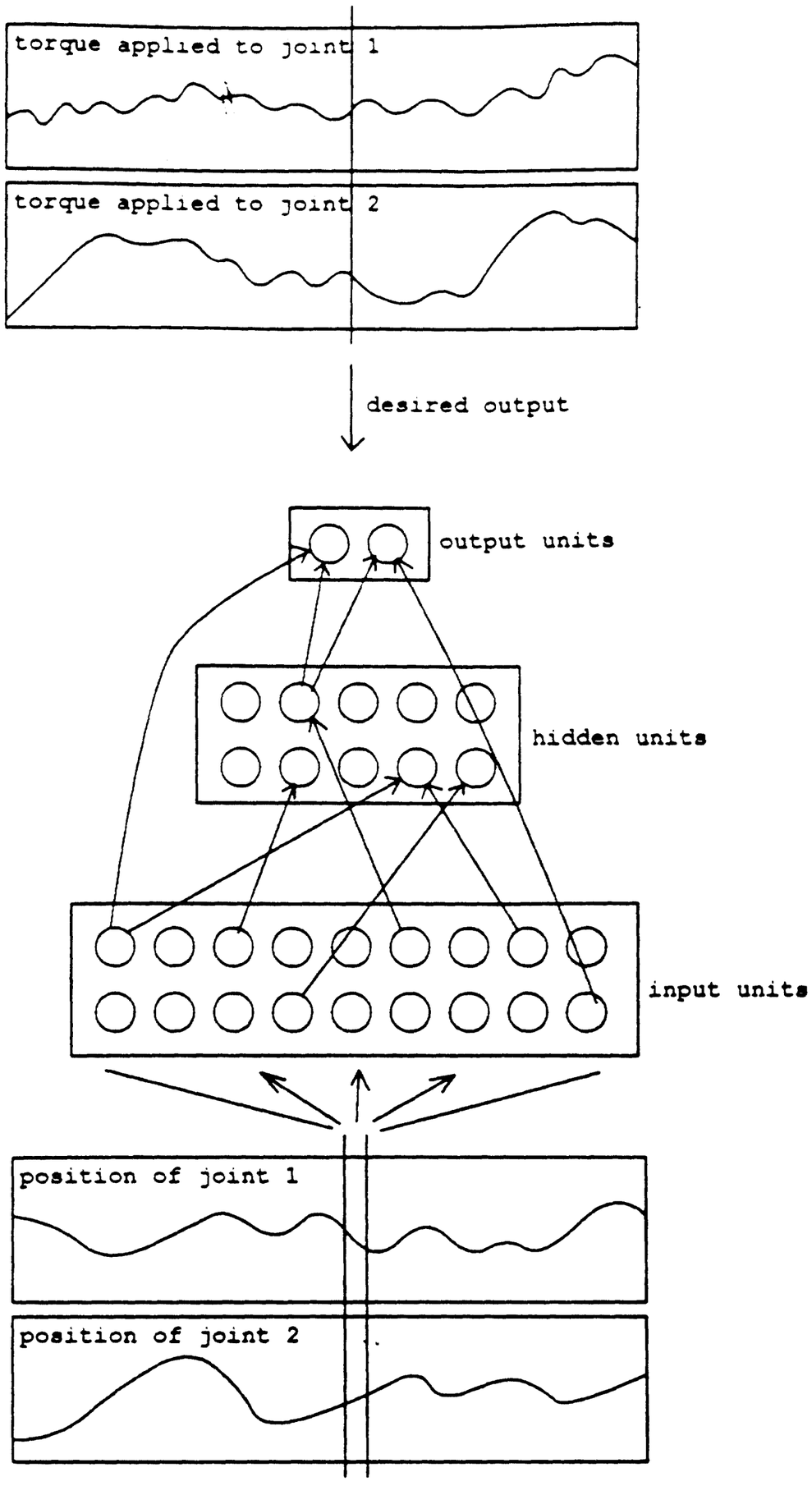


Figure 3-1: Backpropagation Network.

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 crack are relevant if the arm has some elasticity. By providing a window of position values, higher-order terms can be extracted by the network, as can evidence of phenomena like vibration that might be relevant.

3.2. Network Topology and Computational Complexity

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 death," 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

4.1. Actual Robot Arm

Plots of the actual torques overlayed with plots of the torques predicted by the network are shown in figure 4-1. We show here networks with three different window sizes: n=m=5, n=m=10, and n=m=20. By this definition, when we refer to a window of size five we mean a window centered at time t that includes five time steps before and five time steps after t, for a total of eleven.

Table 4-1 shows performance by networks of various window sizes on both the data they were trained on and some independent data drawn from the same distribution as the training data, the usual technique for testing generalization.

Window Size	Training Data	Test Trajectory E	Test Trajectory I
5	0.01296	0.04675	0.03573
10	0.00489	0.02224	0.02445
20	0.00435	0.02862	0.04074

Table 4-1: RMS errors of networks with different window sizes on both testing and training data.

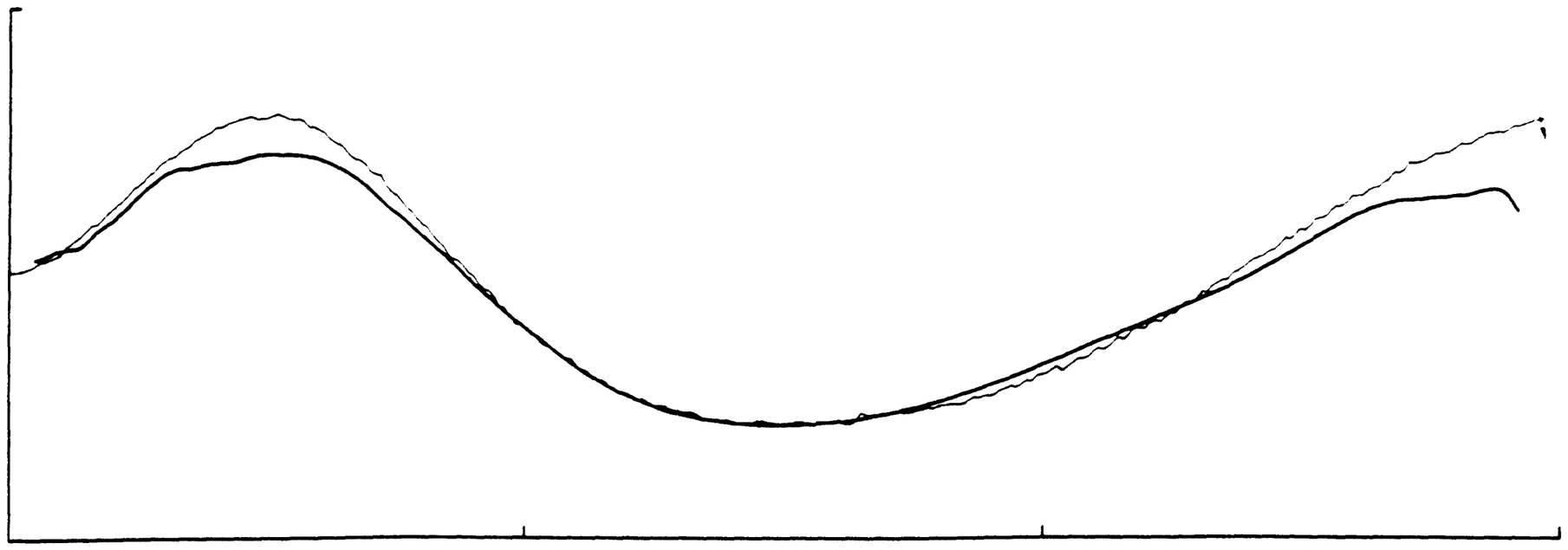
4.2. Simulated Arm

An advantage of a simulated arm over a real arm is that it is easy to uniformly sample phase space. We therefore used a simulated arm to obtain a training corpus of 298 samples of phase space and used them to train the network depicted in figure 5-9 to accept position, velocity and acceleration data and output appropriate torques. Training "to death" took 12,000 epochs.

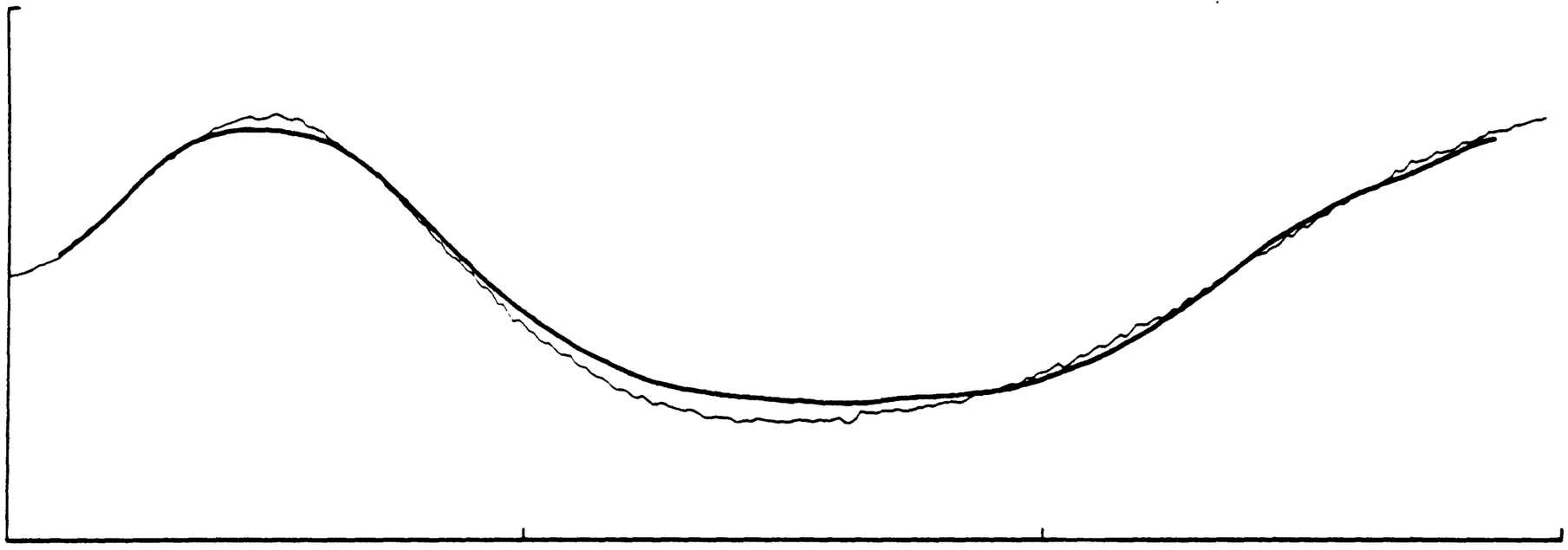
We tested the performance of this network on a test corpus also sampled uniformly from phase space,

---

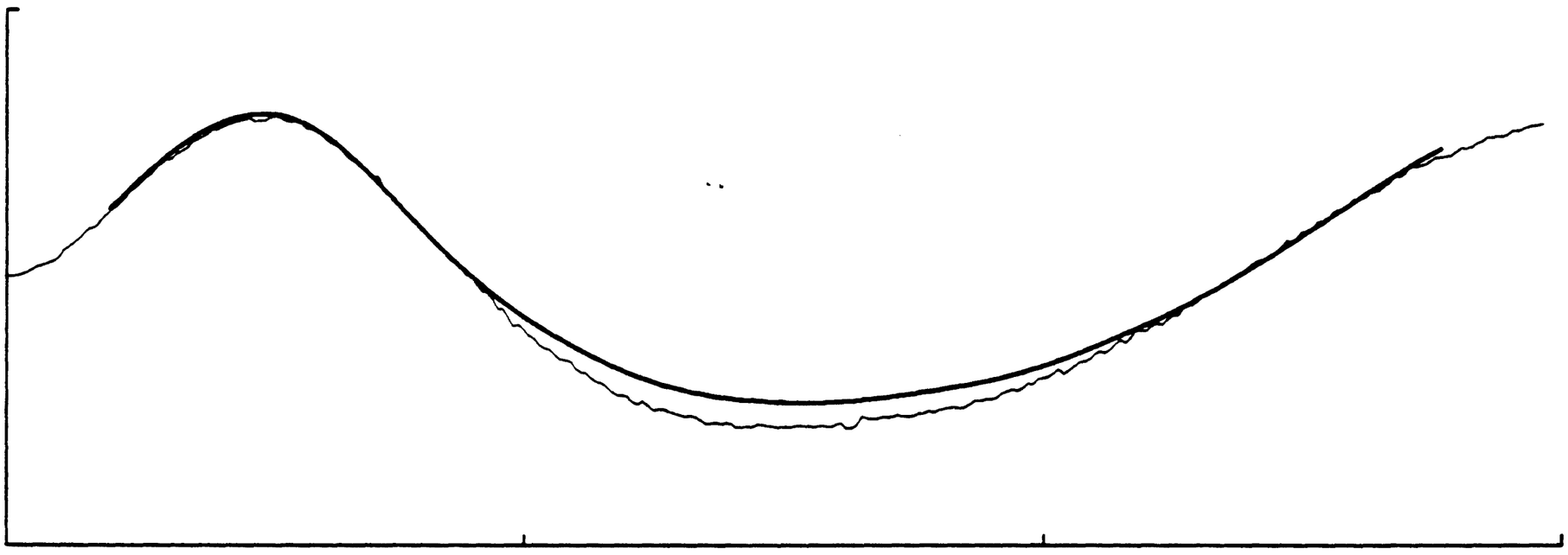
[1] In general, when training on a training corpus and testing using a different testing corpus, performance on the training corpus rises monotonically to an asymptote while performance on the test corpus first rises to a maximum and then falls to an asymptote. Many researchers, quite reasonably, use 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 5 (see figure 5-1)



Window Size 10 (see figure 5-2)



Window Size 20 (see figure 5-3)

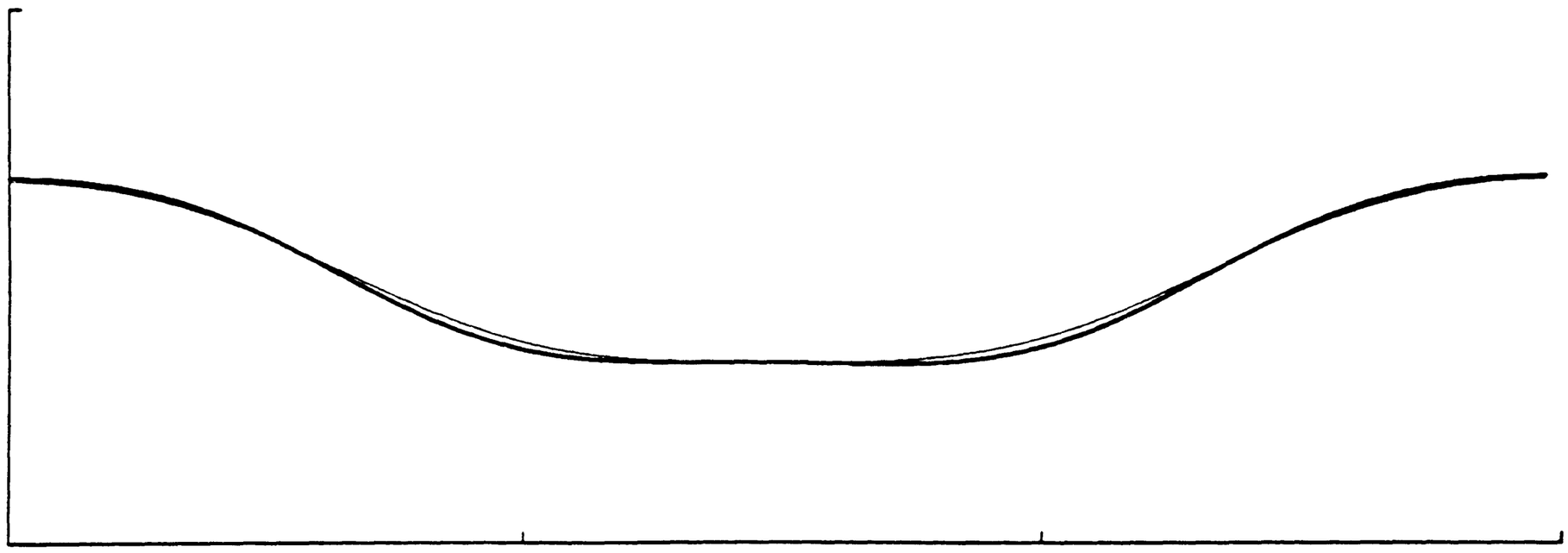
Figure 4-1: These graphs show torque profiles that are to drive joint 1 (the shoulder) through trajectory E, which was the most difficult of the test trajectories. The measured torque profile is drawn with a fine line, while the torque profiles generated by networks are drawn with bold lines.

thus testing generalization, and on a simulated trajectory, thus testing performance in an interesting region of phase space and giving a qualitative picture of network performance. The RMS errors are shown in table 4-2, and overlaid plots of the correct torques and the generated torques for the simulated trajectory

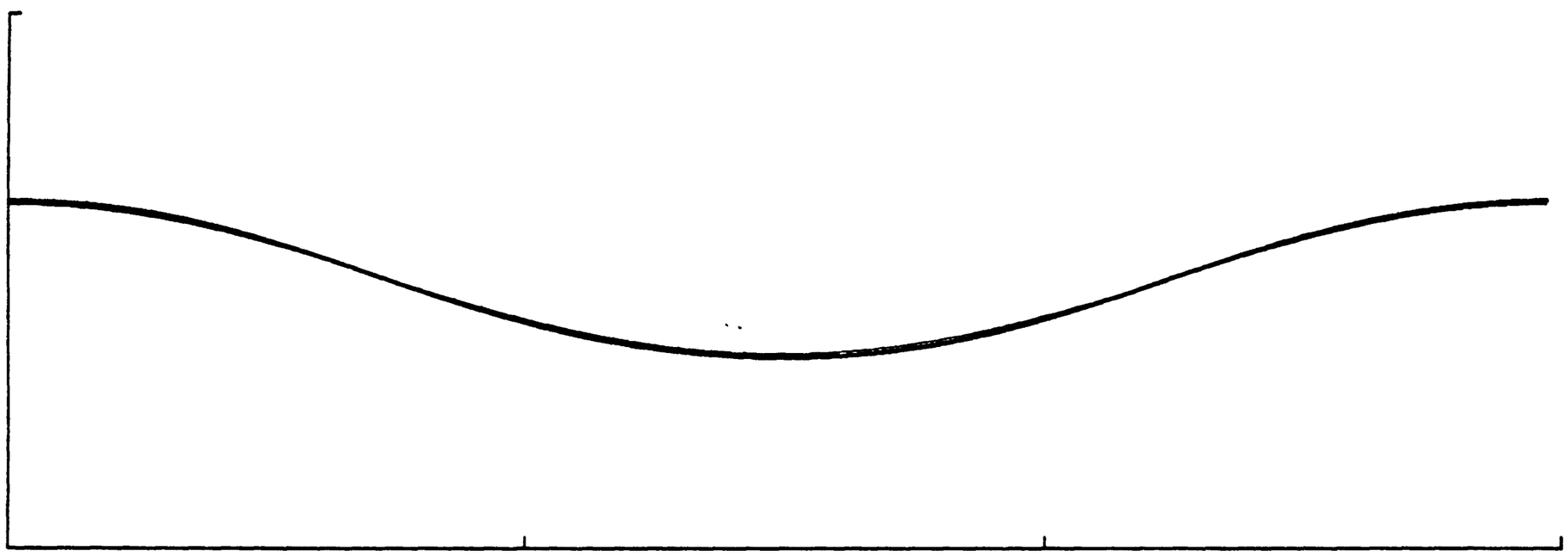
are shown in figure 4-2.

training set	random set A	random set B	trajectory E
random set A	0.98%	1.22%	0.69%
the ''zero'' set	14.37%	14.71%	10.20%

Table 4-2: Root mean square errors on various synthetic data sets. The random sets consist of 298 points chosen randomly with a uniform distribution from phase space. Trajectory E involves moving both the shoulder and elbow joints through a sinusoid.



Torque for trajectory A joint 1



Torque for trajectory E joint 1

Figure 4-2: The network of figure 5-9 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.

5. Analysis

5.1. Explanation of Weight Displays

Figures 5-9, 5-1, 5-2, and 5-3 show the weights developed by the networks after being trained to death on their training corpora. 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 5-9, 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 the each of 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.

5.2. Real Robot Arm

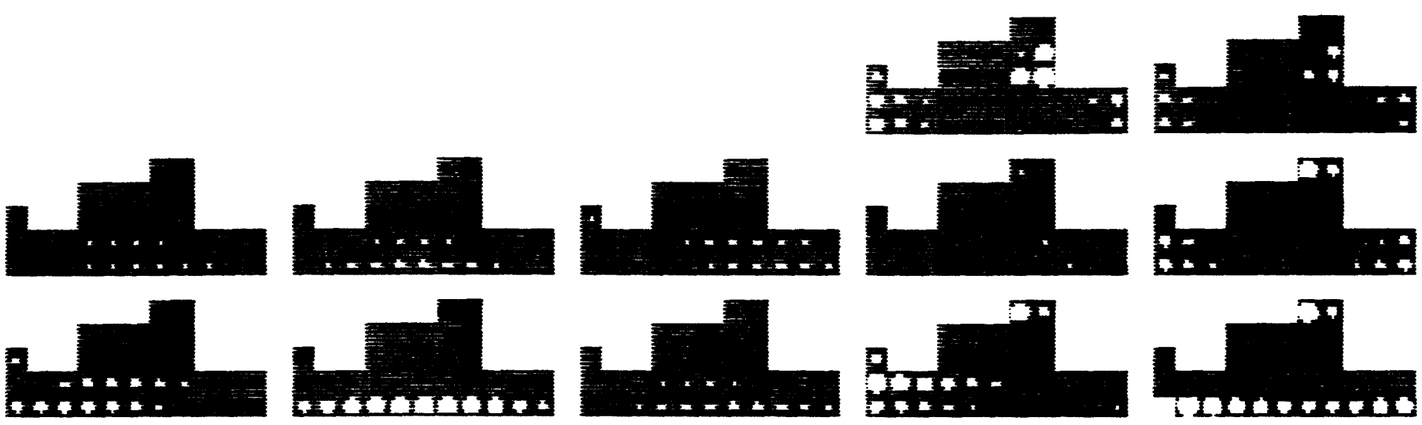


Figure 5-1: This network has a window size of 5. The largest weight has a magnitude of 18.9.

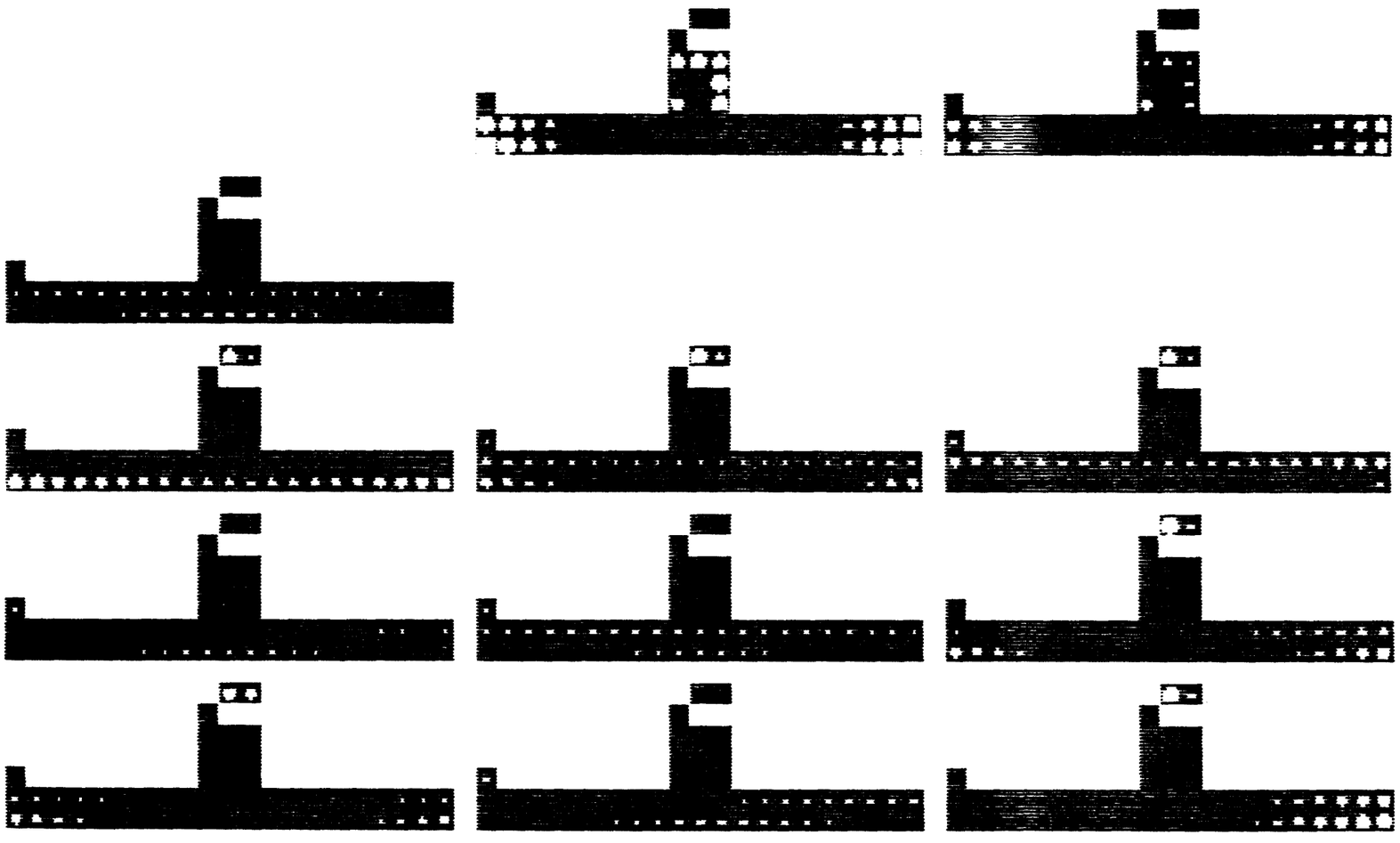


Figure 5-2: This network has a window size of 10. The largest weight has a magnitude of 11.7.

An attempt to figure out how the networks work yields some insights, although a complete understanding is probably impossible. The easiest things to interpret are the weights of connections from the input units, which form temporally smooth filters shaped to detect a linear combination of position,

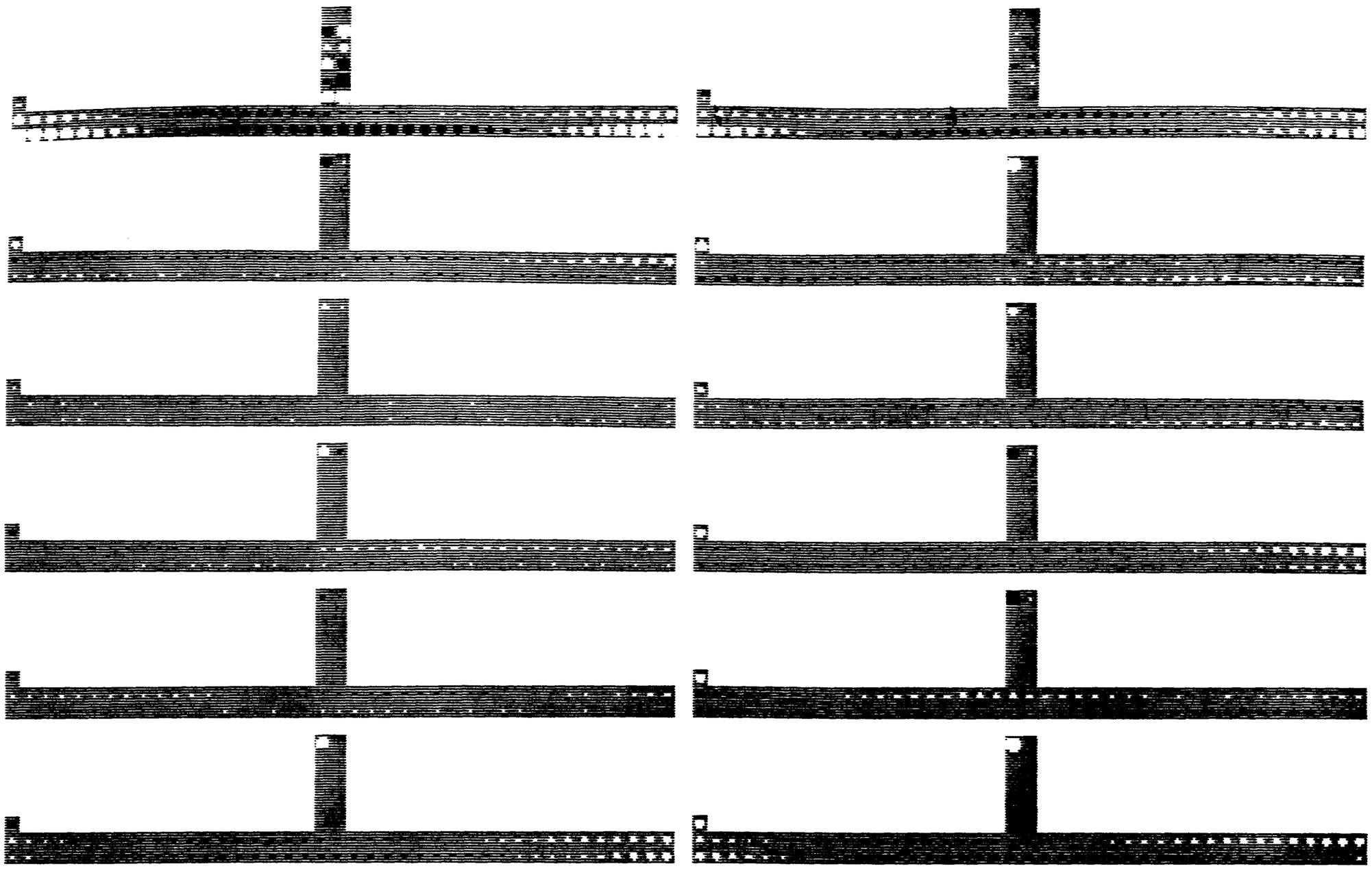


Figure 5-3: This network has a window size of 20. The largest weight has a magnitude of 4.2.

velocity, and acceleration. At first 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 had no built in notion of temporal adjacency, but developed these filters purely in order to map each input to the appropriate output. Because these filters are convolved with the position in every possible place, we should think of them as convolution functions and attempt to analyze them in those term.

5.3. Analysis of Filters

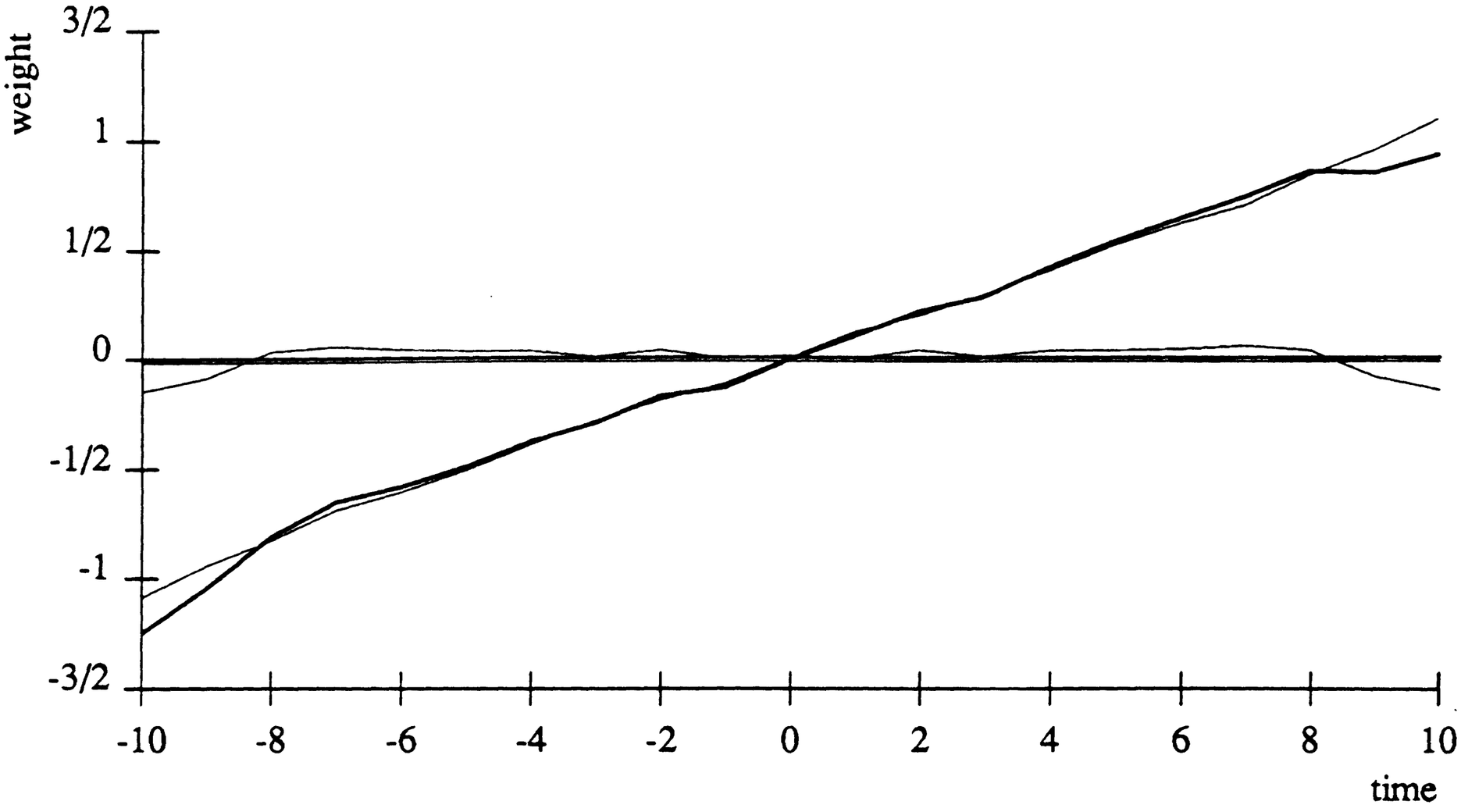
We can operate under the assumption that the filters developed by the network are the superpositions of simple position, velocity, and acceleration filters and attempt to decompose them. We therefore took the pattern of weights to a unit from the input layer, regarded these weights as samples from a continuous function which is being convolved with the position of the joint and decomposed this function $f(x), -w \leq x \leq w$ into a constant part, an odd part, and an even part using the equations

$$const = \int_{-w}^{w} f(x)$$

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

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

The resulting functions can be understood by inspection. For example, figure 5-4 shows that the inputs to unit 51 from joint 2 are can be understood as a simple velocity filter and figure 5-5 shows a unit whose inputs from joint 1 form a simple acceleration filter. More typically, in figure 5-6 we see a unit which is activated by a linear combination of velocity and acceleration.



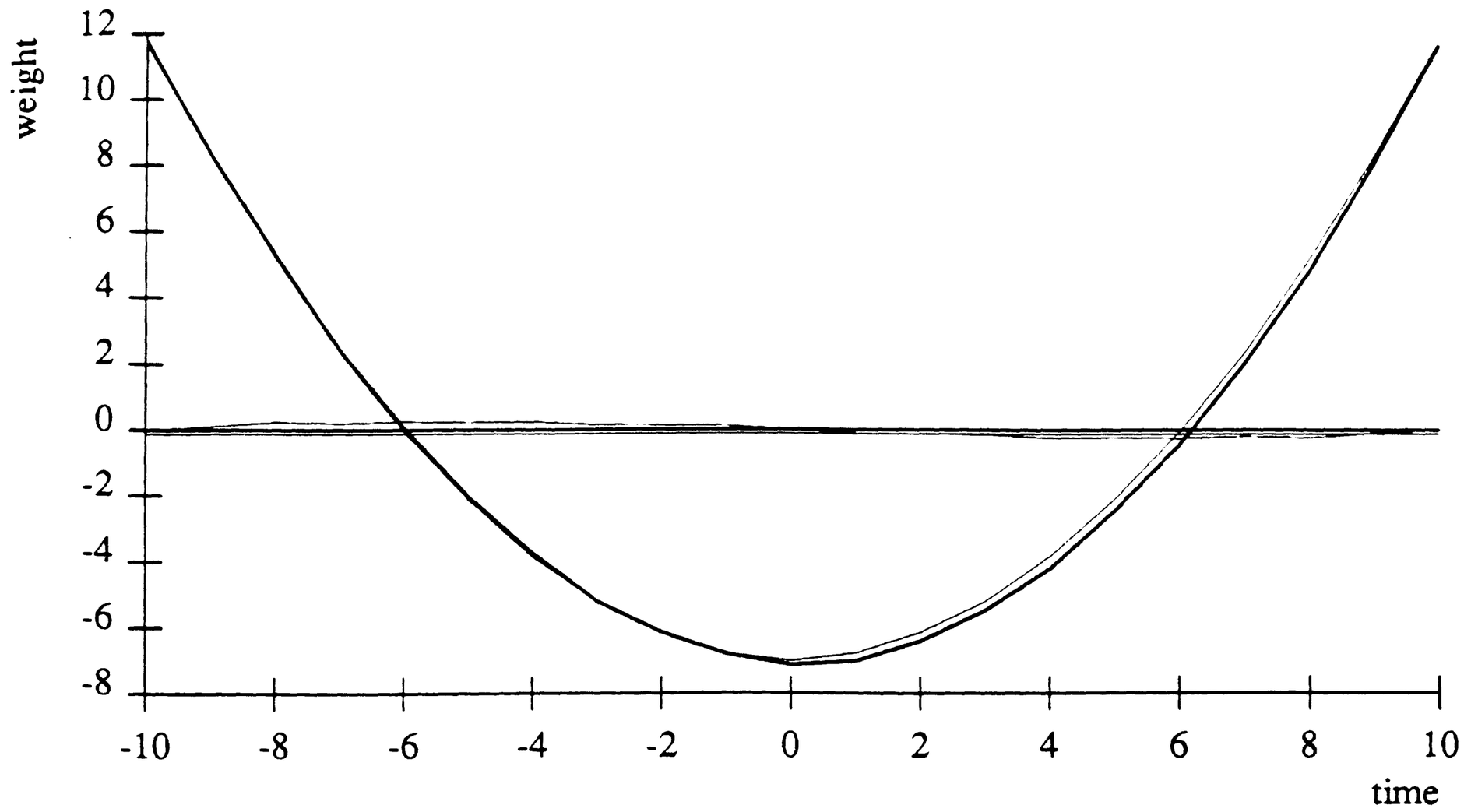
joint 2 to unit 51

Figure 5-4: A graph of the inputs to a unit regarded as a convolution function and decomposed into constant, even, and odd components.

The functions we have examined so far have been extremely smooth. This is in general the case in the network with window size 10, but in the network with window size 20 a new phenomenon appears. In figure 5-7 we see some curves with rough edges. in the section on window size vs. generalization below, we advance an explanation for this odd behavior.

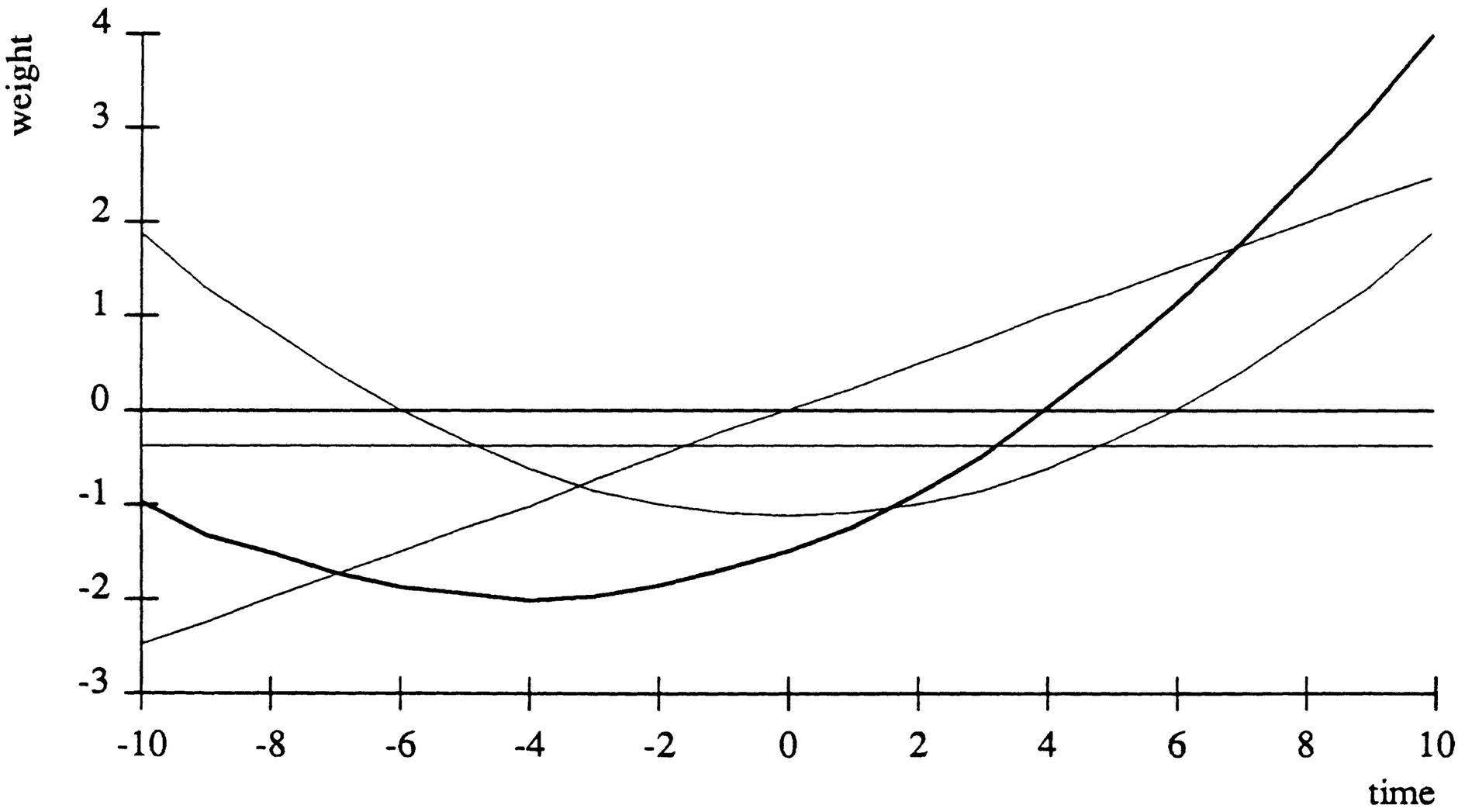
In figure 5-7, the odd component crosses below zero near the edges of the window, evidence that this unit responds not only to velocity and acceleration, but also to jerk (the derivative of acceleration).

It might be objected that any function can be decomposed into even and odd components, and that our analysis therefore is fallacious. However, although any function be can so decomposed, such a decomposition does not typically yield smooth intuitively interpretable curves. For instance, in figure 5-8 we decompose a filter which does not seem explainable in these terms. In further support of this claim, we should point out that in most of the filters we examined the constant term was so small that it could not be distinguished from the X axis.



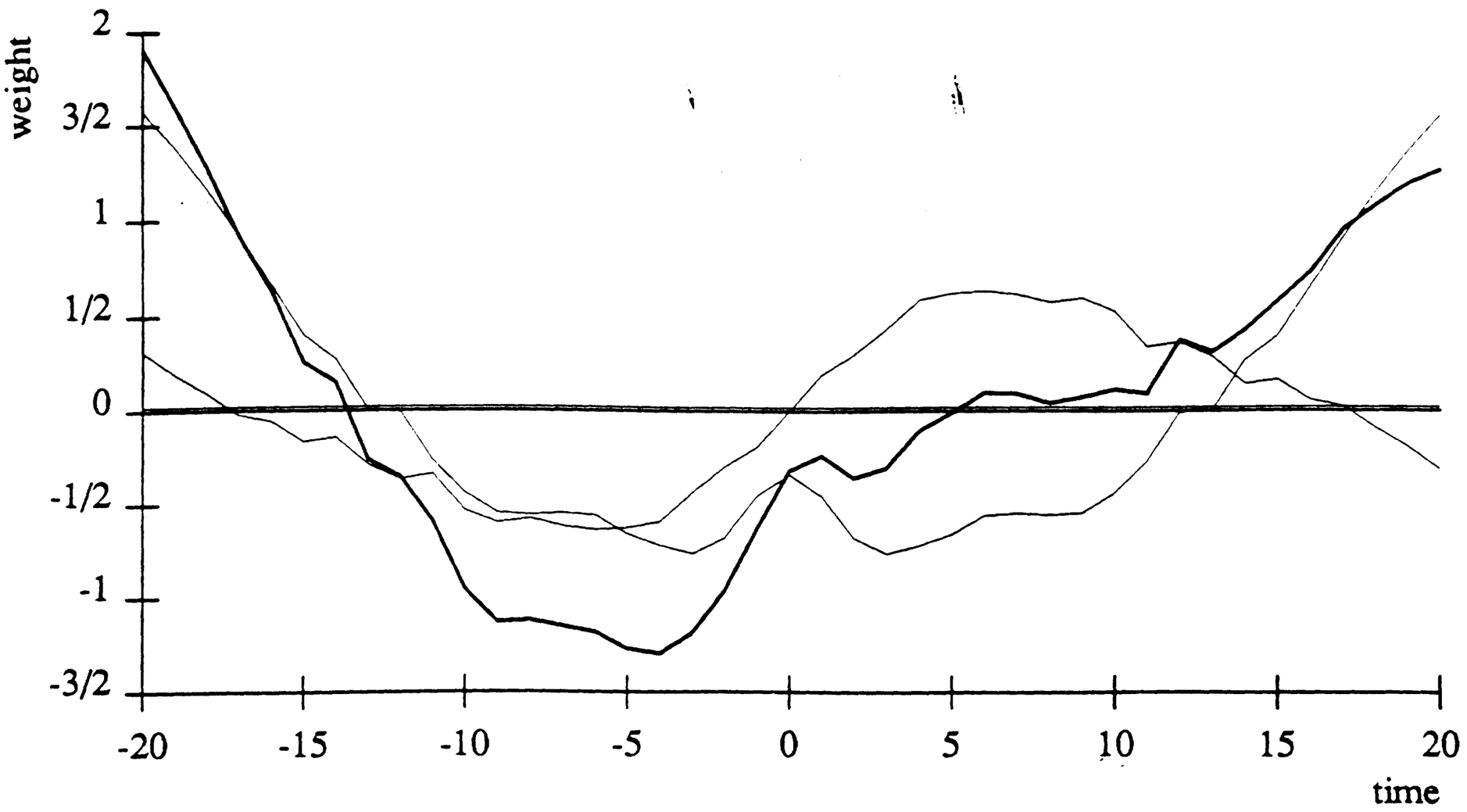
joint 1 to unit 53

Figure 5-5: A graph of the inputs to a unit regarded as a convolution function and decomposed into constant, even, and odd components.



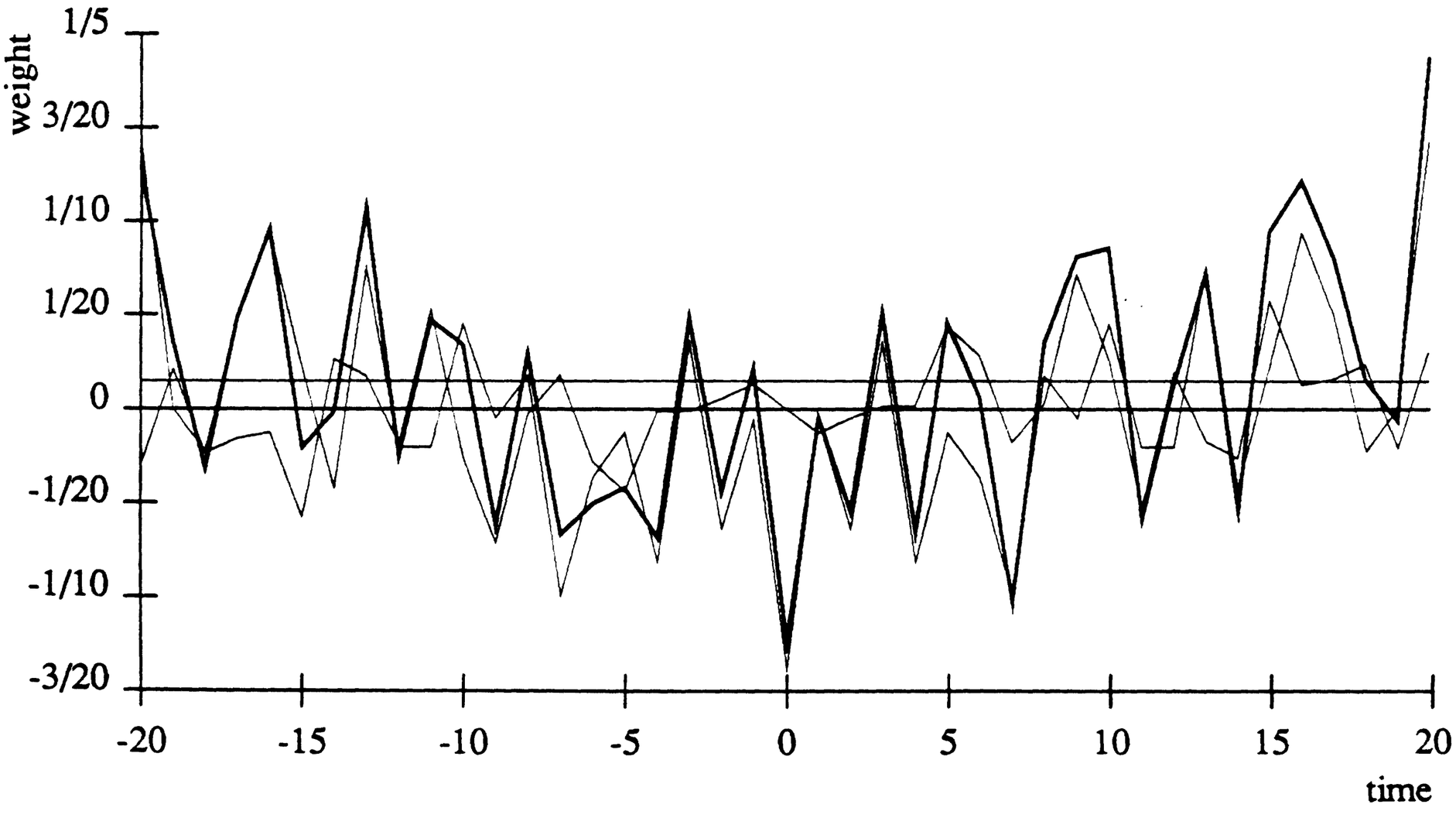
joint 2 to unit 52

Figure 5-6: A graph of the inputs to a unit regarded as a convolution function and decomposed into constant, even, and odd components.



joint 2 to unit 93

Figure 5-7: A graph of the inputs to a unit regarded as a convolution function and decomposed into constant, even, and odd components.



joint 1 to unit 85

Figure 5-8: A graph of the inputs to a unit regarded as a convolution function and decomposed into constant, even, and odd components.

5.4. A More Global Perspective

The roles of the individual hidden units is much more difficult to fathom. Since the weights are quite large, most units are saturated most of the time. Each unit has a transition point at which it is not saturated which is reached only under rare circumstances. For example, a unit might respond to $3.4x_1 + 1.2dx_1/dt - 2.1x_2$, being effectively saturated at 0 if this value is less than 2.3 and at 1 if the value is greater than 2.6, and having a non-binary 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 delicately under various circumstances. In a word, the networks are not modular: it is difficult to understand the roles of the various units in isolation.

5.5. 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 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.

5.6. Simulated Robot Arm

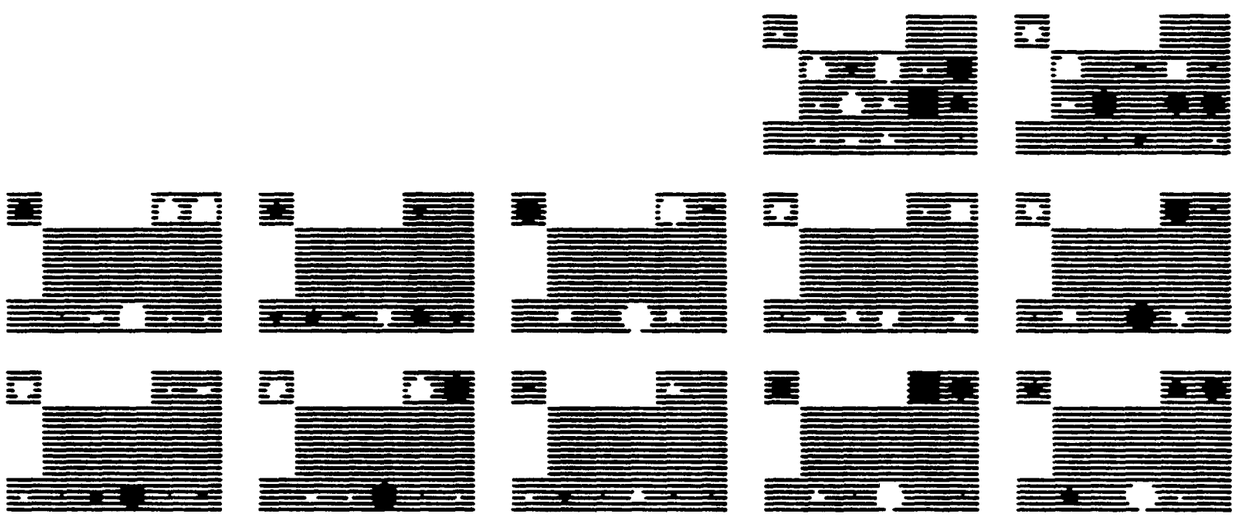


Figure 5-9: This network was trained on 298 samples taken uniformly from the phase space of a simulated arm. The inputs (from left to right) are the position, velocity and desired acceleration of the shoulder joint and the position, velocity and desired acceleration of the elbow joint. The magnitude of the largest weight is 9.63.

It is interesting to try to figure out how the network in figure 5-9 is performing its task. The hidden units each seem to be sensitive to only a particular range of velocities of the shoulder joint, and the position of the shoulder is (properly) ignored.

Quantitative measures of the performance of this network, both its training set and on some testing sets is shown in table 4-2. In that table, the figures for the "zero" set show the relative difficulty of the

task; these figures show the performance of a system which simply measures the mean value of each output in the environment and always outputs that. The excellent performance on random set B indicates that the network has generalized well from its sample of 298 points. The performance on trajectory E, as well as the graphs shown in figure 4-2, show that the network performs well in that portion of phase space which the trajectories lie in. This is strong evidence that neural networks of this sort should be able to learn the dynamics of an actual robot arm over the entirety of its phase space.

6. Conclusion

Backpropagation seems good at identification in this domain, although it is somewhat data-hungry compared to identification techniques tailored specifically to the plant in use. Particularly encouraging is the absence of spikes or oscillations. The fact that the network is able to develop smooth temporal filters without prior knowledge of the temporal ordering of its inputs is also very encouraging.

6.1. Future Work

We have shown that a three-layer backpropagation network is capable of generating accurate feedforward torques in an offline mode for a limited family of pick and place trajectories. 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 involve substantially greater portions of phase space.

We would like to test the neural network on the 3rd and 4th joint of the DD arm, and move up from 2 to 3 dimensions. The extra links will complicate the dynamic interactions and test the robustness of the neural network architecture. Although most pure control and identification techniques scale up simply from 2D to 3D, there is a possibility that learning the 3D dynamics will be much more difficult for backpropagation than learning the 2D dynamics, and checking this will be quite important to the ultimate usefulness of this approach.

Lastly, we would like to integrate the neural network into the controller and construct on on-line version of the system.

I. Trajectories used

The five trajectories used in training were generated using the following maximum flexions of the joints.

shoulder joint	elbow joint
45.0	45.0
38.6	32.2
5.4	39.4
20.9	0.0
0.0	45.0

The test trajectories were generated with the following parameters.

shoulder joint	elbow joint	
39.5	14.4	"E"
0.8	19.5	"I"

18.9	39.3
35.0	39.9
35.2	9.7
38.6	14.2
39.4	24.9

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 et. al.
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 88] Kawato, Mitsuo, Yoji Uno, Michiaki Isobe and Ryoji Suzuki.
Hierarchical Neural Network Model for Voluntary Movement with Application to Robotics.
IEEE Control Systems Magazine 8(2):8(9), April, 1988.

[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.