Department of Electrical Engineering
College of Engineering and Applied Sciences
State University of New York at Stony Brook
Stony Brook, New York 11794-2350

An Illustration of Neuro-Control with Backpropagation of Utility

by

K. Wendy Tang

Technical Report # 711

March, 1995

An Illustration of Neuro-Control with Backpropagation of Utility

K. Wendy Tang
Department of Electrical Engineering
SUNY at Stony Brook, Stony Brook, NY 11794-2350.

ABSTRACT *Backpropagation of utility is one of the five methods for neuro-control. Its goal is to provide a series of control signals to maximize a utility function over time. In this paper, we demonstrate how to use the basic backpropagation and backpropagation through time algorithms as fundamental building blocks for backpropagation of utility. Basically, the algorithm is composed of three subnetworks, the action network, model network, and an utility network or function. Each of these networks includes a feedforward and a feedback component. Pseudo-computer codes for each component and a flow chart for the interaction of these components are included. To further illustrate the algorithm, we use backpropagation of utility for the control of a simple one-dimensional planar robot. We found that the success of the algorithm hinges upon a sufficient emulation of the dynamic system by the model network.* [1]

1 Introduction

Neurocontrol is defined as the use of neural networks to emit control signals for dynamic systems. Neural networks offer several advantages over conventional computing architectures [1]. Calculations are carried out in parallel yielding speed advantages and programming is done by training through examples. These networks are characterized by their *learning* and *generalization* capabilities and can be deployed as "black boxes" that map inputs to outputs with no explicit rules or analytic function [2]. The neural network "learns" the system model by training through a set of desired input-output patterns. Their inherently parallel architecture and trainability make neural networks attractive candidates for fast, real-time control with unknown dynamic models.

The most dominant form of neural networks used is the multi-layer backpropagation network [3, 4, 5]. It is a hierarchical design consisting of fully interconnected layers of

[1]The author acknowledges and appreciates discussions with and contributions from Paul Werbos. This research was supported by the National Science Foundation under Grant No. ECS-9407363. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

neurons [6]. The weights associated with each neuron are updated by taking the gradient of the total squared error with respect to the weights and performing a gradient search of the weight space [7]. Errors are propagated backwards through the network, hence the name *back-propagation.* Despite its popularity, the main drawback of the basic backpropagation algorithm is its slow convergence rate. Various efforts are made to increase the rate of convergence, e.g., the Delta-Bar-Delta Rule [8]. By incorporating memory from previous time periods into current outputs, Werbos developed a more sophisticated version of the basic backpropagation algorithm, known as *backpropagation through time* [9]. In our previous work, we found that by combining the backpropagation through time algorithm with the Delta-Bar-Delta rule, the neural network provides more robust and faster learning [10].

Almost all neural network applications in robot control involve the incorporation of one or more backpropagation (basic or through time) neural networks into the controller [11]-[21],[22] -[26]. Different approaches exist in the method of incorporating the neural network into the controller and of training and adaptation [11]. Among these approaches, there are five basic schemes: the *supervised control*, *direct inverse control*, *neural adaptive control*, *back-propagation of utility*, and *adaptive critic networks.* Werbos [27] provided a detailed summary of the five schemes including the pros and cons of each method. In this report, our objective is to illustrate the theory of Backpropagation of Utility through a simple example, the control of a 1-D planar robot.

This report is organized as follows: Section 2 is a review of all the basic equations for the backpropagation algorithms (both the *basic* and the *through time* versions), including the delta-bar-delta rule for speeding up convergence. A simple network with six neurons is used to illustrate the validity of the equations. Then in Section 3, we show how these equations are used as fundamental building blocks for the *Backpropagation of Utility* algorithm. Section 4 illustrates how the algorithm is used for the control of a 1-D planar robot. It includes first a failed attempt and later a successful one. Finally, conclusions and a summary are included in Section 5.

2 Basic Equations

In this section, we present the basic equations for backpropagation and backpropagation through time algorithms. A simple example is used to illustrate the validity of these equations. For expository convenience, we assume there are m inputs, n outputs, H hidden nodes, and T training samples. The training inputs are presented to the network as $X_i(t), \quad i = 1, \ldots, m, \quad t = 1, \ldots, T$ and the corresponding desired outputs are $Y_i(t), \quad i = 1, \ldots, n, \quad t = 1, \ldots, T$. The detailed of the algorithm can be found in [9]. For the reader's convenience, they are also summarized here.

2.1 Basic Backpropagation Algorithm

The backpropagation algorithm is simply a tool for calculating the derivative of a function. The network equations consist of the feedforward and feedback components. During the feedforward mode, the network calculates an estimated output $\hat{Y}$ as a function of the inputs and the weights associated with the neurons. An error function is then produced by comparing $\hat{Y}$ with the desired output Y. In the feedback mode, the gradients of this error with respect to the weight space are identified. Subsequently, the weights are updated through the steepest decent method. More specifically, for training samples, $t = 1, \ldots, T$, the feedforward equations are:

$$x_i(t) = X_i(t) \qquad 1 \leq i \leq m \tag{1}$$

$$\text{for } i = m+1 \text{ to } i = m+H+n,$$

$$\begin{cases} net_i(t) &= \sum_{j=1}^{i-1} W_{ij} x_j(t) \\ x_i(t) &= s(net_i(t)) \end{cases} \tag{2}$$

$$\hat{Y}_i(t) = x_{m+H+i}(t) \qquad 1 \leq i \leq n \tag{3}$$

The error of the network is obtained by comparing the actual and the desired outputs.

$$E = \sum_{t=1}^{T} E(t) = \sum_{t=1}^{T} \sum_{i=1}^{n} 0.5[\hat{Y}_i(t) - Y_i(t)]^2 \tag{4}$$

where $\hat{Y}_i(t)$ is the output of the neural network and $Y_i(t)$ is the desired outputs. This error is feedback to the network. The error gradient F_W_{ij} with respect to each weight, W_{ij} is calculated with the feedback equations:

For training samples, $t = 1, \ldots, T$, the feedback equations are:

$$F\_\hat{Y}_i(t) = \frac{\partial E}{\partial \hat{Y}_i(t)} = \hat{Y}_i(t) - Y_i(t) \qquad i = 1, \ldots, n \tag{5}$$

for $i = m + H + n$ to $i = m + 1$,

$$\begin{cases} F\_x_i(t) & = F\_\hat{Y}_{i-m-H}(t) + \sum_{j=i+1}^{m+H+n} W_{ji} * F\_net_j(t) \\ F\_net_i(t) & = s'(net_i) * F\_x_i(t) \end{cases} \tag{6}$$

$$F\_W_{ij} = \sum_{t=1}^{T} F\_net_i(t) * x_j(t) \qquad i, j = 1, \ldots, m + H + n \tag{7}$$

where $s(z)$ is the sigmoidal transfer function and $s'(z)$ is the derivative of $s(z)$. Also,

$$s(z) = 1/(1 + e^{-z}) \tag{8}$$

$$s'(z) = s(z) * (1 - s(z)) \tag{9}$$

Once F_W_{ij} (the gradient of E with respect to W_{ij}) is calculated, each weight is updated according to:

$$\text{New } W_{ij} = W_{ij} - \alpha * F\_W_{ij} \quad i, j = 1, \ldots, m + H + n \tag{10}$$

where α is a constant called the *learning rate.*

2.2 An Example

In this section, we use a small network to illustrate the equations presented in Section 2.1 and to demonstrate that the backpropagation algorithm is simply a tool for derivative calculation. Consider the network with $m = 1$ input node, $n = 1$ output node, and two hidden layers each with two neurons. The network is layered and there is no connections for nodes of the same layer. The network with the weights associated at each node is shown in Figure 1.

According to Equations 2, the feedforward equations are:

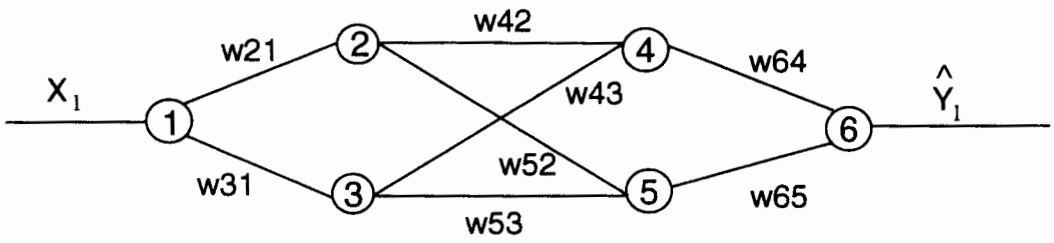

Figure 1: *A Six-Node Two-Layer Network.*

$$
\begin{array}{ll}
x_1(t) = X_1(t) & \\
net_2(t) = W_{21} * x_1(t) & x_2(t) = s(net_2(t)) \\
net_3(t) = W_{31} * x_1(t) & x_3(t) = s(net_3(t)) \\
net_4(t) = W_{42} * x_2(t) + W_{43} * x_3(t) & x_4(t) = s(net_4(t)) \\
net_5(t) = w_{52} * x_2(t) + W_{53} * x_3(t) & x_5(t) = s(net_5(t)) \\
net_6(t) = W_{64} * x_4(t) + W_{65} * x_5(t) & x_6(t) = s(net_6(t)) \\
\hat{Y}_1(t) = x_6(t) &
\end{array}
$$

The idea of backpropagation is to adjust the weights $W_{21}, W_{31}, \ldots, W_{64}, W_{65}$ such that $\hat{Y}(t)$ is close to the desired $Y(t)$. To achieve this goal, we need to find the gradient of the error function with respect to the weights. From Equation 4, the error function is:

$$E = \sum_{t=1}^{T} [\hat{Y}_1(t) - Y_1(t)]^2$$

Using the Equations 6, for $t = 1, \ldots, T$, the backward equations are:

$$
\begin{array}{l}
F\_\hat{Y}_1(t) = \hat{Y}_1(t) - Y_1(t) \\
F\_x_6(t) = F\_\hat{Y}_1(t) \\
F\_net_6(t) = s'(net_6) * F\_x_6(t) \\
\\
F\_x_5(t) = W_{65} * F\_net_6(t) \\
F\_net_5(t) = s'(net_5) * F\_x_5(t) \\
\\
F\_x_4(t) = W_{64} * F\_net_6(t) \\
F\_net_4(t) = s'(net_4) * F\_x_4(t) \\
\\
F\_x_3(t) = W_{43} * F\_net_4(t) + W_{53} * F\_net_5(t) \\
F\_net_3(t) = s'(net_3) * F\_x_3(t) \\
\\
F\_x_2(t) = W_{42} * F\_net_4(t) + w52 * F\_net_5(t) \\
F\_net_2(t) = s'(net_2) * F\_x_2(t) \\
\\
F\_x_1(t) = W_{21} * F\_net_2(t) + W_{31} * F\_net_3(t) \\
F\_net_1(t) = s'(net_1) * F\_x_1(t)
\end{array}
$$

Once the F_net_i are known, the error gradient with respect to the different weights are:

$$F\_W_{21} = \sum_{t=1}^{T} F\_net_2(t) * x_1(t)$$

$$F\_W_{31} = \sum_{t=1}^{T} F\_net_3(t) * x_1(t)$$

$$F\_W_{42} = \sum_{t=1}^{T} F\_net_4(t) * x_2(t)$$

$$\vdots$$

$$F\_W_{64} = \sum_{t=1}^{T} F\_net_6(t) * x_4(t)$$

$$F\_W_{65} = \sum_{t=1}^{T} F\_net_6(t) * x_5(t)$$

The weights are then adjusted according to

$$\text{New } W_{ij} = W_{ij} - \alpha * F\_W_{ij}, \quad i,j = 1,2,\ldots,6$$

To verify that F_W_{ij} indeed corresponds to $\frac{\partial E}{\partial W_{ij}}$, we calculate the gradient directly from the forward equations. As an example, consider $\frac{\partial E}{\partial W_{21}}$,

$$\frac{\partial E}{\partial W_{21}} = \sum_{t=1}^{T} \frac{\partial E}{\partial \hat{Y}_1(t)} \frac{\partial \hat{Y}_1(t)}{\partial W_{21}} \tag{11}$$

where

$$\begin{aligned}
\frac{\partial E}{\partial \hat{Y}_1(t)} &= [\hat{Y}_i(t) - Y_i(t)] = F_Y_1(t) \\
\frac{\partial \hat{Y}_1(t)}{\partial W_{21}} &= \underbrace{\frac{\partial \hat{Y}_i(t)}{\partial net_6(t)}}_{s'(net_6(t))} \frac{\partial net_6(t)}{\partial W_{21}} \\
\frac{\partial net_6(t)}{\partial W_{21}} &= [\underbrace{\frac{\partial net_6(t)}{\partial x_4}}_{W_{64}} \frac{\partial x_4(t)}{\partial W_{21}} + \underbrace{\frac{\partial net_6(t)}{\partial x_5}}_{W_{65}} \frac{\partial x_5(t)}{\partial W_{21}}] \\
\frac{\partial x_4(t)}{\partial W_{21}} &= \underbrace{\frac{\partial x_4(t)}{\partial net_4(t)}}_{s'(net_4(t))} \frac{\partial net_4(t)}{\partial W_{21}} \qquad \frac{\partial x_5(t)}{\partial W_{21}} = \underbrace{\frac{\partial x_5(t)}{\partial net_5(t)}}_{s'(net_5(t))} \frac{\partial net_5(t)}{\partial W_{21}} \\
\frac{\partial net_4(t)}{\partial W_{21}} &= [\underbrace{\frac{\partial net_4(t)}{\partial x_2}}_{W_{42}} \frac{\partial x_2(t)}{\partial W_{21}} + \underbrace{\frac{\partial net_4(t)}{\partial x_3}}_{W_{43}} \frac{\partial x_3(t)}{\partial W_{21}}] \\
\frac{\partial x_2(t)}{\partial W_{21}} &= \underbrace{\frac{\partial x_2(t)}{\partial net_2(t)}}_{s'(net_2(t))} \underbrace{\frac{\partial net_2(t)}{\partial W_{21}}}_{x_1(t)} \qquad \frac{\partial x_3(t)}{\partial W_{21}} = \underbrace{\frac{\partial x_3(t)}{\partial net_3(t)}}_{s'(net_3(t))} \underbrace{\frac{\partial net_3(t)}{\partial W_{21}}}_{x_1(t)}
\end{aligned} \tag{12}$$

Hence substituting Equations 12 to Equation 11, we have

$$\begin{aligned}
\frac{\partial E}{\partial W_{21}} &= \sum_{t=1}^{T} \underbrace{F_Y_1(t)\, s'(net_6(t))}_{F_net_6(t)} \; [\, W_{64}\, s'(net_4(t))\, W_{42}\, s'(net_2(t))\, x_1(t) \\
&\quad + W_{65}\, s'(net_5(t))\, W_{52}\, s'(net_2(t))\, x_1(t)\,] \\
&= \sum_{t=1}^{T} [\, \underbrace{\overbrace{F_net_6(t)\, W_{64}}^{F_x_4}\, s'(net_4(t))}_{F_net_4(t)} W_{42} + \underbrace{\overbrace{F_net_6(t)\, W_{65}}^{F_x_5}\, s'(net_5(t))}_{F_net_5(t)} W_{52}\,]\, s'(net_2(t))\, x_1(t) \\
&= \sum_{t=1}^{T} \underbrace{[\, F_net_4(t)\, W_{42} \;+\; F_net_5(t)\, W_{52}\,]}_{F_x_2(t)}\, s'(net_2(t))\, x_1(t) \\
&= \sum_{t=1}^{T} \underbrace{F_x_2(t)\, s'(net_2(t))}_{F_net_2(t)}\, x_1(t) \\
&= \sum_{t=1}^{T} F_net_2(t)\, x_1(t) = F_W_{21}
\end{aligned}$$

The rest of F_W_{ij} can be proved similarly. From these equations, we can see that the backpropagation algorithm is simply a tool for calculating the gradient of the error function with respect to the weight space.

2.3 Backpropagation Through Time Algorithm

Backpropagation through time was first proposed by Werbos [9]. It is basically an extension of the basic backpropagation algorithm but consider also memory from previous time periods. Mathematically, this is implemented through the introduction of a second set of weights W'. In our version of the backpropagation through time algorithm, we associated a weight W' at each hidden and output node. A more general version that includes a W' for each connection can be found in [9]. In our version, the second equation of the feedforward equations (Eq 2) is replaced by:

$$net_i(t) = \sum_{j=1}^{i-1} W_{ij} x_j(t) + W'_i x_i(t-1), \quad m < i \leq m + H + n \tag{13}$$

And the second equation of the feedback equations (Equation 6) is replaced by:

$$\begin{aligned}
F_x_i(t) = F_\hat{Y}_{i-m-H}(t) + \textstyle\sum_{j=i+1}^{m+H+n} W_{ji} * F_net_j(t) \quad & + W'_i * F_net_i(t+1) \\
& i = m + H + n, \ldots, m + 1
\end{aligned} \tag{14}$$

For adaptation of the W',

$$\begin{array}{lll} F\_W_i' & = \sum_{t=1}^{T} F\_net_i(t) * x_i(t) & \\ \text{New } W_i' & = W_i' - \beta * F\_W_i' & i = m+1, \ldots, m+H+n \end{array} \tag{15}$$

where β is the constant learning rate for W'.

2.4 Delta-Bar-Delta Rule

To improve the convergence speed of the steepest decent/ascent method, Jacob proposed the delta-bar-delta algorithm [8]. Basically, the algorithm is a special case of the *Adaptive Learning Rate* (ALR) discussed in [27]. Every weight of the network is given its own learning rate and that the rates changes with time. According to [8], the learning rate update rule is:

$$\triangle\alpha_{ij}(t) = \begin{cases} \kappa & \text{if } \bar{\delta}_{ij}(t-1)\delta_{ij}(t) > 0 \\ -\phi\alpha_{ij}(t-1) & \text{if } \bar{\delta}_{ij}(t-1)\delta_{ij}(t) < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

where

$$\begin{array}{lcl} \delta_{ij}(t) & = & F\_W_{ij} \\ \bar{\delta}_{ij}(t) & = & (1-\theta)\delta_{ij}(t) + \theta\bar{\delta}_{ij}(t-1) \\ \alpha_{ij}(t) & = & \alpha_{ij}(t-1) + \triangle\alpha_{ij}(t) \end{array}$$

In these equations, $\delta_{ij}(t)$ is the partial derivative of the error with respect to W_{ij} at time t and $\bar{\delta}_{ij}(t)$ is an exponential average of the current and past derivatives with θ as the base and time as the exponent[8]. If the current derivative of a weight and the exponential average of the weight's previous derivatives possess the same sign, the learning rate for that weight is incremented by a constant κ. If the current derivative of a weight and the exponential average of the weight's previous derivatives possess opposite signs, the learning rate for the weight is decremented by a proportion ϕ of its current value [8].

As discussed in [14], we found the best result comes from a combination of the back-propagation through time algorithm with the delta-bar-delta rule. In this case, $\beta_i(t)$, the learning rate for W_i' also changes with time. More specifically,

$$\triangle\beta_i(t) = \begin{cases} \kappa & \text{if } \bar{\gamma}_i(t-1)\gamma_i(t) > 0 \\ -\phi\beta_i(t-1) & \text{if } \bar{\gamma}_i(t-1)\gamma_i(t) < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

where

$$\begin{aligned}\gamma_i(t) &= F\_W_i' \\ \bar{\gamma}_i(t) &= (1-\theta)\gamma_i(t) + \theta\bar{\gamma}_i(t-1) \\ \beta_i(t) &= \beta_i(t-1) + \triangle\beta_i(t)\end{aligned}$$

3 Backpropagation of Utility Algorithm

The backpropagation of utility was first proposed by Werbos [28]. The objective of the algorithm is to provide a set of *action* or *control signals* to a dynamic system to maximize a utility function over time. The utility function can be total energy, cost-efficiency, smoothness of a trajectory, etc. For expository convenience, we assume the notation $X(t)$ for system state at time t, $u(t)$ for the control signal, and $U(t)$ for the utility function which is usually a function of the system state.

The system is composed of three subsystems, an *Action* network, a *Model* network, and a *Utility* network, which can often be represented as a performance function. The Action network is responsible for providing the control signal to maximize the utility function. This goal is achieved through adaptation of the internal weights of the action network. Such adaptation is accomplished through steepest decent with iterations. For each iteration, there are the feedforward and feedback components. In the feedforward mode, the Action network outputs a series of control signals, $u(t), t = 1, \ldots, T$ whereas adaptation of the internal weights is accomplished through the feedback mode.

The Model network provides an exact emulation of the dynamic system in a neural network format. Its function is two folded: (i) in the feedforward mode, it predicts the system state $X(t+1)$ for a given system state $X(t)$ and control signal $u(t)$ at time t; and (ii) in the feedback mode, it inputs the derivative of the utility function $U(t)$ with respect to the system state $X(t)$ and outputs the derivative of the utility with respect to the control signal, i.e., $\frac{\partial U(t)}{\partial u(t)}$ which is used for the adaptation of the action network. The Utility network, on the other hand, provides a measure of the system performance $U(t)$ as a function of the system state, $X(t)$. In the feedforward mode, it calculates a performance value $U(t)$ and in the feedback mode, it identifies $\frac{\partial U(t)}{\partial X(t)}$ which is used by the Model network.

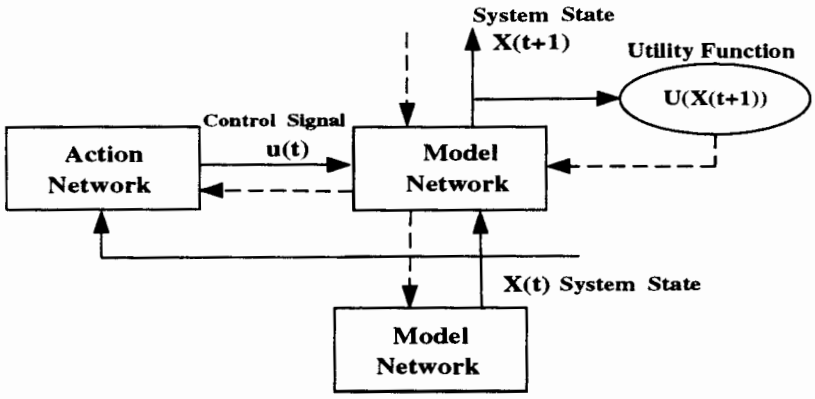

Figure 2: *A Backpropagation of Utility System.*

The basic idea is that assuming we have an exact model of the system formulated as a neural network (the Model network), we can use the backpropagation method to calculate the derivative of the utility function with respect to the control signal from the action network, i.e., $F\_u(t) = \frac{\partial U(t)}{\partial u(t)}$. Such derivative is then used to calculate the gradient of the Utility with respect to the internal weights of the action network. Figure 2 shows a block-diagram representation of the system. The dashed lines represent the feedback mode, or derivative calculations.

The successful application of backpropagation of utility hinges upon an accurate Model network that represents the system. The establishment of such a Model network is accomplished through training with the basic backpropagation and backpropagation through time algorithms. Once an accurate Model network is obtained, the internal weights of the Action network is adapted to output a series of desired control action, according to the flow chart in Figure 3. In this flow-chart, **Action, Model, Utility** represent the feedforward components of the corresponding networks whereas **F_Utility, F_Model, F_Action** are the feedback components. The details of the construction of the Model network and the adaptation of the Action networks are included in the following subsections.

3.1 Training of the Model Network

The establishment of a Model network that represents the system is accomplished through training with either the basic or the backpropagation through time algorithm combined with Jacob's delta-bar-delta rule discussed in Section 2.3.

First, a sufficient number of training samples, T_M must be obtained. These training samples consists of m_M inputs ($X_{Mi}(t), \quad i = 1, \ldots, m_M, \quad t = 1, \ldots, T_M$), and n_M desired outputs ($Y_{Mi}(t), \quad i = 1, \ldots, n_M, \quad t = 1, \ldots, T_M$). The objective of a trained Model network is to emulate the dynamic system. In the feedforward mode, it outputs the system state $X(t+1)$ at time $t+1$ for a given system state $X(t)$ and control signal $u(t)$ at time t. That is , $X_M(t)$ consists of $X(t)$ and $u(t)$ and $Y_M(t)$ is composed of the system state $X(t+1)$ at time $t+1$. For expository convenience, we assume there are H_M hidden nodes. A pseudo-code for training the Model network with backpropagation through time algorithm is presented in Table 1. The version for the basic algorithm can be obtained by setting $W' = 0$ for all nodes at all times.

3.2 Adaptation of the Action Network

Upon completion of training of the Model network, we are ready for the adaptation of the Action network. In this stage, we adapt the weights of the Action network to output a series of desired control action $u_i(t)$, $i = 1, \ldots, n$ for time period $t = 1, \ldots, T$. Again, the system state is $X_i(t)$, $i = 1, \ldots, m$ and the initial system state $X_i(t = 1)$ is known for all i. This adaptation process is accomplished through a number of iterations and is best described through the flow-chart shown in Figure 3.

There are basically six fundamental building blocks, **Action, Model**, and **Utility** in the feedforward mode; and **F_Utility, F_Model**, and **F_Action** in the feedback model. For each iteration, in the feedforward mode, a series of predicted control signals $u(t)$ for $t = 1, \ldots, T$ are provided by the **Action** routine. These control signals are inputs to the

Assume $m_M, n_M, N_M, H_M, T_M, X_M, Y_M$ as defined in Section 2.
For $i, j = 1, \ldots, N_M + n_M$

W_{ij}, are randomly distributed between ± 1, $\delta_{ij} = \bar{\delta}_{ij} = 0$.

Also, $W_{ij} = 0$ for nodes i, j in the same layer and is fixed for all iterations.

Initialize, $W'_i = \gamma_i = \bar{\gamma}_i = 0$ for $i = m_M + 1 \ldots, N_M + n_M$.

for (Iteration =1 to Maximum Iteration)

{

Step 1: for ($t = 1$ to $t = T_M$)

$$\begin{cases} \text{for } (i = 1 \text{ to } i = m_M) \quad x_{Mi}(t) = X_{Mi}(t) \\ \text{for } (i = m_M \text{ to } i = N_M + n_M) \\ \quad \begin{cases} net_{Mi}(t) &= \sum_{j=1}^{i-1} W_{Mij} x_{Mj}(t) + W'_{Mi} x_{Mi}(t-1) \\ x_{Mi}(t) &= s(net_{Mi}(t)) \end{cases} \\ \text{for } (i = 1 \text{ to } i = n_M) \quad \hat{Y}_{Mi}(t) = x_{M(m+H+i)}(t) \end{cases}$$

Step 2: Compute the Error, E_M.

$$E_M = \sum_{t=1}^{T_M} E_M(t) = \sum_{t=1}^{T_M} \sum_{i=1}^{n_M} 0.5[\hat{Y}_{M\,i}(t) - Y_{M\,i}(t)]^2$$

Step 3: for ($t = T_M$ to $t = 1$)

$$\begin{cases} \text{for } (i = 1 \text{ to } i = n_M) \quad F\_\hat{Y}_{M\,i}(t) = \hat{Y}_{M\,i}(t) - Y_{M\,i}(t) \\ \text{for } (i = N_M + n_M \text{ to } i = m_M + 1) \\ \quad \begin{cases} F\_x_{M\,i}(t) = & F\_\hat{Y}_{M\,(i-m-H)}(t) + \sum_{j=i+1}^{m+H+n} W_{M\,(ji)} * F\_net_{M\,j}(t) \\ & + W'_{M\,i} * F\_net_{M\,i}(t+1) \\ F\_net_{M\,i}(t) = & s'(net_{M\,i}) * F\_x_{M\,i}(t) \end{cases} \end{cases}$$

Step 4: For $i, j = 1, \ldots, N_M + n_M$, compute

$$(i)\ F\_W_{M\,(ij)} = \delta_{ij} = \sum_{t=1}^{T} F\_net_{M\,i}(t) * x_{M\,j}(t); \quad (ii)\ \triangle\alpha_{ij} = \begin{cases} \kappa & \text{if } \bar{\delta}_{ij}\delta_{ij} > 0 \\ -\phi\alpha_{ij} & \text{if } \bar{\delta}_{ij}\delta_{ij} < 0 \\ 0 & \text{otherwise.} \end{cases}$$

$$(iii)\ \bar{\delta}_{ij} = (1-\theta)\delta_{ij} + \theta\bar{\delta}_{ij} \quad \text{and} \quad (iv)\ \alpha_{ij} = \alpha_{ij} + \triangle\alpha_{ij}$$

Step 5: For $i = m + 1, \ldots, m + H + n$, compute

$$(i)\ F\_W'_{M\,i} = \gamma_i = \sum_{t=1}^{T_M} F\_net_{M\,i}(t) * x_{M\,i}(t); \quad (ii)\ \triangle\beta_i = \begin{cases} \kappa & \text{if } \bar{\gamma}_i\gamma_i > 0 \\ -\phi\beta_i & \text{if } \bar{\gamma}_i\gamma_i < 0 \\ 0 & \text{otherwise.} \end{cases}$$

$$(iii)\ \bar{\gamma}_i = (1-\theta)\gamma_i + \theta\bar{\gamma}_i \quad \text{and} \quad (iv)\ \beta_i = \beta_i + \triangle\beta_i$$

Step 6: New $W_{M\,(ij)} = W_{M\,(ij)} - \alpha_{ij} * F\_W_{M\,(ij)} \quad i, j = 1, \ldots, m + H + n$

New $W'_{M\,i} = W'_{M\,i} - \beta_i * F\_W'_{M\,i} \quad i = m + 1, \ldots, N_M + n_M$

}

Table 1: *A Pseudo-Code for Training of Model Network.*

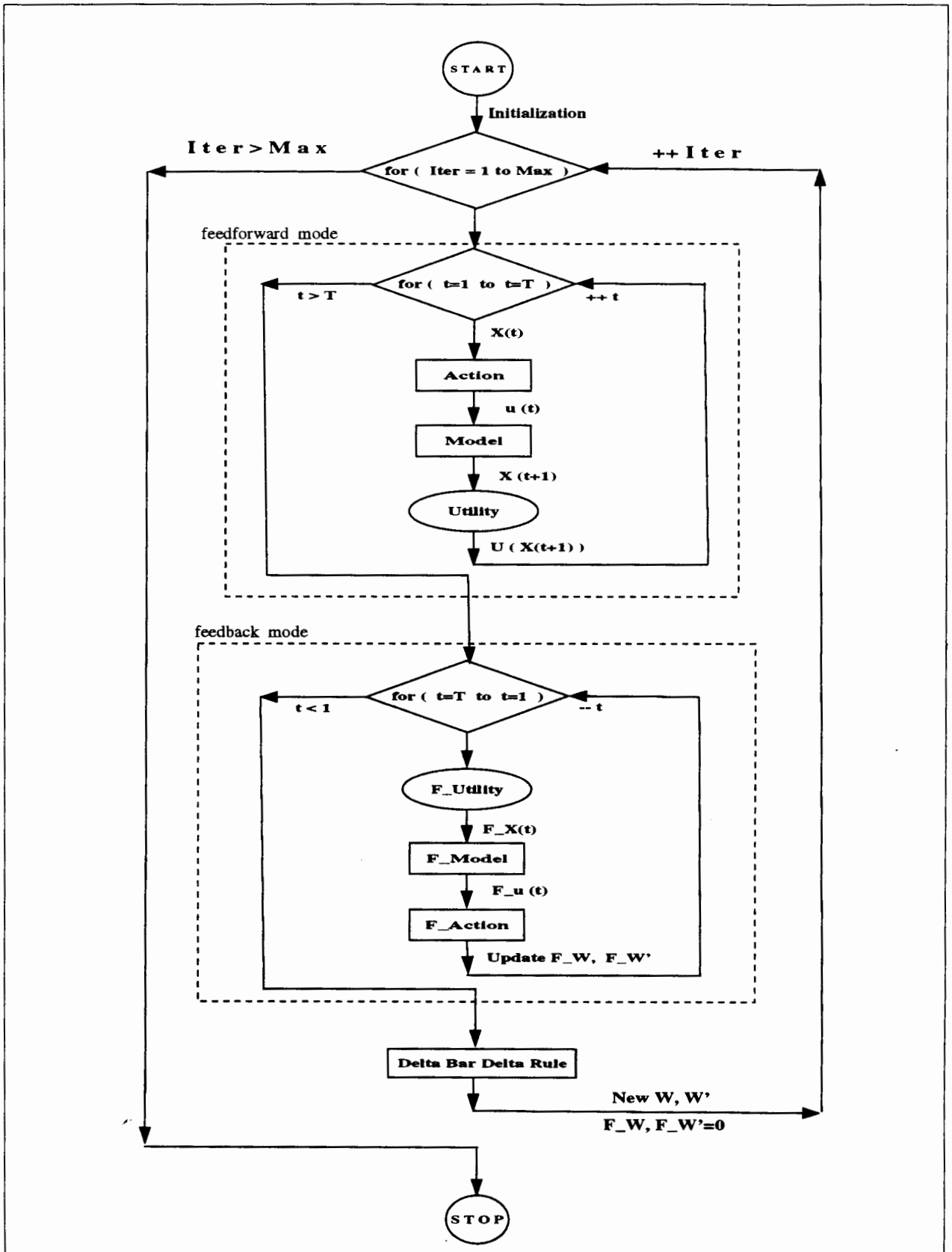

Figure 3: *A Flow-Chart for Adaptation of Action Network.*

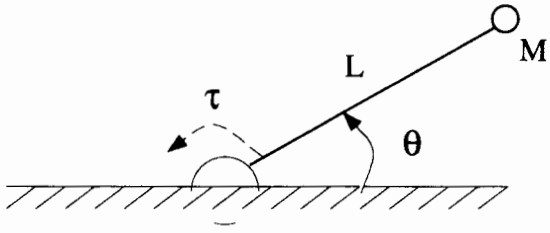

Figure 4: *A 1-D Planar Robot.*

Model routine which outputs the next system state $X(t+1)$. This system state is then used to calculate the Utility function.

In the feedback mode, the training samples are traversed backward. Since the Utility function is normally an explicit function of the system state, we can usually obtain $F_X(t) = \frac{\partial U(t)}{\partial X(t)}$ analytically. The value $F_X(t)$ is then input to the routine **F_Model** which corresponds to the feedback component of the Model network. **F_Action** is the next routine which takes the output $F\_u(t) = \frac{\partial U(t)}{\partial u(t)}$ from the **F_Model** routine to calculate the gradient of the Utility function with respect to the weight-space, i.e., $F\_W_{ij} = \frac{\partial U(t)}{\partial W_{ij}}$ and $F\_W'_i = \frac{\partial U(t)}{\partial W'_i}$ for all weights W_{ij} and W'_i of the Action network. Once the effect of all training samples are accounted for in F_W and F_W', delta-bar-delta rule is used to update the weights, W and W', and the system is ready for the next iteration.

Note that, for simplicity, in Figure 3 we use a predefined value Max to determine the number of iterations. However, this is not always the best strategy, other stopping criteria such as a predefined utility value can also be used to determine the number of iterations. Pseudo-codes for these building blocks are included in Tables 2 to 5.

4 An Example: 1-D Robot Control

As an example, we consider a simple planar manipulator with one rotational joint (Figure 4). We assume, without loss of generality, that the robot links can be represented as point-masses concentrated at the end of the link. The link mass and length are respectively: $M = 0.1\ kg$,

Assume m inputs, n outputs, H hidden nodes, T samples, and $N = m + H$.

Inputs: $X_i(t)$ system state at time t $i \in \{1, \ldots, m\}$
W_{ij} internal weights $i, j \in \{1, \ldots, N + n\}$
W'_i internal weights $i, j \in \{m + 1, \ldots, N_M + n_M\}$

Outputs: $x_i(t)$ internal state $i \in \{1, \ldots, N + n\}$
$\hat{Y}_i(t) = u_i(t)$ control signals $i \in \{1, \ldots, n\}$

Action$(X(t), W, W', x(t), \hat{Y}(t))$
{

Step 1: for $(i = 1$ to $i = m)$
$x_i(t) = X_i(t)$

Step 2: for $(i = m$ to $i = N + n)$
$$\begin{cases} net_i(t) &= \sum_{j=1}^{i-1} W_{ij} x_j(t) + W'_i x_i(t-1) \\ x_i(t) &= s(net_i(t) \end{cases}$$

Step 3: for $(i = 1$ to $i = n)$
$\hat{Y}_i(t) = u_i(t) = x_{(m+H+i)}(t)$

}

Table 2: *A Pseudo-Code for Subroutine Action.*

Let $m, n, T, X(t), u(t)$ be defined in Table 2.
Assume $m_M = m + n$ inputs, $n_M = m$ outputs, H_M hidden nodes, and $N_M = H_M + n_M$.

Inputs: $X_{Mi}(t)$ contains $X(t)$ and $u(t)$ $i \in \{1, \ldots, m_M\}$
$W_{M\,(ij)}$ resultant weights after training $i, j \in \{1, \ldots, N_M + n_M\}$
$W'_{M\,i}$ resultant weights after training $i, j \in \{m + 1, \ldots, N_M + n_M\}$

Outputs: $x_{Mi}(t)$ internal state of Model network $i \in \{1, \ldots, N_M + n_M\}$
$\hat{Y}_{Mi}(t)$ outputs to Utility function $i \in \{1, \ldots, n_M\}$

Model$(X_M(t), W_M, W'_M, x_M(t), \hat{Y}_M(t))$
{

Step 1: for $(i = 1$ to $i = m_M)$
$x_{Mi}(t) = X_{Mi}(t)$

Step 2: for $(i = m_M$ to $i = N_M + n_M)$
$$\begin{cases} net_{Mi}(t) &= \sum_{j=1}^{i-1} W_{Mij} x_{Mj}(t) + W'_{Mi} x_{Mi}(t-1) \\ x_{Mi}(t) &= s(net_{Mi}(t) \end{cases}$$

Step 3: for $(i = 1$ to $i = n_M)$
$\hat{Y}_{Mi}(t) = x_{M(m+H+i)}(t)$

}

Table 3: *A Pseudo-Code for Subroutine Model.*

Let $m, n, T, X(t), u(t)$ be defined in Table 2, and $m_M = m + n$, $n_M = m$.
Assume H_M hidden nodes, and $N_M = H_M + n_M$.

Inputs:

$F\_X_i(t)$	from F_Utility	$i \in \{1, \ldots, n_M\}$
$x_{M\,i}(t)$	internal state of Model network	$i \in \{1, \ldots, N_M + n_M\}$
$W_{M\,(ij)}$	resultant weights after training	$i, j \in \{1, \ldots, N_M + n_M\}$
$W'_{M\,i}$	resultant weights after training	$i, j \in \{m+1, \ldots, N_M + n_M\}$

Outputs: $F\_u_i(t) = \frac{\partial U(t)}{\partial u_i(t)} \quad i \in \{1, \ldots, n\}$

F_Model($F\_X(t), W_M, W'_M, x_M(t), F\_u_i(t)$)
{

Step 1: for ($i = 1$ to $i = N_M$) $\quad F\_x_{M\,i}(t) = 0.0$

Step 2: for ($i = 1$ to $i = n_M$) $\quad F\_x_{M\,(N_M+i)}(t) = F\_X_i(t)$

Step 3: for ($i = N_M + n_M$ to $i = 1$)

$$\begin{cases} F\_x_{M\,i}(t) = F\_x_{M\,i}(t) + \sum_{j=i+1}^{N_M+n_M} W_{M\,(ji)} * F\_net_{M\,j}(t) \\ \qquad\qquad + W'_{M\,i} * F\_net_{M\,i}(t+1) \\ F\_net_{M\,i}(t) = s'(net_{M\,i}) * F\_x_{M\,i}(t) \end{cases}$$

Step 4: for ($i = 1$ to $i = n$) $\quad F\_u_i(t) = F\_x_{M\,m+i}$

}

Table 4: *A Pseudo-Code for Subroutine F_Model.*

Let $m, n, T, X(t), u(t)$ be defined in Table 2.
Assume H_M hidden nodes, and $N_M = H_M + n_M$.

Inputs:

$F\_u_i(t)$	from F_Utility	$i \in \{1, \ldots, n\}$
$x_i(t)$	internal state of Action network	$i \in \{1, \ldots, N + n\}$
$W_{(ij)}$	internal weights	$i, j \in \{1, \ldots, N + n\}$
W'_i	internal weights	$i, j \in \{m+1, \ldots, N + n\}$

Outputs: $F\_W_{ij}(t) = \frac{\partial U(t)}{\partial W_{ij}} \quad i, j \in \{1, \ldots, N + n\}$

$F\_W'_i(t) = \frac{\partial U(t)}{\partial W'_i} \quad i \in \{1, \ldots, N + n\}$

F_Action($F\_X(t), W_M, W'_M, x_M(t), F\_u_i(t)$)
{

Step 1: for ($i = 1$ to $i = N$) $\quad F\_x_i(t) = 0.0$

Step 2: for ($i = 1$ to $i = n$) $\quad F\_x_{(N+i)}(t) = F\_u_i(t)$

Step 3: for ($i = N + n$ to $i = 1$)

$$\begin{cases} F\_x_i(t) = F\_x_i(t) + \sum_{j=i+1}^{N+n} W_{(ji)} * F\_net_j(t) + W'_i * F\_net_i(t+1) \\ F\_net_i(t) = x_i(t) * (1 - x_i(t)) * F\_x_i(t) \\ F\_W_{ij} = F\_W_{ij} + F\_net_i(t) * x_j(t) \quad j = 1, \ldots, n \end{cases}$$

Step 4: for ($i = 1$ to $i = n$) $\quad F\_u_i(t) = F\_x_{M\,m+i}$

}

Table 5: *A Pseudo-Code for Subroutine F_Action.*

$L = 1$ m. This simple dynamic system is governed by the equation:

$$\tau(t) = M\ L^2\ddot{\theta}(t) + M\ g\ L\ \cos(\theta(t)) \tag{18}$$

where $g = 9.81\text{m/s}^2$ is the gravitational constant.

For illustration purpose, we further simplify Equation 18 by substituting

$$\begin{aligned} \dot{\theta}(t) &\approx \tfrac{\theta(t)-\theta(t-1)}{\delta t} \\ \ddot{\theta}(t) &\approx \tfrac{\dot{\theta}(t)-\dot{\theta}(t-1)}{\delta t} \end{aligned}$$

into the equation. The resultant equation is a non-linear function of $\theta(t)$ only, assuming $\theta(t-1)$ and $\dot{\theta}(t-1)$ are known at time t. More specifically,

$$\tau'(t) = \tau(t) + \frac{M\ L^2\ \theta(t-1)}{\delta t^2} + \frac{M\ L^2\ \dot{\theta}(t-1)}{\delta t} \approx \frac{M\ L^2\ \theta(t)}{\delta t^2} + M\ g\ L\ \cos\theta(t) \tag{19}$$

We emphasize that Equation 19 is a simplified representation of the actual system. It reduces the system into a function of a single variable, $\theta(t)$. Such simplification allows us to gain insights in the theory of Backpropagation of Utility. Our objective is to produce a forecast of control signals, $\tau'(t)$ and eventually $\tau(t)$ for the dynamic system described in Equation 19 to move from some initial angle, θ_0 to the final position, θ_f, in a specific amount of time with sampling period $\delta t = 0.2$ seconds. We assume the initial joint velocity and acceleration are $\dot{\theta}(t=0) = \ddot{\theta}(t=0) = 0$. The desired trajectory $\theta_d(t)$ at various sampling points is specified by the user.

Since our objective here is tracking control, we use the utility function

$$U(t) = \sum_{t=1}^{t_f} \frac{1}{2}\left(\theta(t) - \theta_d(t)\right)^2 \tag{20}$$

where $\theta(t)$ is the actual joint angle at time t and is also the system state $X(t)$. In other words, $m = n = 1$. As described in Section 3, application of the Backpropagation of Utility algorithm involves first, training of the Model network and then adaptation of the Action network. In the training of the Model network, $\tau'(t)$ is the input and $\theta(t)$ is the output. That is, $m_M = n_M = 1$.

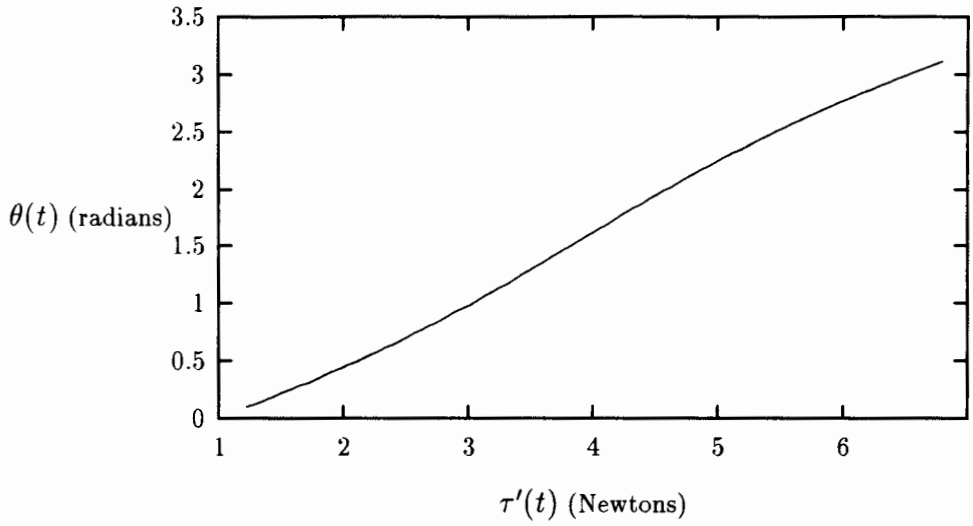

Figure 5: *The Inverse Dynamic Equation ($\theta(t)$ vs. $\tau'(t)$).*

For adaptation of the Action network, there are the feedforward and feedback components (Section 3.2 and Figure 3). In the feedforward mode, the input of the Action network is $X(t) = \theta(t)$, its output is $u(t) = \tau'(t)$ which is used for the Model network to determine the system state at time $t+1$, i.e., $X(t+1) = \theta(t+1)$. After a series of $u(t)$ is produced, in the feedback mode, the gradient of the Utility with respect to system state is:

$$\frac{\partial U(t)}{\partial X(t)} = \frac{\partial U(t)}{\partial \theta(t)} = \theta(t) - \theta_d(t).$$

This result is used by the Model network (F_Model routine) to determine

$$\frac{\partial U(t)}{\partial u(t)} = \frac{\partial U(t)}{\partial \theta(t)} \frac{\partial \theta(t)}{\partial \tau'(t)} = (\theta(t) - \theta_d(t)) F\_u(t)$$

which in turn is used to determine $\frac{\partial U(t)}{\partial W_{ij}}$, the gradient of the utility with respect to the weight space, W_{ij} in the Action network.

In the implementation of the Backpropagation of Utility algorithm, we found that the success of the algorithm hinges upon how accurate the Model network can emulate the inverse dynamic system given by Equation 19. To gain insights in the problem, Figure 5 shows the inverse dynamic system by plotting $\theta(t)$ as a function of $\tau'(t)$. Once trained, the subroutine

Model will provide $\theta(t)$ for any given $\tau'(t)$ in the feedforward mode. In the feedback mode, $F\_u(t) = \frac{\partial X(t)}{\partial u(t)} = \frac{\partial \theta(t)}{\partial \tau'(t)}$ is identified by the Model network through the subroutine **F_Model**.

In our initial attempt, we tried to train the Model network very accurately for a few points to emulate the function depicted in Figure 5. More specifically, we use a two-layer Model network with $H_{M\,1} = H_{M\,2} = 5$ neurons in each layer. The input and output layer each has $m_M = n_M = 1$ neuron. We use the basic backpropagation algorithm outlined in Table 1 (with $W' = 0$ for all nodes at all times) for training the Model network with $T = 5$ sampling points equally spaced over $\theta_0 = 0.1$ to $\theta_f = \pi$ radians. Since the sigmoid function (Equation 8) is used as transfer function, the desired outputs must be between 0 and 1. From our experimentation, we found that learning is more efficient, if the desired output is scaled between $ymin = 0.1$ and $ymax = 0.9$. That is, for training of the Model network, the input and desired output sampling points are:

$$X_M(t) = \frac{ML^2}{\delta t^2}\theta(t) + M\ g\ L\cos(\theta(t))$$

$$Y_M(t) = \frac{\theta(t)-\theta_0}{\theta_f-\theta_0}(ymax - ymin) + ymin$$

where $\theta(t) = \theta_0 + t * d\theta$, $t = 0, \ldots, T-1$ and $d\theta = (\theta_f - \theta_0)/(T-1)$. With this small number of training samples ($T = 5$), the network is able to train very accurately in the forward mode. However, when we use this trained Model network to calculate the feedback component $F\_u(t) = \frac{\partial X(t)}{\partial u(t)} = \frac{\partial \theta(t)}{\partial \tau'(t)}$ (using the **F_Model** routine), we found that the results are not accurate and consequently, the backpropagation of utility algorithm does not work. We attribute this failure to insufficient sampling points.

To obtain an adequate representation of the system, we need to train the Model network with sufficient number of sampling points and good accuracy. In this case, we use a bigger network with two hidden layers and 30 hidden nodes in each layer. Figures 6 and 7 show the results for training with $T = 100$ samples over the entire possible range of $\tau'(t)$. Both the forward and backward components have good accuracy. Figure 6 plots the error $|\theta_d(t) - \theta(t)|$ vs. $\tau'(t)$. This error ranges from approximately, 10^{-4} to 10^{-6}. The error for the backward component $|F\_u_{\text{desired}} - F\_u_{\text{actual}}|$ is the difference between the output of the **F_Model** routine and that of the desired value. Such error versus $\tau'(t)$ is plotted in Figure 7. Again, the accuracy is good. The error ranges, approximately, from $10^{-2.5}$ to 10^{-5}.

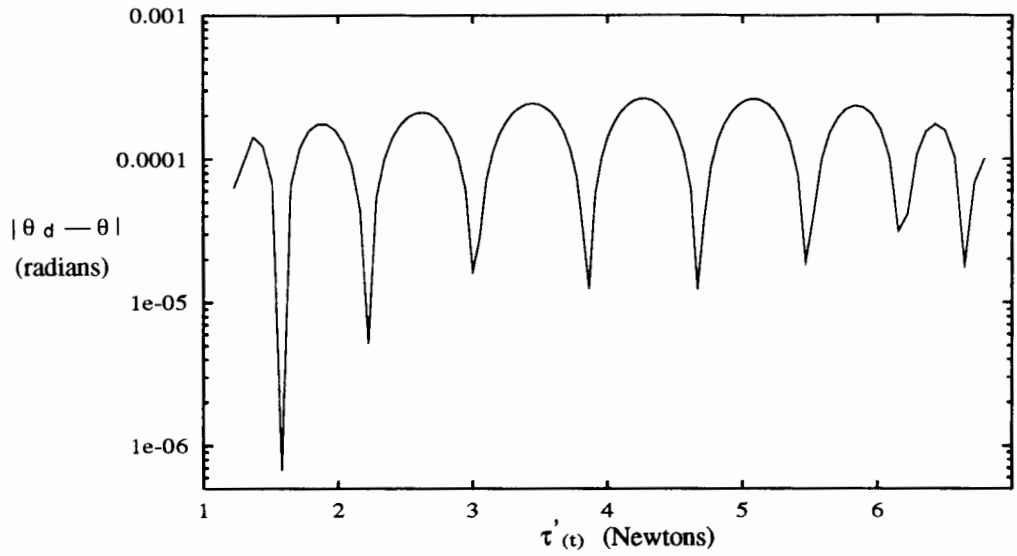

Figure 6: *Error in the Feedforward Component of Model Network (100 Training Points).*

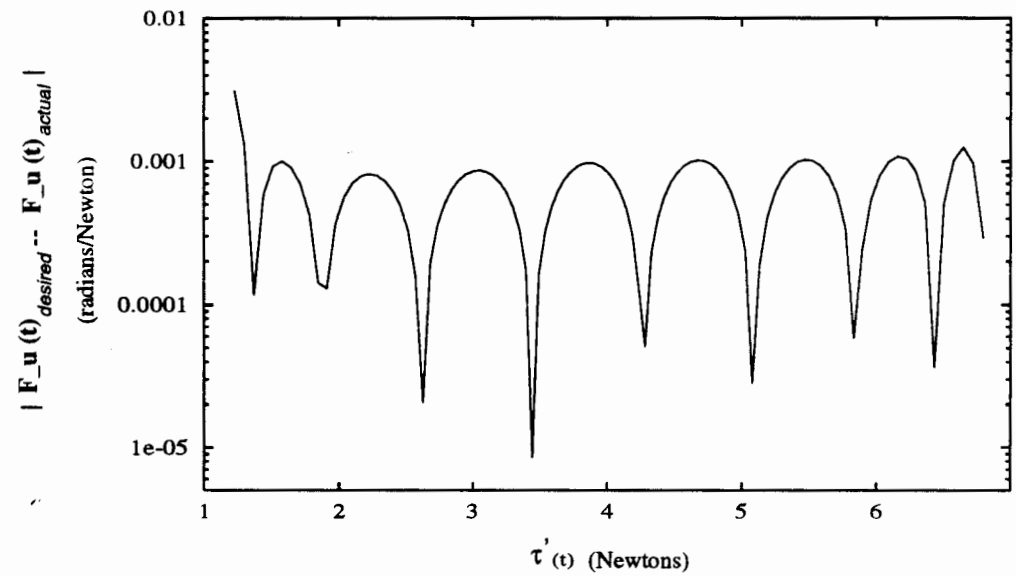

Figure 7: *Error in the Feedfback Component of Model Network (100 Training Points).*

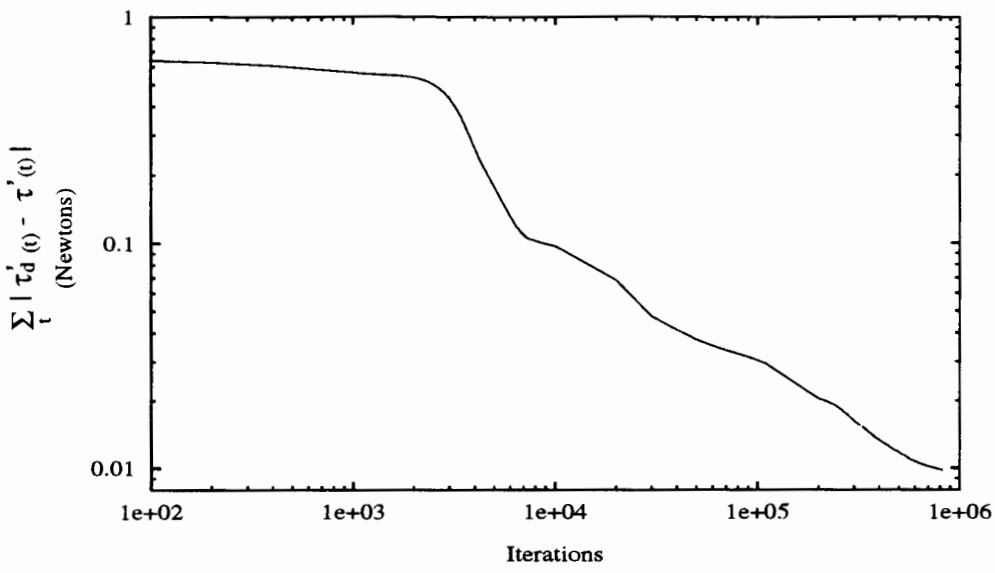

Figure 8: *Adaptation of Action Network.*

With the Model network successfully trained, we are ready for the adaptation of the Action network as outlined in Figure 3. As an example, we adapt the action network to generate a series of control signal $\tau'(t)$ to drive the 1-D robot from $\theta_0 = 0.1$ to $\theta_f = \pi$ in 4 seconds with a sampling period of 0.2 second. That is, a total of 20 control signals $\tau'(t)$ are required. We use a two-layer Action network with 10 nodes in each hidden layer and the backpropagation through time algorithm (with adjustment of the W' weight in the Action network).

Figure 8 shows the total error $\sum_{t=0.0}^{t=4} |\tau'(t) - \tau'_d(t)|$ ($\tau'(t)$ is the predicted valude from the Action network and the $\tau'_d(t)$ is the desired value) versus the number of iterations. Figure 9 plots the difference between the desired and the actual $\tau'(t)$ generated by the Action network over the entire period. Note that, unlike basic supervised control, the Backpropagation of Utility algorithm does not require the desired value $\tau'_d(t)$ be available to the Action network. They are used here only to illustrate the performance of the action network. These figures show clearly that the weights of the action network is adapting to generate a forecast of the desired control signals based soley on the feedback signals $F\_u(t)$ from the **F_Model** routine.

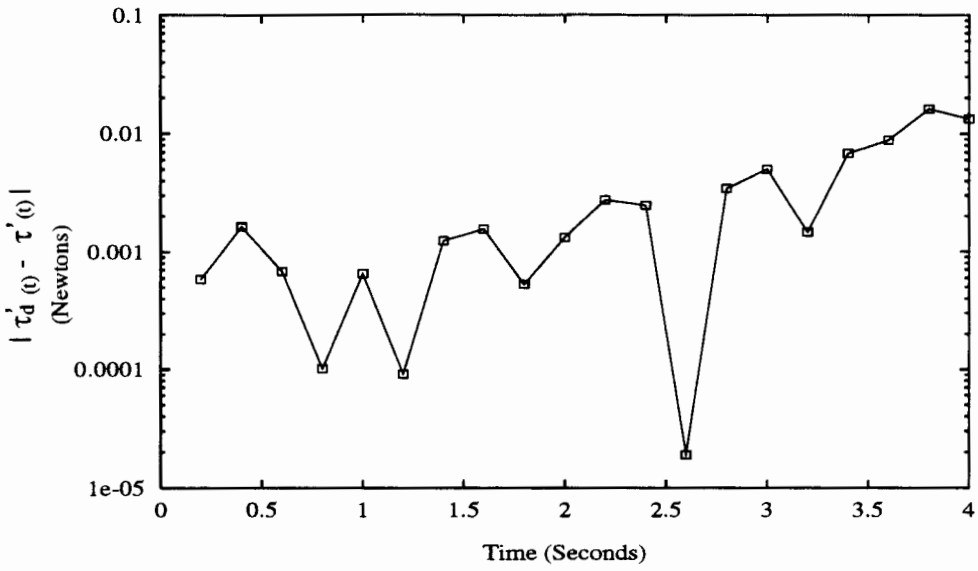

Figure 9: *Errors in Control Signals.*

5 Conclusions

Backpropagation of Utility is one of the five methods for neuro-control. Its goal is to provide a series of control signals to maximize a utility function. In this paper we demonstrated how to use the basic backpropagation and backpropagation through time algorithms as building blocks for the backpropagation of utility algorithm.

Basically, the algorithm is composed of three subnetworks, the *Action* network, *Model* network and the *Utility* network which sometimes can be represented as a simple *Utility* function. Each of these networks has the feedforward components **Action**, **Model** and **Utility** and the feedback components **F_Action**, **F_Model** and **F_Utility**, respectively. The interaction of these components can best be described in the flow chart of Figure 3. To further illustrate the algorithm, we use the algorithm to control a $1 - D$ planar robot. We showed that the success of the algorithm hinges upon a sufficient emulation of the dynamic system by the model network (both the **Model** and **F_Model** routines). Pseudo computer-codes for all routines and basic equations for each building block are also included.

References

[1] Richard L. Lippmann. "An Introduction To Computing With Neural Nets". *IEEE ASSP Magazine*, pages 4–22, April 1987.

[2] Judith E. Dayhoff. *Neural Network Architectures.* Van Nostrand Reinhold, New York, 1990.

[3] W. Horne, M. Jamshidi, and N. Vadiee. "Neural Networks in Robotics: A Survey". *IEEE Journal of Intelligent and Robotic Systems*, 3:51–66, 1990.

[4] Paul J. Werbos. "Backpropagation and Neurocontrol: A Review and Prospectus". In *International Joint Conference on Neural Networks*, pages 209–216, Washington, DC, June 18-22 1989.

[5] Paul J. Werbos. "Backpropagation: Past and Future". In *IEEE International Conference on Neural Networks*, pages 343–353, San Diego, CA, July 24-27 1988.

[6] D. Rumelhart and J. McClelland. *Parallel Distributed Processing*, volume 1. MIT Press, Cambridge, MA, 1986.

[7] B. Widrow and M.A. Lehr. "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation". *Proceedings of the IEEE*, 78(9):1415–1442, September 1990.

[8] Robert A. Jacobs. "Increased Rates of Convergence Through Learning Rate Adaptation". *Neural Networks*, 1:295–307, 1988.

[9] Paul J. Werbos. "Backpropagation Through Time: What It Does and How to Do It". *Proceedings of the IEEE*, 78(10):1550–1560, October 1990.

[10] K. W. Tang and H-J Chen. A Comparative Study of Basic Backpropagation and Backpropagation Through Time Algorithms. Technical Report TR-700, State University of NY at Stony Brook, College of Engineering and Applied Sciences, November 1994.

[11] D. Psaltis, A. Sideris, and A. Yamamura. "A Multilayered Neural Network Controller". *IEEE Control Systems Magazine*, pages 17–21, April 1988.

[12] W.T. Miller, R.P. Hewes, F.H. Glanz, and L.G. Kraft. "Real-time Control of an Industrial Manipulator using a Neural Network Based Learning Controller". *IEEE Journal of Robotics Research*, 6(2):84–98, 1987.

[13] W. T. Miller. "Real-time Application of Neural Networks for Sensor-Based Control of Robots with Vision". *IEEE Transactions on Systems, Man and Cybernetics*, 19(4):825–831, July/August 1989.

[14] J. Tanomaru and S. Omatu. "Towards Effective Neuromorphic Controllers". In *IECON*, pages 1395–1400, Kobe, November 1991.

[15] M.B. Leahy, M.A. Johnson, and S.K. Rogers. "Neual Network Payload Estimation for Adaptive Robot Control". *IEEE Transactions on Neural Networks*, 2(1):93–100, 1991.

[16] A. Guez, J.L. Eilbert, and M. Kam. "Neural Network Architecture for Control". *IEEE Control Systems Magazine*, pages 22–25, April 1988.

[17] A. Guez et al. "Neuromorphic Architectures for Adaptive Robot Control: A Preliminary Analysis". In *IEEE International Conference on Neural Networks*, pages 567–572, 1987.

[18] A. Guez et al. "Neuromorphic Architectures for Fast Adaptive Robot Control". In *IEEE International Conference on Robotics and Automation*, pages 145–149, 1988.

[19] L.G. Kraft and D.P. Campagna. "Comparison of CMAC Architecture for Neural Network Based Control". In *The 29th Conference on Decision and Control*, pages 3267–3269, Honolulu, Hawaii, December 1990.

[20] A.Y. Zomaya and T.M. Nabhan. "Centralized and Decentralized Neuro-Adaptive Robot Controllers". *Neural Networks*, 6:223–244, 1993.

[21] B. Bavarian. "Introduction to Neural Networks for Intelligent Control". *IEEE Control Systems Magazine*, pages 3–7, April 1988.

[22] K.S. Narendra and K. Parthasarathy. "Identification and Control of Dynamical Systems Using Neural Networks". *IEEE Transactions on Neural Networks*, 1(1):4–27, March 1990.

[23] Paul J. Werbos. "Neural Networks for Control: An Overview". In *1990 American Control Conference*, pages 983–984, San Diego, CA, May 23-25 1990.

[24] Paul J. Werbos. "An Overview of Neural Networks for Control". *IEEE Control Systems Magazine*, 11(1):40–41, January 1991.

[25] Paul J. Werbos. "Neural Networks for Robotics and Control". In *Wescon/89*, pages 684–688, San Francisco, CA, November 14-15 1989.

[26] Paul J. Werbos. "Neural Networks for Control and System Identification". In *IEEE Conference on Decision and Control*, pages 260–265, Tampa, FL, December 13-15 1989.

[27] Paul J. Werbos. Neurocontrol and Supervised Learning: an Overview and Evaluation. In D.A. White and D.A. Sofge, editors, *Handbook of Intelligent Control*, pages 65–89. Van Nostrand Reinhold, 1992.

[28] Paul J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, Cambridge, MA, November 1974.

[29] J. J. Craig. *Adaptive Control of Mechancial Manipulators.* Addison-Wesley Publishing Co., New York, NY, 1988.