Department of Computer Science

Controlling a Dynamic System in Real Time

Eric J. Furrow and Charles W. Anderson

Technical Report CS-94-119

August 10, 1994

Colorado State University

CONTROLLING A DYNAMIC SYSTEM IN REAL TIME

A fixed controller can only control a system within certain parameters. A learning controller can learn to control a system even if the parameters change. First, the fixed proportional derivative controller is explained and shown to be able to control the temperature on the top of a beaker of water in real time. Using a type of neural net called a Qnet which combines temporal difference and reinforcement learning it is possible to learn to control the temperature. With this neural network, better results were obtained than with the proportional derivative algorithm.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Often, controlling a dynamic system in real time can be a difficult task. One way of controlling such a system is to figure out the equations that describe the system, and then use these equations to control the system. It may be very difficult to figure out these equations - indeed, it can be difficult to determine all the variables that affect the equations. In addition, if the system is changed in even a minor way, the old equations are useless and new ones must be derived.

A neural net could potentially be used to control such a system. It does not need to "know" the important variables in the equation. The neural net can filter out useless variables, while a person would have to know how each variable affects the system in order to write an equation to control it. Furthermore, if the system is changed, the neural net just needs to be re-trained. Retraining is potentially much easier and faster than rederiving a new set of equations.

I explored using both a neural net and a physical control equation to keep the top of a beaker of water close to a desired temperature. The only way to heat or cool the water was by turning a hot plate on or off. The hotplate could not be turned part way on; only discrete on/off control was allowed. The only feedback is from several thermocouples on the outside of the beaker and in the water. There are currently four thermocouples. There is also a mixer which can be turned on or off (see Figure 1.1).

Only sensor #0 is actually in the water. Sensors #1 and #2 are attached to the outside of the beaker. Sensor #3 is on top of the hot plate. The water's desired temperature is kept at 80° C. If the top temperature is greater than 92° C, the program automatically turns off the hotplate and shuts itself down.

1.1 The Proportional Derivative Algorithm

The Proportional Derivative (PD) algorithm is one standard way of controlling the temperature at the top of the water. First, an equation relating the temperature change to the error is needed. Notice that when the water is cold, it should be heated quickly. When the water is near the desired temperature (called the setpoint), it should be heated more slowly. Therefore, the slope

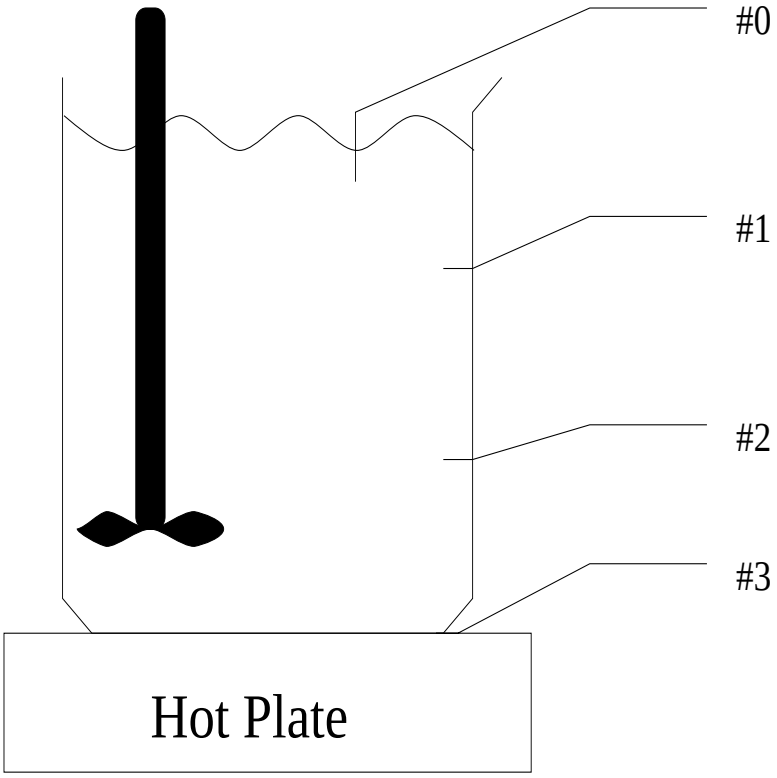

Figure 1.1: Beaker and Hotplate

of the temperature change should be directly proportional to the distance the actual temperature is from the setpoint, as shown in Figure 1.2. This equation can be written as:

$$(dT/dt) = C * error$$

In the equation above, dT/dt is the change in temperature with respect to time, $error$ equals the setpoint minus the actual temperature, and C is a constant. This equation can be rewritten as:

$$C * error - (dT/dt) = 0$$

The difference in temperature from one time step to the next is a very noisy measure of the slope, however. An exponential decay is used to average the slope over time.

$$DT = \gamma DT + (1 - \gamma)(dT/dt)$$

Let pd equal the error minus the exponential average of the current and past derivatives.

$$pd = C * error - DT$$

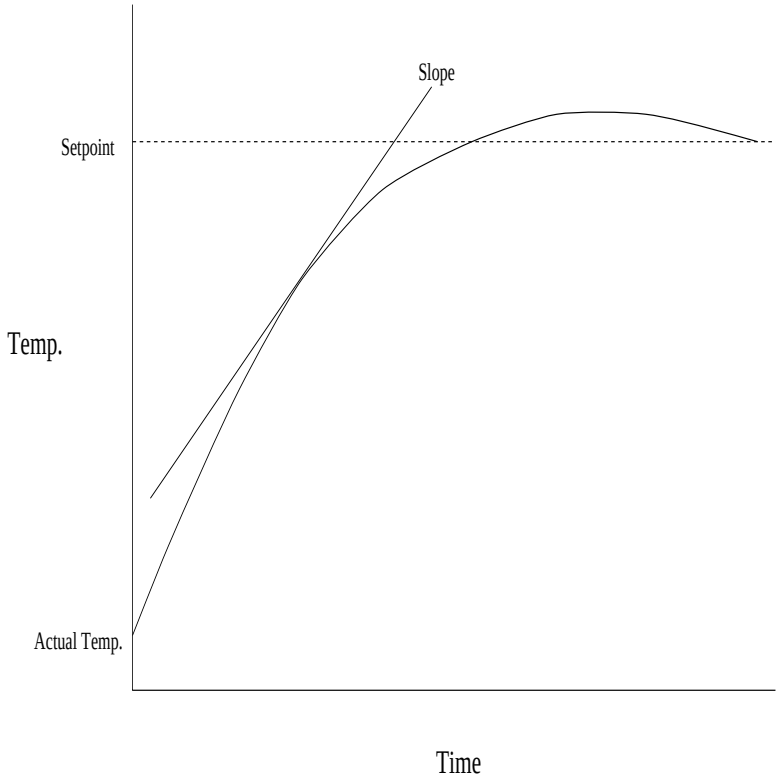

Figure 1.2: Heating Beaker

The sign of pd is used to decide when to turn on or off the hotplate. If $pd < 0$, the temperature is rising too fast and the hot plate is turned off. If $pd > 0$, the temperature is rising too slowly and the hot plate is turned on.

This procedure works quite well. It does have some problems, however. A given C works only in a small range of the variables affecting the problem, such as: desired temperature, amount of water, room temperature, wind speed, fan speed, etc. It is not really possible to add these variables into the equation, since it would quickly get incredibly complex. Another problem is that we do not know how to measure many of these variables.

1.2 Overview

First, several ways of representing neural nets are examined in Chapter 2. These include different ways of inputting data, types of hidden units, and training paradigms. In Chapter 3, the nets which use the above representations are discussed. Chapter 4 deals with the results of the Qnet, and Chapter 5 discusses extensions to the work already done.

Chapter 2

REPRESENTATIONS OF NEURAL NETS

The neural nets I used in this project had many different forms. There were many different ways to have the data input to the neural net, have the neural network represent the data, and to train the neural net. Following are some descriptions of the various issues in designing a neural network to control the beaker.

2.1 Input Representation

Two roughly equivalent ways to represent the input data exist. First, the error and the slope of the error could be input. This gives the distance from the setpoint and how fast that distance is changing. The other way is to input all of the temperatures from the thermocouples and the setpoint. The temperatures at different levels give an indication of how fast the temperature is rising or falling. For example, if the temperatures at the lower levels are high and the temperature at the top level is low, the temperature is rising. The difference between the temperatures from the thermocouples would correspond to the slope, and the difference between the setpoint and the top temperature would be the error.

There are essentially two ways to input the temperatures to the neural net. The obvious representation is using continuous temperatures. This has the advantage of using all the possible information about the problem. The other representation is as discrete temperatures. For example, all temperatures from 20° C to 50° C could be treated as a single "temperature." Obviously, a lot of information is lost in this representation, but training the neural net would take fewer iterations because a single point in an area would determine the neural net's response over that area. Of course, the ranges would have to be chosen intelligently so that each area has only one action associated with it. For example, the action for all temperatures below 70° C is the same, namely turning on the hotplate.

2.2 Internal Representation

There are various ways to represent the data inside the neural net. The hidden nodes can be sigmoid units, boxes, or Gaussian units. Each is explained below.

2.2.1 Sigmoid Units

Sigmoid units are the standard hidden units for a neural net. One disadvantage to them is that they are slow to train. Another is that it is not clear what a single node is doing to minimize the error for any particular temperature. Advantages is that they are easy to train and are not too sensitive to training parameters.

2.2.2 Boxes

The boxes representation uses a "box" for each hidden node. The box covers a range of the input space. In my implementation, the "boxes" do not move or overlap. The advantage is that it is easy to tell what each box is doing. The disadvantage is that the boxes do not generalize very well – the hotplate will either be on or off over the entire box. If the boxes are small, the accuracy will be good (that is, a single box will not cover regions where the hotplate should be on and a region where the hotplate should be off) but not all the spaces will be visited. If the boxes are large, most of the boxes will be visited but the accuracy will suffer.

2.2.3 Gaussian Units

Instead of boxes, the Gaussian units represent circles. The output of the Gaussian node gets higher as the input gets closer to the center of the node. Since the "bumps" overlap, several different nodes can be activated for any input. Since several nodes can contribute to one area in the state space, the net is able to average results smoothly across the state space, unlike the boxes representation. It is also fairly easy to see what each node is doing.

2.3 Training Paradigms

2.3.1 Reinforcement Learning

Reinforcement learning is a way of training a neural net that is based on getting a reinforcement, or evaluation of how well the net is doing. This is different from supervised learning, where the correct output is available. Obviously, the correct action for the beaker is difficult to determine, so supervised learning is hard to implement.

2.3.2 Temporal Difference

The problem with using just reinforcement learning for the beaker is that it can take several time steps for a given action to have any effect on the reinforcement. Temporal difference is a way of predicting the future results. The temporal difference algorithm uses the difference between the value of $state_{t+1}$ and $state_t$ as the error to update the neural net. Thus,

$$e_t = r_{t+1} + \gamma output_{t+1} - output_t$$

In other words, the error of the net at time t equals the reinforcement at the next time step plus the output of the net at time $t+1$ minus the output of the net at time t. γ is the discount value ($0 < \gamma < 1$).

2.3.3 Qnet

The Qnet predicts a number to be optimized (the reinforcer) for a control system. The output of the Qnet when trained should be:

$$Q(x_t) = output_t \approx \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

In the above equation r_t is the reinforcement at time t and x_t is the state at time t. The way the Qnet is trained depends on whether the action (hotplate on/off) is an input or the action is an output.

Action as input When the action is an input, one of the inputs to the Qnet is the hotplate action (on/off), and there is one output giving the predicted reinforcement. The error of the Qnet is:

$$e_t = r_{t+1} + \gamma \max_{a_{t+1} \in \{ON, OFF\}} \{Q(x_{t+1}, a_{t+1})\} - Q(x_t, a_t)$$

where $Q(x_t, a_t)$ is the output of the Qnet for state x and action a at time t. For each time step, the Qnet is run twice: once with the input for the hotplate action being on and one with the action being off. The one with the higher output is taken.

Action as output When the action is an output, only the state is input to the net. There are two outputs, one for the predicted reinforcement if the hotplate is on at time t ($Q_{ON}(x_t)$) and one for the predicted reinforcement if the hotplate is off at time t ($Q_{OFF}(x_t)$). The action chosen at a given time step is determined by the outputs of $Q_{ON}(x_t)$ and $Q_{OFF}(x_t)$; the higher output corresponds to the action chosen. The error for the Qnet is:

$$e_t = r_{t+1} + \gamma \max\{Q_{ON}(x_{t+1}), Q_{OFF}(x_{t+1})\} - Q_{a_t}(x_t)$$

where $Q_{ON}(x_t)$ and $Q_{OFF}(x_t)$ are the outputs of the net for the hotplate being on and off, respectively. $Q_{a_t}(x_t)$ is the output of the Qnet for the action actually taken at time t. Only the weights to the action chosen are updated; the others remain unchanged.

Chapter 3

METHODOLOGY

I tried several different methods to try to control the beaker. These methods use combinations of the representations presented in the last chapter. Results of these methods are discussed in Chapter 4.

3.1 Modelling PD

This attempt at controlling the temperature was an attempt to simplify the problem and to see whether a neural net could actually keep the temperature at the top of the beaker constant. In this procedure, the temperatures were split up into discrete ranges.

The neural net had only input nodes and a single output node. Only one input node was active at a time. All of the sensors were used as input. The 4 temperatures were split into 6 discrete ranges. This gave a total of 6^4, or 1296 possible combinations. An example of a combination might be: sensor#0 at 80° C, sensor #1 at 70° C, sensor #2 at 71° C, and sensor #4 at 536° C. Each of the 1296 input nodes corresponded to one of the possible combinations.

The neural net was trained by running the PD algorithm and adding the output of that algorithm to the weight of the one node that was active at each time step. So if pd was positive (the hotplate turned on), the active weight was increased. If pd was negative, the active weight was decreased. After the net was trained, a positive output resulted in the hotplate being turned on; a negative output resulted in the hotplate being turned off. Some nodes were never activated, but enough were trained so that the net could run independently of the PD algorithm.

When the net was run on its own, it did surprisingly well. Although it did not work as well as the PD algorithm, it did keep the actual temperature reasonably close to the desired temperature. Since this was just a simple test, I did not save any actual measurements of the control runs. This did prove that the temperatures represented the needed information to control the beaker. Therefore, the next method I tried used the temperatures and setpoint as the input.

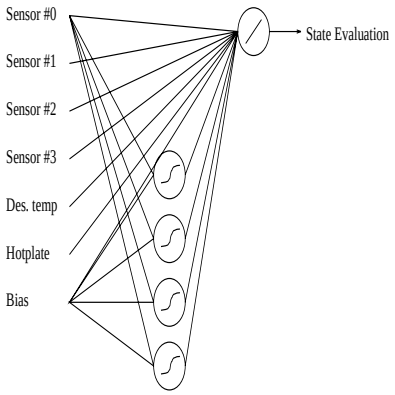

Figure 3.1: Qnet, action as input

3.2 Qnet, action as input

The next method I used to control the beaker was based on *Strategy Learning with Multilayer Connectionist Representations* [And88]. There is a difference between what Anderson did and what I am doing, though. Anderson's work is with the cart-pole problem. In the cart-pole problem there was a discrete failure state (where the cart failed to balance the pole). In my beaker problem "failure" is less well defined; the goal is to minimize the error over time, where error is defined as the difference between the setpoint and the actual temperature.

The inputs to the Qnet are from the four sensors on the beaker, the desired temperature, pd, and the hot plate state (on or off). An explanation of the Qnet is given in chapter 2. The network is a fully connected one hidden layer feed-forward net (see Figure 3.1). Note that all of the connections are not drawn.

The network is supposed to work by running the net on the current state with the input for the hotplate being on and then with the input for the hotplate being off. The outputs of the two runs are compared, and the action taken corresponds to the run which has a higher output.

Unfortunately, in practice this didn't work. The output of the net was always higher over all states for either a) the hotplate on or b) the hotplate off. The net may have worked better had I only trained it when the actual value was near the setpoint. Thus, for the next try, I decided to change the net substantially, using local information instead of global information.

3.3 Qnet, action as output

This method is based on *Q-Learning with Hidden-Unit Restarting* [And93]. The previous Qnet had all temperatures and the setpoint as input; this net uses just $error$ and $error'$ as input, where $error'$ is equivalent to DT in the PD algorithm. Also, the previous net used sigmoidal units while this net uses Gaussian units.

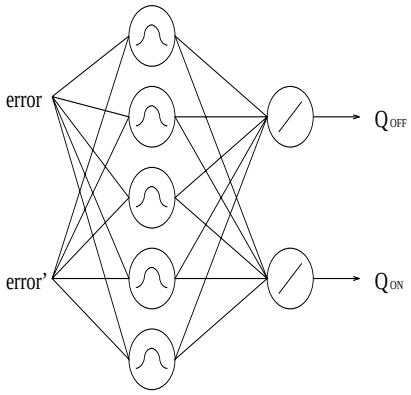

Figure 3.2: Qnet, action as output

This network has two inputs, 121 hidden nodes, and two outputs. $Error$ and $error'$ are scaled to have a range of one from the lowest to the highest value. The object of the Qnet is to predict an exponential discount of the future reinforcement such that:

$$Q_{a_t}(x_t) \approx \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

The hidden nodes have a Gaussian function and are fixed in place so they cover the state space (see Figure 3.2). Note that $error$ and $error'$ can be at any point in the space, the place they are at determines how many nodes are activated and how much each node is activated. The Gaussian nodes overlap significantly. Figure 3.3 is an actual representation of how much the nodes overlap. The Gaussian nodes' output, y_j is equal to:

$$dist_j = (error - w_{j,error})^2 + (error' - w_{j,error'})^2$$

$$y_j = \begin{cases} (1 - dist_j/\rho)^2, & \text{if } dist_j < \rho, \\ 0, & \text{otherwise} \end{cases}$$

ρ^2 is the radius of a Gaussian node. $w_{j,error}$ is the weight of the input to the hidden node. The output of $Q_{a_t}(x_t)$, where a_t is either ON or OFF, is:

$$Q_{a_t}(x_t) = \sum_{j=1}^{121} v_{j,a_t} y_j$$

In the above equation, v_{j,a_t} is the weight from hidden node j to $Q_{a_t}(x_t)$. The action that has the higher Q value is taken, since $Q_{a_t}(x_t)$ predicts the future reinforcement given that action and we want to maximize the reinforcement value. Only the weight to the output which was used is adjusted, as per the following formula:

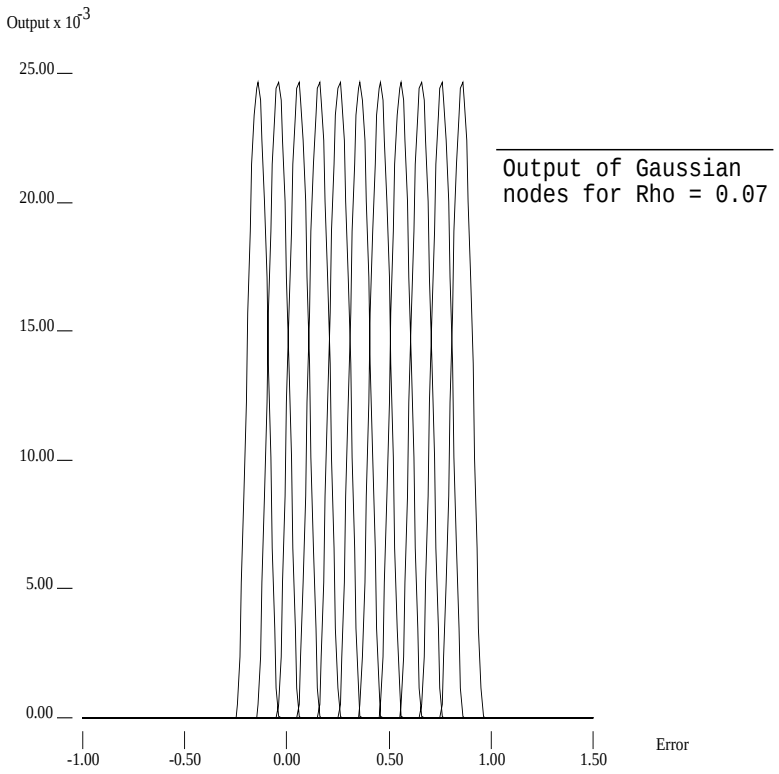

Figure 3.3: Outputs of Gaussian nodes

$$v_{j,a_t} = v_{j,a_t} + \beta e_t y_j$$

$$e_t = r_{t+1} + \gamma max(Q_{ON}(x_{t+1}), Q_{OFF}(x_{t+1})) - Q_{a_t}(x_t)$$

.

3.3.1 Other Qnet examples

I tried using different types of hidden units for the Qnet. One used boxes instead of Gaussian units (see Figure 3.4) to find how much effect the overlapping areas for the Gaussian nodes had. Another used sigmoid units (see Figure 3.5) to use global information instead of local information, which the Gaussian nodes used. I also tried using no hidden nodes (see Figure 3.6) to see how neccessary hidden nodes were.

3.4 On-line vs. Off-line training

There are two ways to train the neural nets: on-line and off-line. On-line means that the neural net is being trained in real time while the beaker is running. Off-line means that the net is being trained on several pre-run data files which contain measurements from previous runs. The

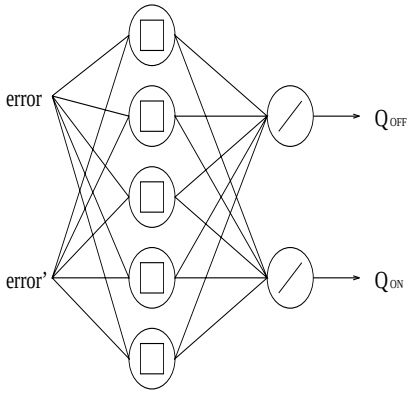

Figure 3.4: Boxes Representation

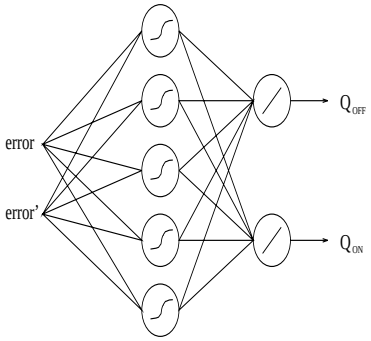

Figure 3.5: Sigmoid Net

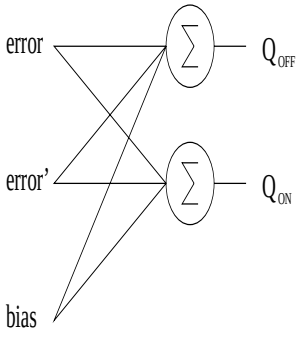

Figure 3.6: Linear Net

previous runs are an agglomeration of "noisy" data, runs of neural nets that did not keep the temperature near the setpoint, and runs of neural nets that did work well. The noisy data was recorded using the PD algorithm with a 0% to 50% chance of an opposite action from what the PD algorithm would do.

The advantage of working off-line is that it's faster. The disadvantage is that it relies on already generated files, which means that some states may never be explored with both the hotplate on and the hotplate off. The off-line runs were generally run for 200 through 1000 iterations through the training data.

The advantage of training the net on-line is that in theory it ought to examine most of the states, but it takes a long time. Given that it takes the off-line program several hundred iterations to train properly and that a run takes a minimum of 20 minutes, this could be a long time indeed.

Chapter 4

RESULTS OF QNET

Most of the results from the various methods were disappointing. However, the results using the Qnet with the action as output were quite promising.

4.1 Qnet, action as output

4.1.1 Gaussian hidden units

The reinforcement I first used was $|pd_{t-1}| - |pd_t|$. The reasoning behind this choice is that pd is a measure of the "goodness" of the current state. The closer pd is to 0, the better. Therefore, the reinforcement should reward the net (give a higher reinforcement) as pd gets closer to 0. The absolute value is there since pd can be either positive or negative. So $|pd_{t-1}| - |pd_t|$ rewards the net if pd is positive and getting less positive, or negative and getting less negative. I was eventually able to get the net to train properly. However, the reinforcement function was very noisy (see Figure 4.1). The function is noisy because the slope is noisy. Also, the thermocouples themselves do not give precise measurements of the temperature. Thus, the difference between the two will usually be around zero. Therefore, I decided to change the reinforcement function to $-abs(error_t)$. Figures 4.1 and 4.2 show graphs of the scaled error and the discounted sum of the future reinforcement. The reinforcement $-abs(error_t)$ was chosen since the error gets smaller as the state improves. With this reinforcement, the function is a lot smoother. All the results in the paper are using a reinforcement equal to $-abs(error_t)$. Figure 4.3 shows a graph of the discounted sum of the future reinforcement and the reinforcement predicted by the Qnet.

Our goal is to minimize the root mean squared error over the entire run; however, since the starting temperature ranges from 19° C to 25° C and since the length of the runs vary somewhat, it does not make sense to blindly measure the root mean squared error over the run. Therefore, I broke each run into four areas.

Figure 4.4 shows a run of the neural net with $\rho = 0.1$, where ρ is a measure of the width of a Guassian node. In Figure 4.4 Area A is the space from the starting temperature to 40° C. In this area, all that was measured was the number of time steps the hotplate was off. Since the

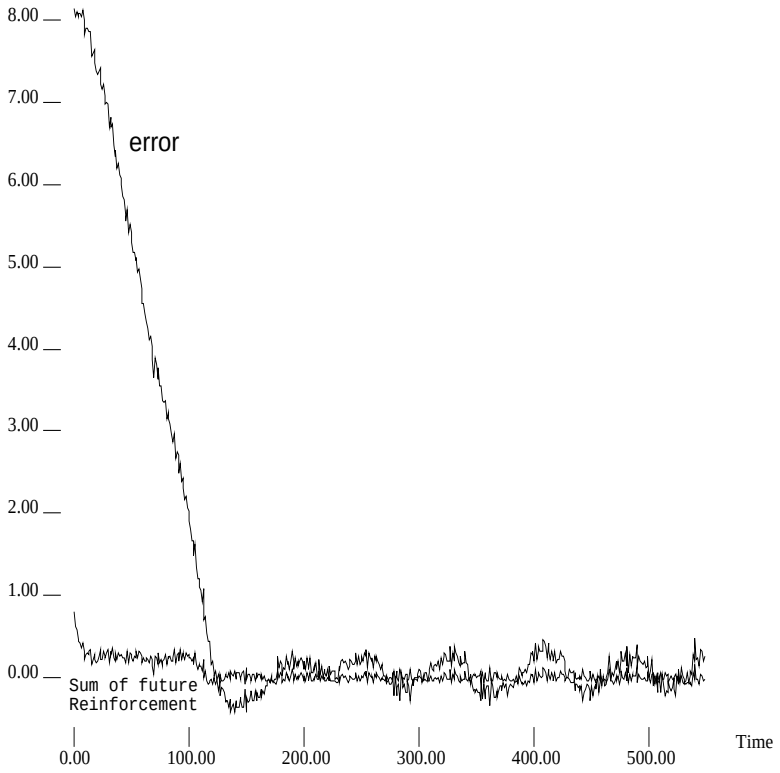

Figure 4.1: Discounted Sum of Future Reinfocement; Reinforcement = $|pd_{t-1}| - |pd_t|$

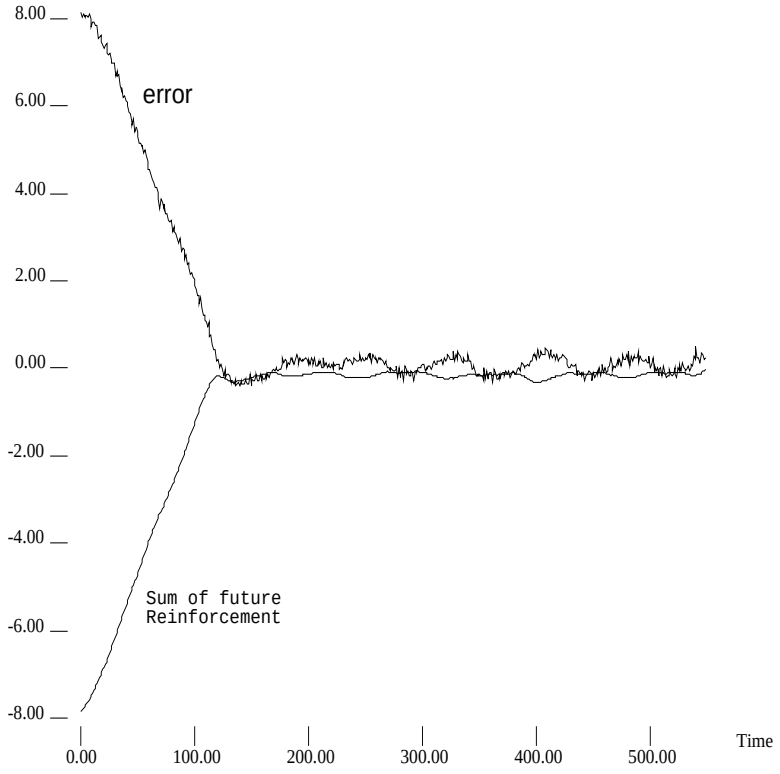

Figure 4.2: Discounted Sum of Future Reinforcement; Reinforcement = $-abs(error)$

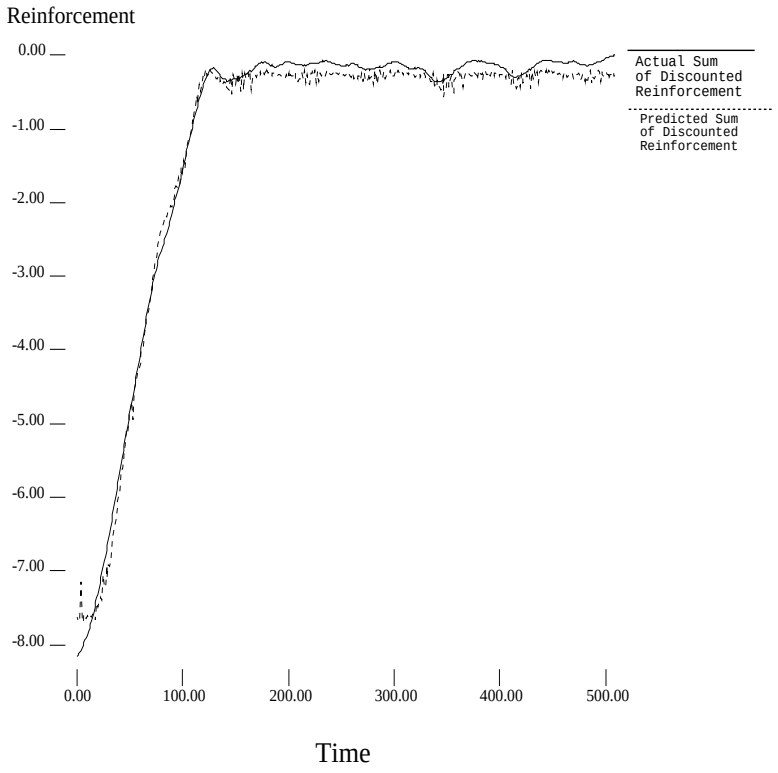

Figure 4.3: Discounted Sums of the Actual and Qnet Reinforcements

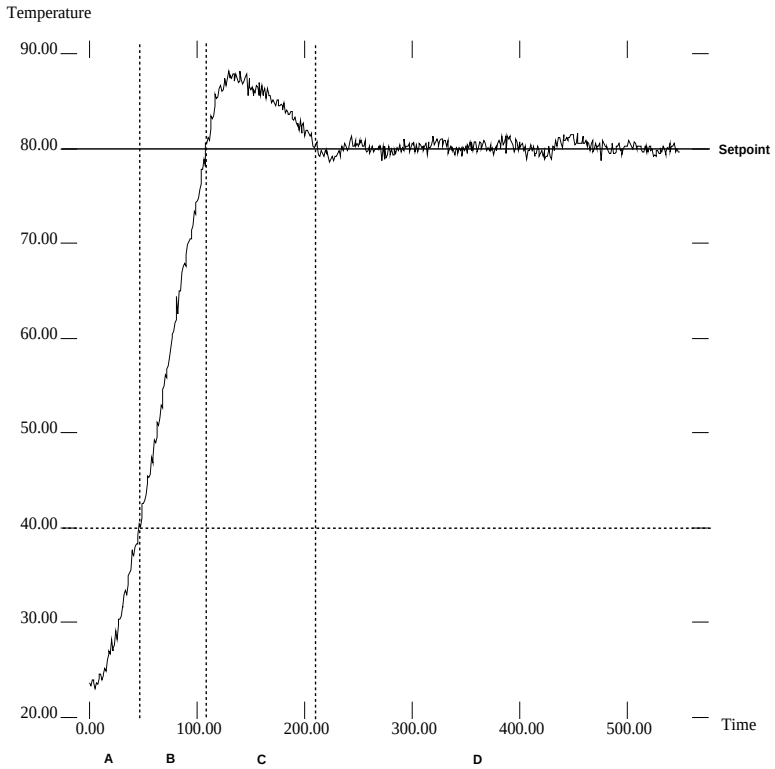

Figure 4.4: Typical run

	Area A	Area B	Area C	Area D
PD	0	22.43	2.27	0.85
PD	0	22.36	2.68	1.47
PD	0	22.83	4.23	1.21
PD	0	22.48	3.83	1.10
PD	0	22.70	3.02	0.96
Rho=0.07	10	21.52	1.98	1.29
Rho=0.07	1	21.46	1.43	1.70
Rho=0.07	4	20.65	2.22	1.70
Rho=0.07	0	21.10	2.12	1.72
Rho=0.07	5	22.07	3.43	1.68
Rho=0.1	7	23.41	4.94	0.83
Rho=0.1	10	22.78	5.97	0.86
Rho=0.1	6	22.70	5.43	0.81
Rho=0.1	10	22.85	5.28	0.60
Rho=0.1	3	23.00	5.31	0.80

Table 4.1: List of Results

	Area A	Area B	Area C	Area D
PD	0	22.56	3.21	1.12
Rho=0.07	4	21.36	2.24	1.62
Rho=0.1	7.2	22.75	5.39	0.77

Table 4.2: Comparisons of algorithms

hotplate should always be on in this area, this gives some indication of how well the neural net is doing. After 40° C, the RMS error is measured until the temperature hits 80° C. 40° C was somewhat arbitrarily chosen as a temperature at which the amount of heat in the beaker would be approximately the same regardless of the starting temperature. Area C is the area in which the algorithm (PD or the neural net) overshoots the setpoint. Since the overshoot can be fairly large, the RMS error could be altered significantly if it was measured in with Area D. Therefore, I measured Area C separately. Area C gives an indication of how quickly the algorithm settles down around the setpoint. The RMS error in Area D measures the long term quality of the algorithm.

I trained the neural net off-line with $\rho = 0.7$ and $\rho = 0.1$ on the same data. The net was a Qnet with 121 hidden nodes. The data consisted of several runs which ranged from just having a 50% chance of turning the hot plate on or off to just running the PD algorithm with no noise.

Table 4.1 shows how well each run did for each interval. Table 4.2 shows a scatter plot of each of the four areas for $\rho = 0.7$, $\rho = 1.0$, and the PD algorithm. Figures 4.5 and 4.6 are scatter plots of the values in Table 4.2 over areas A, B, C, and D. Each of the nets was trained off line and then run five times on the beaker. The results in Figure 4.6 show that the neural net with $\rho = 0.1$ can control the beaker better in the long run than the PD algorithm can.

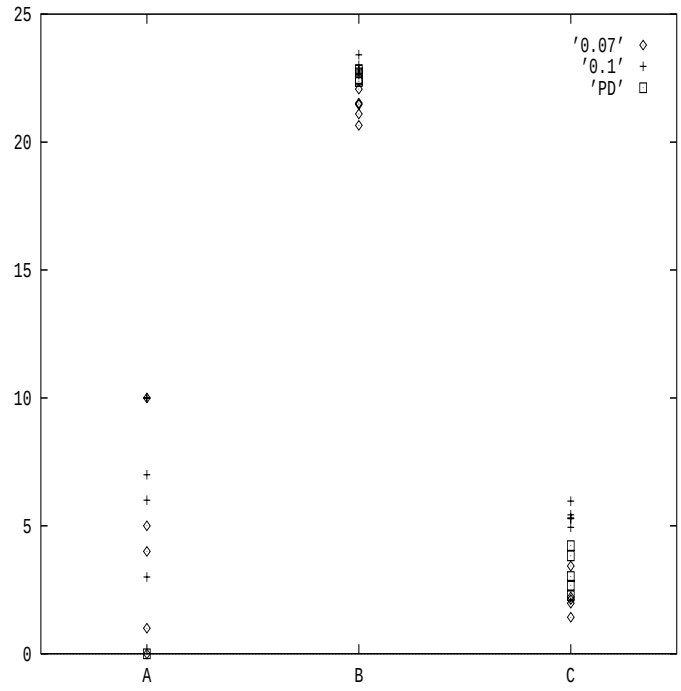

Figure 4.5: Scatter Plot for Areas A, B, and C

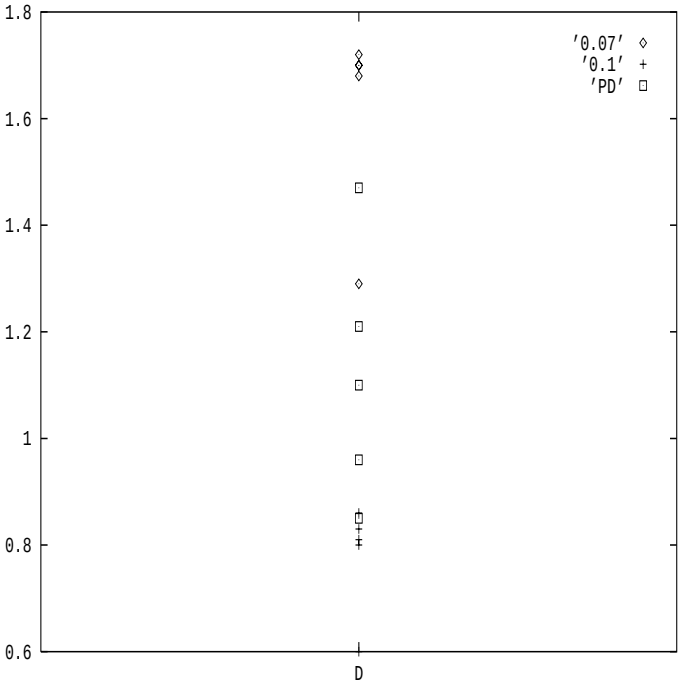

Figure 4.6: Scatter Plot for Area D

Unfortunately, I was never able to get the neural net to train on-line properly. When run from a "blank" net (i.e., all the output weights set to 0), it slowly adjusts the weights so that Q_{ON} and Q_{OFF} approximate the correct reinforcement. I did not really have time to run this more than 10 or so runs, after which it still was no good at controlling the beaker. It would either just keep heating until the cut-off point (92° C) or would stay stuck at some temperature for a long time, and then heat up to the cut-off point.

So I then took a net trained off-line (the $\rho = 0.7$ net in Table 4.2) and tried training some more on-line. This still did not work. On the first run, the error stayed about the same. On the second, the top temperature stayed about 5° C below the setpoint. On the third run, the top temperature was about 3° C below the setpoint. On the fourth run, the top temperature hit 92° C and the hotplate was turned off.

4.1.2 Sigmoid Hidden Units and No Hidden Units

The Qnet with Gaussian nodes worked well, but I also explored some other nets. I tried using sigmoid units to see whether using a global representation would work as well as the local representation Gaussian units use. A net with sigmoid units was unable to control the beaker. The number of hidden units with the sigmoid net did not seem to affect the results as long as there were more than four units. A sample of the predicted reinforcement and the actual reinforcement for the sigmoid net is shown in Figure 4.7. Although it is hard to see in the graph, Q_{OFF} is always higher than Q_{ON}. These results are for a net with four hidden nodes trained off-line on 29 runs for 2000 iterations. The net with no hidden nodes performed similarly, though Q_{OFF} and Q_{ON} did not follow the reinforcement as well. This suggests that local information is important in controlling the beaker.

4.1.3 Boxes

I also wanted to know how important the overlap feature of the Gaussian units was. The easiest way to find out was to test a similar net with box units instead of Gaussian units. Figure 4.8 shows the output of a net trained off-line on 29 runs for 2000 iterations with the boxes representation. The output is for one of the training examples. The error shown is the setpoint minus the actual temperature. The hotplate action is on (1) when Q_{ON} is greater than Q_{OFF} and off (-1) otherwise. In a few areas where the temperature is rising, the net has the incorrect output; however, it does seem to work better when the temperature is around the setpoint. Figure 4.9 shows that the output (Q_{ON} and Q_{OFF}) does not follow the reinforcement as well as the Gaussian nodes did (see Figure 4.3). The output of the net looks like it might run well on the beaker, but I have not tried that.

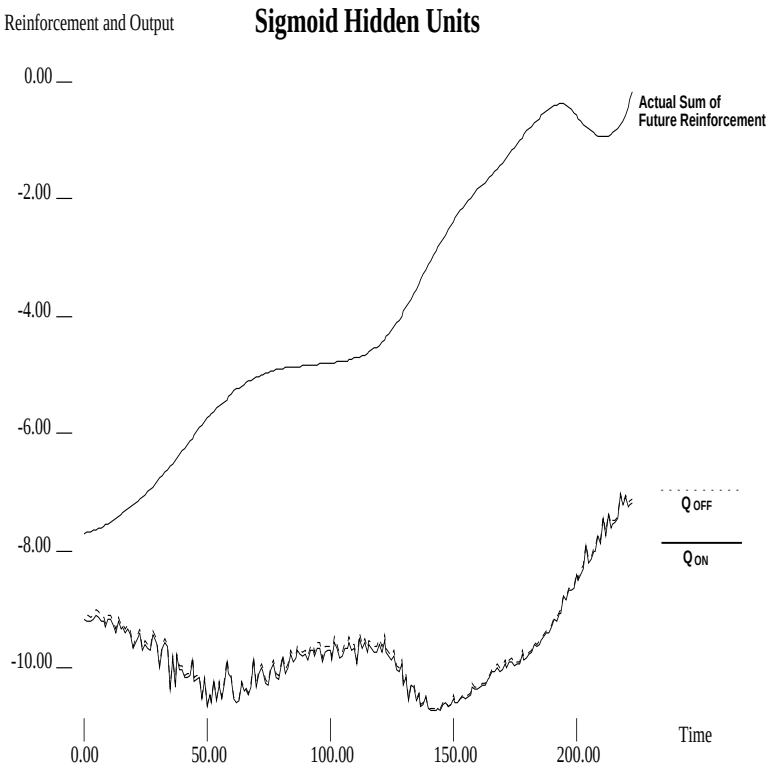

Figure 4.7: Actual and Predicted Reinforcement with Sigmoid Units

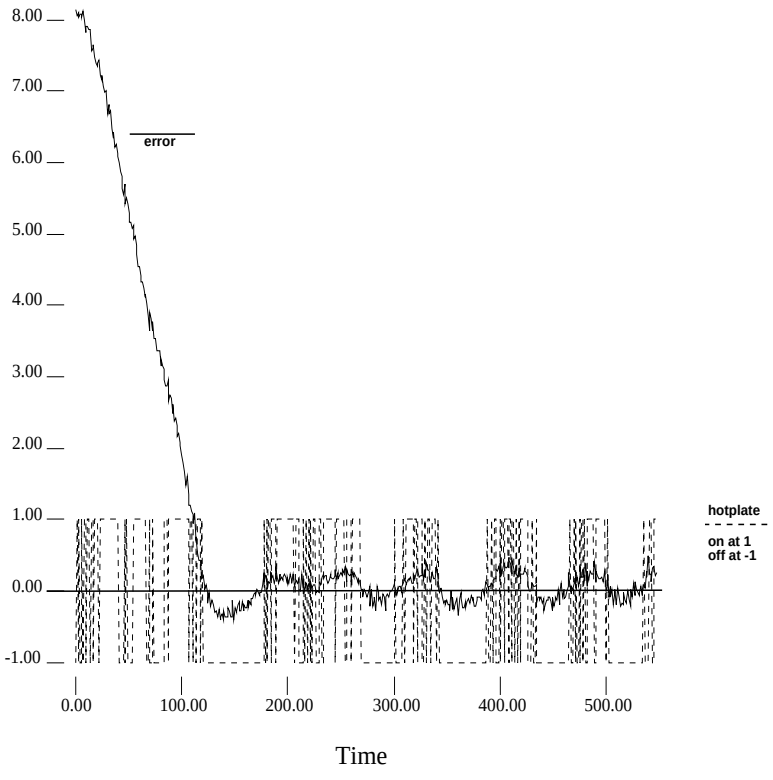

Figure 4.8: Output of Boxes Representation

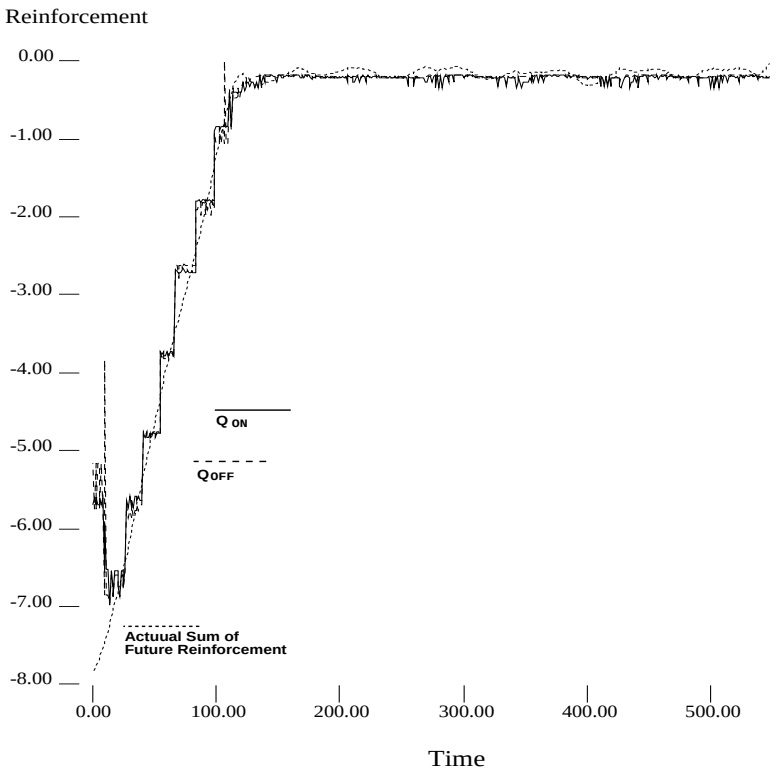

Figure 4.9: Actual and Predicted Reinforcement with Boxes Units

Chapter 5

PROBLEMS AND EXTENSIONS

5.1 Limitations

An advantage of the Qnet is that it should be able to adjust to changing conditions. Therefore, one big disappointment is that the Qnet did not work well when being trained on-line. Unfortunately, I did not have time to explore this phenomenon fully. A reason the net did not train well on-line may be that it did not learn the proper future reinforcement while being trained off-line. That is, the output was close, but not quite correct. Then when further training occurred, the output of the net was switched to the incorrect output. If this is the case, then either extensive training on-line or having more training sets off-line should eliminate the problem.

Another limitation of the work is that only the top temperature sensor was included. The control might be better if more of the sensors were used.

One inherent problem with the beaker is that it is extremely slow. Heating the beaker from 20° C to 80° C takes at least 15 minutes. The runs I used to test how well the various algorithms worked took one and a half hours. It would have been nice to work with a quicker system. Another inherent problem is that the water boils away. The maximum time the water can stay at 80° C with the thermocouple in the water is at most three hours (if that).

Although the Qnet did perform better than the PD algorithm, this does not necessarily show much. There are undoubtably better ways of controlling the beaker than by the proportional derivative algorithm. However, the main point is that the neural net was able to improve upon the training runs it was given.

One advantage of the neural net is that it is quicker to train the net than it is to tweak the PD algorithm until it works well. It is especially time consuming to get the PD algorithm working better when it is already working well – the constants must be changed a little, then the program has to be run for an hour or so to see how well it does, after which the whole thing has to be repeated. So at least the neural net is faster to get working, and I believe that neural net is better at controlling the beaker than the PD algorithm can be.

5.2 Future Work

The first thing to do would be to find why on-line training is not working. A way to find out is to try running it on-line after it has been trained off-line until it can control the beaker. Then examine the output of the net over $error$ and $error'$ to see how the net is changing.

Another thing to try would be to gradually decrease the noise over several runs. Most of the runs I made were either with 50% noise (i.e. random actions) or no noise. The net could be trained on-line with the noise slowly dropping over several runs.

One thing I did not do was to change the setpoint during the run. This really should be done to make sure the net is generalizing properly.

Once the on-line training is working, it would be nice to see how well the neural net can adjust to changing conditions. For example, if it was trained to work when the beaker was at 2000mL, could it keep the top of the water at a constant temperature as it boils away to 200mL? It would be interesting to see how fast the net can adjust to a changing situation.

5.3 Products of work

There are several products of this work. The hardware/software interface for measuring temperatures has been developed. This is general to almost any temperature control problem. Also, the software for neural nets to control a real time control system has been developed. The neural net is general; it can be applied to many control problems with very little adjustment.

REFERENCES

[And88] Charles W. Anderson. Strategy learning with multilayer connectionist representations. Technical Report TR 87-509.3, GTE Labs, 1988.

[And93] Charles W. Anderson. Q-learning with hidden unit restarting. *Advances in Neural Information Processing Systems 5*, 1993.

Appendix A

SYMBOLS

γ: discount factor, set at .9

pd: output of pd algorithm

$error$: difference between the actual temperature and the desired temperature

$error'$: slope of temperature change

dT: change in temperature

dt: change in time

DT: exponential decay of Dt/dt

r: reinforcement

Q_{ON}: output of Qnet for hotplate on

Q_{OFF}: output of Qnet for hotplate off

ρ: measure of the radius of a gaussian node

β: learning rate

ϵ: error for updating the neural net