6 Nonlinear Control with Neural Networks

Derrick H. Nguyen
and Bernard Widrow
Department of Electrical Engineering, Stanford University

ABSTRACT

Neural networks can be used to solve highly nonlinear control problems. This chapter shows how a neural network can learn of its own accord to control a nonlinear dynamic system. An emulator, a multilayered neural network, learns to identify the system's dynamic characteristics. The controller, another multilayered neural network, next learns to control the emulator. The self-trained controller is then used to control the actual dynamic system. The learning process continues as the emulator and controller improve and track the physical process. Two examples are given to illustrate these ideas. A neural network is trained to control an inverted pendulum on a cart. This is a self-learning "broom-balancer." The "truck backer-upper," a neural network controller steering a trailer truck while backing up to a loading dock, is also demonstrated. The controller is able to guide the truck to the dock from almost any initial position. The technique explored here should be applicable to a wide variety of nonlinear control problems.

INTRODUCTION

Layered neural networks adapted by means of the back propagation algorithm discovered by Rumelhart, Hinton, and Williams (1986), Parker (1985), and Werbos (1974) are powerful tools for pattern recognition, associative memory, and adaptive filtering. In this chapter, adaptive neural networks will be used to solve nonlinear adaptive control problems that are very difficult to solve with conventional methods. The methodology shows promise for application to control problems that are so complex that analytical design techniques do not exist

and may not exist for some time to come. Neural networks can be used to implement highly nonlinear controllers with weights or internal parameters that can be determined by a self-learning process.

THE CONTROL PROBLEM

The standard representation of a finite-dimensional discrete-time plant is shown in Figure 1. The vector u_k represents the inputs to the plant at time k and the vector z_k represents the state of the plant at time k. The function $A(z_k, u_k)$ maps the current inputs and state into the next state. When the plant is linear, the usual state equation holds, where F and G are matrices:

$$z_{k+1} = A(z_k, u_k) = Fz_k + Gu_k. \qquad (1)$$

The function $A(z_k, u_k)$ would be nonlinear for a nonlinear plant.

A common problem in control is to provide the correct input vector to drive a nonlinear plant from an initial state to a subsequent desired state z_d. The typical approach used in solving this problem involves linearizing the plant around a number of operating points, building linear state-space models of the plant at these operating points, and then building a controller which employs state feedback to control the plant. Another approach involves open-loop optimal control as described by Bryson and Ho (1975). In this approach, an objective function measuring the performance of the system is defined, and the set of u_k that minimizes this function is analytically computed. These approaches are usually computationally intensive and requires considerable design effort.

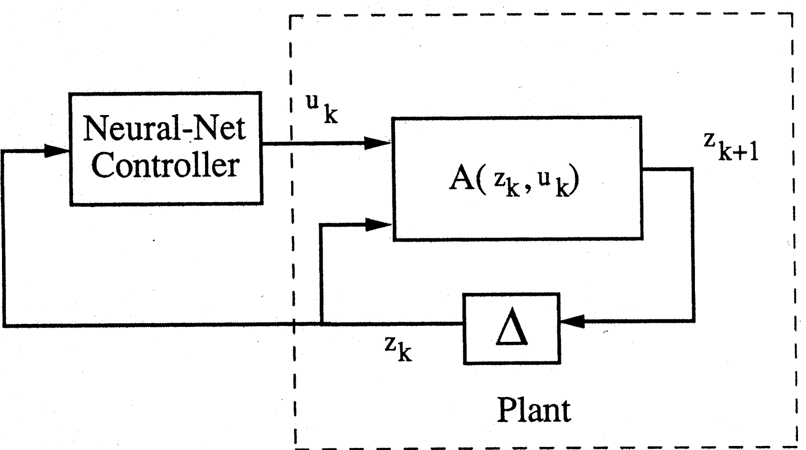

Figure 1. The plant and the controller.

In this presentation, the objective is to train a controller, in this case a neural network, to produce the correct signal u_k to drive the plant to the desired state z_d given the current state of the plant z_k (Figure 1). Each value of u_k over time plays a part in determining the state of the plant. Knowing the desired state, however, does not easily yield information about the values of u_k that would be required to achieve it.

A number of different approaches for training a controller have been described in the literature. They include reinforcement learning, which has been explored by Widrow, Gupta, and Maitra (1973), by Barto, Sutton, and Anderson (1983), and by Anderson (1989), inverse control, which has been explored by Widrow (1986) and by Psaltis, Sideris, and Yamamura (1988), and optimal control, which has been explored by Psaltis et al. (1988) and by Jordan (1988). The architecture and training algorithms presented in this chapter are novel in that they require little guidance from the designer to solve the control problem. This approach uses neural networks in optimal control by training the controller to maximize a performance function. The approach is different from Psaltis et al. (1988) in that the plant can be an unknown plant and plant identification is a part of the algorithm. A similar approach has been used by Widrow and Stearns (1985), Widrow (1986), and Jordan (1988).

SINGLE-STAGE CONTROLLERS

Given that the plant is in state z_k and that the objective is to drive it to the desired state z_d, it is simplest to train the controller to produce u_k so that the plant's next state z_{k+1} will be close to z_d. Therefore, we would like to minimize the error function C, where $E(\cdot)$ denotes an average and C is averaged over the set of possible starting states z_k:

$$C = E(\|z_d - z_{k+1}\|^2) \qquad (2)$$

A controller trained in this fashion is called a single-stage controller since it tries to make the next state z_{k+1} as close as possible to the desired state. The fact that the controller's output u_k will also affect the state of the plant after time $k + 1$ is ignored by the algorithm.

Plant Identification—Training the Plant Emulator

Before training the neural net controller, it is useful to train a separate neural net to behave like the plant. Specifically, the neural net is trained to emulate $A(z_k, u_k)$. The purpose of training this emulator will be explained later. Training the emulator is similar to plant identification in control theory, except that the plant identification here (Figure 2) is done automatically by a neural network capable of modeling nonlinear plants.

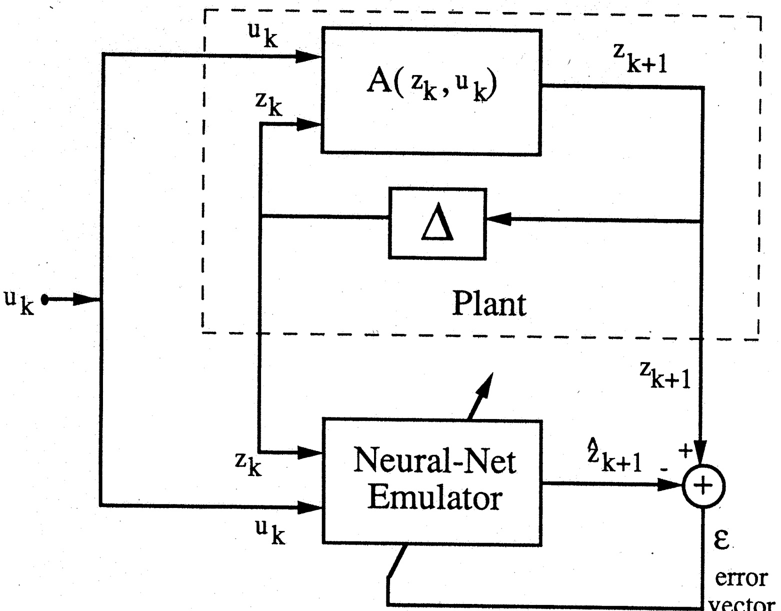

Figure 2. Training the neural net plant emulator.

In this chapter we assume that the states of the plant are directly observable without noise. A neural net with as many outputs as there are states, and as many inputs as there are states plus plant inputs, is created. The number of layers in the neural net and the number of nodes in each layer are presently determined empirically since they depend on the degree of nonlinearity of the plant.

In Figure 2, the training process begins with the plant in an initial state. The plant inputs are randomly generated. At time k, the input of the neural net is set equal to the current state of the plant z_k and the plant input u_k. The neural net is trained by back propagation to predict the next state of the plant, with the value of the next state of the plant z_{k+1} used as the desired response during training. This process is roughly analogous to the steps that would be taken by a human designer to identify the plant. In this case, however, the plant identification is done automatically by a neural network. The neural network emulator is presented here is analogous to the mental model of Rumelhart and the forward model of Jordan (1988).

Training the Neural Network Controller

After the plant emulator has been trained, we use it for the purpose of training the controller. The training process starts with the plant and the neural net plant

emulator in a state z_k. The initially untrained controller will output a control signal u_k to the plant, and the plant moves to the next state z_{k+1}. To train the controller, we need to know the error in the controller output u_k. Unfortunately, only the error in the plant state, $z_d - z_{k+1}$, is available. However, since the controller and the emulator are connected in series, they may be considered to be a single network. Therefore, the plant error $z_d - z_{k+1}$ may be propagated through the plant emulator (while keeping the weights of the emulator fixed) to get an error for the controller. This error is then used to update the weights of the controller by using the back-propagation algorithm. The emulator in a sense translates the error in the final plant state to the error in the controller output. The real plant cannot be used here because the error cannot be propagated through it. This is why the neural network emulator is needed.

An Example: The Inverted Pendulum (the Broom-Balancer)

The inverted pendulum on a cart problem (Figure 3) is used here to illustrate the approach. This is a standard problem that has been used by many researchers in the field of neural control in the past. In this problem, the goal is to design a controller to move a cart so that an inverted pendulum mounted on the cart stays vertical. For now, to keep the problem simple, the position of the cart is not controlled. To make the problem nonlinear, a "bang-bang" controller is used. The controller is only allowed to push the cart with maximum force to the left or to the right. This is accomplished by placing a hard quantizer between the

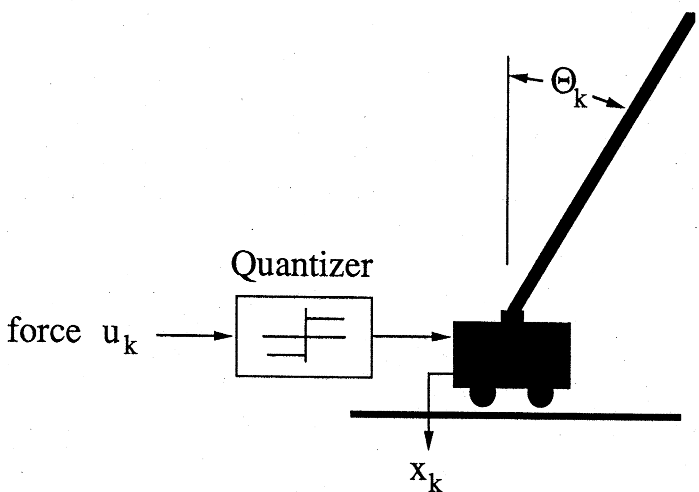

Figure 3. Cart/pendulum (broom-balancer) system.

controller and the cart/pendulum system. This quantizer is considered to be part of the plant in the algorithm described earlier. A linear controller is also possible. We have trained such a controller with the algorithm, and it learned much more quickly than the bang-bang controller.

For this example, the cart/pendulum system is simulated on a digital computer. The mass of the cart and of the pendulum are 1.0 kg and 0.1 kg, respectively, and the length of the pendulum is 1.0 m. The frictional force due the motion of the cart is proportional to its velocity with a proportional constant of 0.0005 N-s/m. The controller is allowed to push on the cart with a force of 10 N to either the left or the right. The cart position x is constrained to be between -1.4 and 1.4 m. A trial is started by placing the cart at the center of the track and the pendulum vertical, and a crash occurs when the magnitude of the pendulum angle θ exceeds 40° or when the cart runs out of track ($|x| \geq 1.4$). When a crash occurs, the trial is stopped, the cart and the pendulum are reset to their initial positions, and the next trial is initiated. The controller neural net is given the current state and it computes a new control signal every 0.03 sec.

In this problem, the state vector z_k consists of four variables: the position of the cart x, the velocity of the cart $\dot{x}$, the angular position of the pendulum θ, and the angular velocity of the pendulum $\dot{\theta}$. The values of these variables completely describe the state of the system at any moment in time. The cart position x is measured from the center of the track, and the pendulum θ is measured from

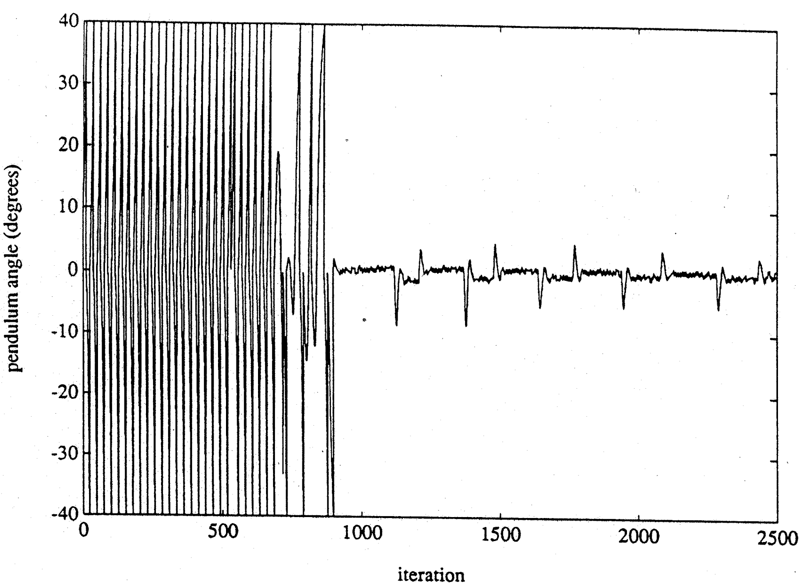

Figure 4. Control of broom-balancer angle.

vertical. The state variables and the force input to the plant are all scaled so that they are roughly between -1 and 1 before being fed into the neural networks.

The plant-emulator neural net has five inputs and it contains two layers of adaptive weights, six units in the hidden layer, and four units in the output layer. It is trained for about 40,000 iterations to emulate the real plant as described before. During this training process, the force input to the plant is generated randomly and uniformly between -10 and 10 N.

After the emulator has been trained, its weights are fixed and the controller neural net is trained to keep the pendulum vertical, that is, to keep its angle as close as possible to zero. The controller is a single adaptive unit with four inputs. Figure 4 shows the pendulum angle θ as functions of time during training. Note that the pendulum angle stays closer to zero as training time increases. The lengths of the trials remain relatively short, however, even after large training time, because the cart eventually runs into the ends of the track and the trial is stopped. Recall that the position of the cart x is not controlled.

THE MULTISTAGE NEURAL NETWORK CONTROLLER

For the cart/pendulum experiment the neural network controller was trained to keep the pendulum as vertical as possible ($\theta_{\text{desired}} = 0$), but not to control the other three state variables, the cart position and the velocities. Because of this, the cart wanders away from center of the track and eventually crashes into the ends.

One may expect that incorporating the desired response $x_{\text{desired}} = 0$ into the training process would improve performance since the controller would try to keep the cart near the center and the pendulum vertical, and the system would never crash. Unfortunately this does not work. The controller will never learn to do this even after long training.

The problem is that we are trying to drive two state variables, x_{k+1} and θ_{k+1}, to their desired values by adjusting the parameter u_k. However, in general it is impossible to control x_{k+1} and θ_{k+1} by choosing a single parameter u_k because there are not enough degrees of freedom to satisfy the requirements.

To circumvent this problem, we need to train the controller to look farther ahead. It will control x_{k+2} and θ_{k+2} by adjusting u_k and u_{k+1}. Note that x_{k+1} and θ_{k+1} take on values as they will as we control x_{k+2} and θ_{k+2}. (We may think of this as training the network to be more "farsighted." It may make a move that is bad in the near term but good in the long run.)

In general, if a plant has M state variables of which K are to be controlled, then it will take at least K time steps to bring them to their desired values if the plant is linear. If nonlinearities exist, more time steps may be required. The training algorithm will need to adjust the controller to outputs a series of K control signals u_k and to sense only the plant state at the end to compute the final

plant state error. It then uses this error to adjust the weights of the controller. A controller trained in this fashion is called a multistage controller after Bryson and Ho (1975).

Training the Neural Network Controller

Before training the controller, we train a neural network to emulate the plant we have described. Given that the emulator closely matches the plant dynamics, we use it for the purpose of training the controller. The controller learns to drive the plant emulator from an initial state z_0 to the desired state z_d in K time steps. Learning takes place during many trials or runs, each starting from an initial state and terminating at a final state z_K. The objective of the learning process is to find a set of controller weights which minimizes the error function C, where C is averaged over the set of initial states z_0:

$$C = E(\|z_d - z_K|^2), \tag{3}$$

The training process for the controller is illustrated in Figure 5. The training process starts with the neural net plant emulator set in a random initial state z_0. Because the neural net controller is initially untrained, it will output an erroneous control signal u_0 to the plant emulator and to the plant itself. The plant emulator will then move to the next state z_1, and this process continues for K time steps. At this point the plant is in the state z_K (Note that the number of time steps K is problem dependent and needs to be determined by the designer.)

We now would like to modify the weights in the controller network so that the square error $(z_d - z_K)^2$ will be less at the end of the next run. To train the controller, we need to know the error in the controller output u_k for each time step

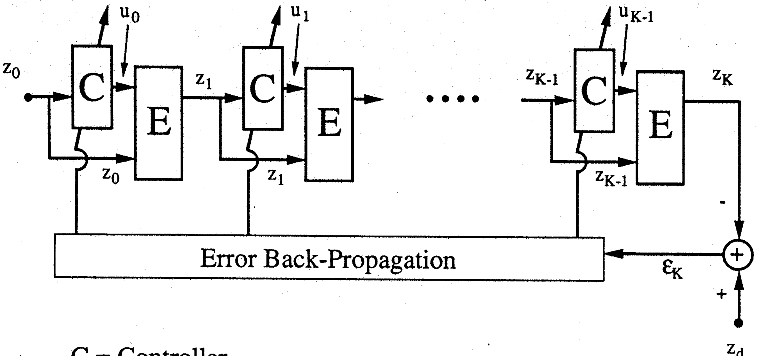

Figure 5. Training the controller.

k. Unfortunately, only the error in the final plant state, $z_d - z_K$, is available. Just as in the case of the single-stage controller, we can back-propagate the final plant error $z_d - z_K$ through the plant emulator to get an equivalent error for the controller in the Kth stage and use this error to train the controller. We then continue to back propagate the error through all K stages of the run, and the controller's weight change is computed for each stage. The weight changes from all the stages obtained from the back-propagation algorithm are added together and then added to the controller's weights. This completes the training for one run. (This method of breaking the feedback and unfolding the network in time has been explored for noncontrol applications by Rumelhart et al. (1986). See also the chapter in this book by Williams and Zipser.)

The algorithm described would require saving all the weight changes so that they can be added to the original weights at the end of the run. In practice, for simplicity's sake, the weight changes are added immediately to the weights as they are computed. This does not significantly affect the final result, since the weight changes are small and do not affect the controller's weights very much after one run. It is their accumulated effects over a large number of runs that improve the controller's performance.

Figure 5 represents the controller training process. For reasons of clarity, the details of error back propagation are not illustrated there, but are described above. Because the training algorithm is essentially an implementation of gradient descent, local minima in the error function may yield suboptimal results. In practice, however, a good solution is almost always achieved by using a large number of adaptive units in the hidden layers of the neural networks.

We will show two examples to illustrate the approach: a neural truck backer-upper and a broom-balancer that centers the cart as well as keeps the pendulum vertical.

An Example: The Truck Backer-Upper

The first example involves training a neural network to steer a truck while it backs up to a loading dock. Backing a trailer truck to a loading dock is a difficult exercise for all but the most skilled truck drivers. Anyone who has tried to back up a house trailer or a boat trailer will realize this. Normal driving instincts lead to erroneous movements, and a great deal of practice is required to develop the requisite skills.

When watching a truck driver backing toward a loading dock, one often observes the driver backing and then going forward and repeating this several times until finally backing to the desired position along the dock. The forward and backward movements help to position the trailer for successful backing up to the dock. A more difficult backing-up sequence would only allow backing with no forward movements permitted. The specific problem treated in this example is that of the design by self-learning of a nonlinear controller to control the steering

of a trailer truck while backing up to a loading dock from any initial position. Only backing up is allowed. Computer simulation of the truck and its controller has demonstrated that the algorithm described can train a controller to control the truck very well. An experimental neural controller consisting of two layers of adaptive weights with 25 adaptive neural units in the hidden layer and one unit in the output layer has exhibited excellent backing-up control. The trailer truck can be straight or initially jackknifed and aimed in many different directions, toward and away from the dock, but as long as there is sufficient clearance the controller appears to be capable of finding a solution.

Figure 6 shows a computer-screen image of the truck, the trailer, and the loading dock. The state vector representing the position of the truck consists of the following elements: θ_{cab}, the angle of the cab, $\theta_{trailer}$, the angle of the trailer, and $x_{trailer}$ and $y_{trailer}$, the Cartesian position of the rear of the center of the trailer. The definition of the state variables is illustrated in Figure 6.

The truck is placed at some initial position and is backed up while being steered by the controller. The run ends when the truck comes to the dock. The goal is to cause the back of the trailer to be parallel to the loading dock, that is, to make $\theta_{trailer}$ go to zero, and to have the point $(x_{trailer}, y_{trailer})$ be aligned as closely as possible with the point (x_{dock}, y_{dock}). The final cab angle does not matter. The controller will learn to achieve these objectives by adapting its weights to minimize the objective function C, where C is averaged over all training runs:

$$C = E(\alpha_1(x_{dock} - x_{trailer})^2 + \alpha_2(y_{dock} - y_{trailer})^2 + \alpha^3(0 - \theta_{trailer})^2). \quad (4)$$

The constants α_1, α_2, and α_3 are chosen by the designer to weight the importance of each error component.

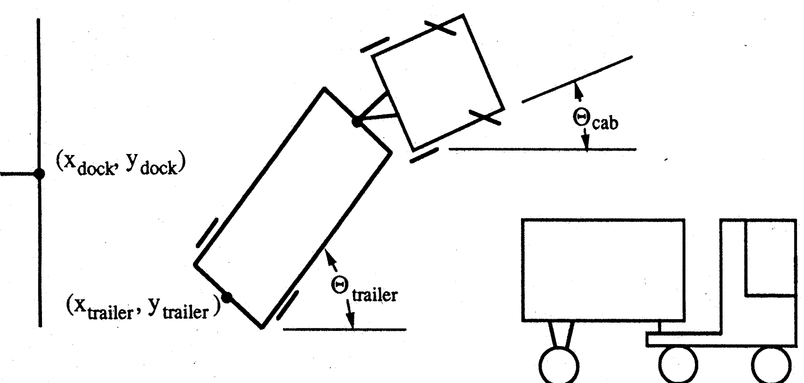

Figure 6. Truck, trailer, and loading dock.

Training

As described in the previous section, the learning process for the truck backer-upper controller involves two stages. The first stage trains a neural network to be an emulator of the truck and trailer kinematics. The second stage enables the neural network controller to learn to control the truck by using the emulator as a guide. The control process consists of feeding the state vector z_k to the controller, which in turn provides a steering signal u_k between -1 (hard right) and $+1$ (hard left) to the truck (k is the time index). Each time step, the truck backs up by a fixed small distance. The next state is determined by the present state and the steering signal, which is fixed during the time step.

The process used to train the emulator is shown in Figure 2. The emulator used in this example is a network consisting of two layers of adaptive weights with 25 units in the hidden layer and 4 units in the output layer. A suitable architecture for this network was determined by experiment. Experience shows that the choice of network architecture is important but a range of variation is permissible (See the chapters in this book on the theory of feedforward networks for discussions on network architecture.) The emulator network has five inputs corresponding to the four state variables x_k and the steering signal u_k, and four outputs corresponding to the four next state variables z_{k+1}.

During training, the truck backs up randomly, going through many cycles with randomly selected steering signals. The emulator learns to generate the next positional state vector when given the present state vector and the steering signal. This is done for a wide variety of positional states and steering angles. The emulator is adapted by means of the back propagation algorithm. By this process, the emulator "gets the feel" of how the trailer and truck behave. Once the emulator is trained, it can then be used to train the controller.

Refer to Figure 5. The identical blocks labeled C represent the controller net. The identical blocks labeled E represent the truck and trailer emulator. Let the weights of C be chosen at random initially. Let the truck back up. The initial state vector z_0 is fed to C whose output sets the steering angle of the truck. The backing-up cycle proceeds with the truck backing a small fixed distance so that the truck and trailer soon arrive at the next state z_1. With C remaining fixed, a new steering angle is computed for state z_1, and the truck backs up a small fixed distance once again. The backing-up sequence continues until the truck hits something and stops. The final state z_K is compared with the desired final state (the rear of the trailer parallel to the dock with proper positional alignment) to obtain the final state error vector ϵ_K. (Note that in reality there is only one controller C. Figure 5 shows multiple copies of C for the purpose of explanation.) The error vector contains four elements which are the errors of interest in $x_{trailer}$, $y_{trailer}$, $\theta_{trailer}$, and θ_{cab} and are used to adapt the controller C. The final angle of the cab θ_{cab} does not matter, so the element of the error vector due to

θ_{cab} is set to 0. Each element of the error vector is also weighted by the corresponding α_i of Equation 4.

The method of adapting the controller C is illustrated in Figure 5. The final state error vector ϵ_K is used to adapt the blocks labeled C, which are maintained identical to each other throughout the adaptive process. The controller C is a neural network with two layers of adaptive weights. The hidden layer has the six state variables as inputs and contains 25 adaptive units. The output layer has one adaptive unit and produces the steering signal as its output.

The controller C is adapted as described in the previous section. The weights of C are chosen initially at random. The initial position of the truck is chosen at random. The truck backs up, undergoing many individual back-up moves, until it comes to the dock. The final error is then computed and used by back propagation to adapt the controller. The error is used to update the weights as it is back-propagated through the network. This way, the controller is adapted to minimize the sum of the squares of the components of the error vector. The entire process is repeated by placing the truck and trailer in another initial position and allowing it to back up until it stops. Once again, the controller weights are adapted, and so on.

The controller and the emulator are neural networks with two layers of adaptive weights, each containing 25 hidden units. Thus, each stage of Figure 5 amounts to four layers of adaptive weights. The entire process of going from an initial state to the final state can be seen from Figure 5 to be analogous to a neural network having a number of layers of adaptive weights equal to four times the number of backing-up steps when going from the initial state to the final state. The number of steps K varies of course with initial position of the truck and trailer relative to the position of the target, the loading dock. In this experiment, we use initials positions of the truck that require from as few as four time steps to as many as 50 time steps.

The diagram of Figure 5 was simplified for clarity of presentation. The output error actually back propagates through the E blocks and C blocks. Thus, the error used to adapt each of the C blocks does originate from the output error ϵ_K, but travels through the proper back-propagation paths. For purposes of back propagation of the error, the E blocks are the truck emulator. But the actual truck kinematics are used when sensing the error ϵ_K itself.

The training of the controller was divided into several "lessons." In the beginning, the controller was trained with the truck initially set to points very near the dock and the trailer pointing at the dock. Once the controller was proficient at working with these initial positions, the problem was made harder by starting the truck farther away from the dock and at increasingly difficult angles. This way, the controller learned to do easy problems first and more difficult problems after it mastered the easy ones. There were 16 lessons in all. In the easiest lesson the trailer was set about half a truck length from the dock in the

x direction pointing at the dock, and the cab at a random angle between $\pm 30°$. In the last and most difficult lesson the rear of the trailer was set randomly between one and two truck lengths from the dock in the x direction and ± 1 truck length from the dock in the y direction. The cab and trailer angle was set to be the same, at a random angle between $\pm 90°$. (Note that uniform distributions were used to generate the random parameters.) The controller was trained for about 1000 truck backups per lesson during the early lessons, and 2000 truck backups per lesson during the last few. It took altogether about 20,000 backups to train the controller.

Results

The controller learned to control the truck very well with the above training process. Near the end of the last lesson, the root-mean-square error of $y_{trailer}$ was about 3% of a truck length. The root-mean-square error of $\theta_{trailer}$ was about 7°. There is no error in $x_{trailer}$ since a truck backup is stopped when $x_{trailer} = x_{dock}$. One may, of course, trade off the error in $y_{trailer}$ with the error in $\theta_{trailer}$ by giving them different weights in the objective function during training.

After training, the controller's weights were fixed. The truck and trailer were placed in a variety of initial positions, and backing up was successfully done in each case. A backup run when using the trained controller is demonstrated in Figure 7. Initial and final states are shown on the computer screen displays, and the backing up trajectory is illustrated by the time-lapse plot. The trained controller was capable of controlling the truck from initial positions it had never seen. For example, the controller was trained with the cab and trailer placed at angles between $\pm 90°$, but was capable of backing up the truck with cab and trailer placed at any angle provided that there was enough distance between the truck and the dock.

A More Sophisticated Objective Function

The truck controller was trained to minimize only the final state error. One can also train it to minimize total path length or control energy in addition to the final state error. For example, the objective function to minimize control energy is the following, with C averaged over all training trials.

$$C = E\left[\alpha_1(x_{dock} - x_{trailer})^2 + \alpha_2(y_{dock} - y_{trailer})^2 + \alpha_3(\theta_{dock} - \theta_{trailer})^2 + \alpha_e \sum_{k=0}^{K-1} u_k^2 \right]. \tag{5}$$

A simple change is made to the algorithm to minimize this objective function. In the original algorithm, the equivalent error for the controller at each time step k is

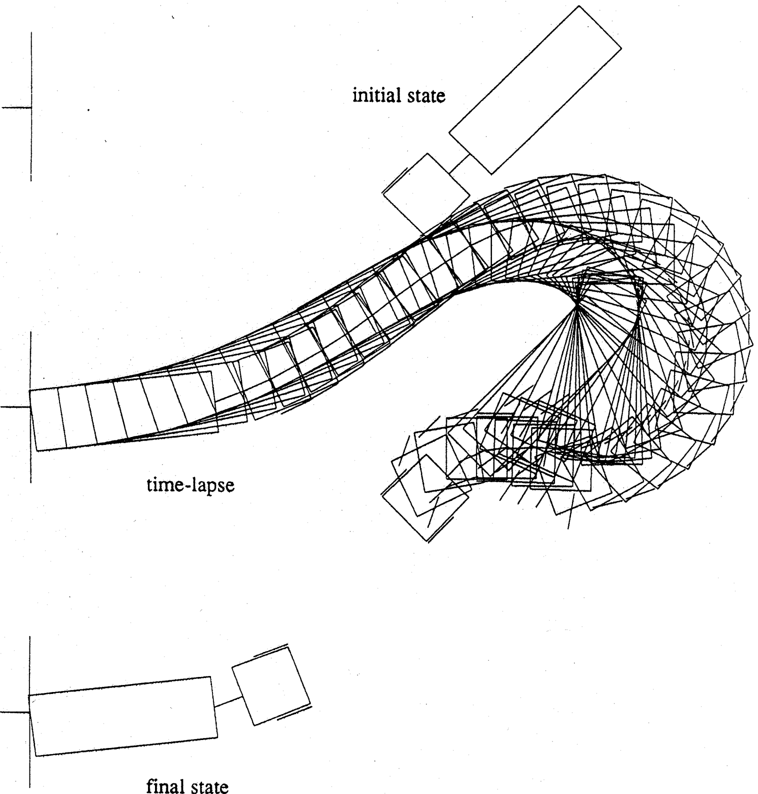

Figure 7. A backing-up example.

computed during the backward pass of the back-propagation algorithm. It is easy to show that control energy can be minimized by adding $-\alpha_4 u_k$ to the equivalent error of the controller at each time step. The modified equivalent error is then back-propagated through the controller to update the controller's weights just as before. This change makes sense, since using $-\alpha_4 u_k$ as an error in u_k causes the controller to learn to make u_k smaller in magnitude.

Training the controller to minimize control energy would cause it to drive the truck to the dock with as little steering as possible. An example with the controller trained in this manner is shown in Figure 8. This example uses the same truck and trailer initial position as with the example of Figure 7. Note that the path of the truck controlled by the new controller contains fewer sharp turns. Of course, the final state error increases somewhat because of the new control objective.

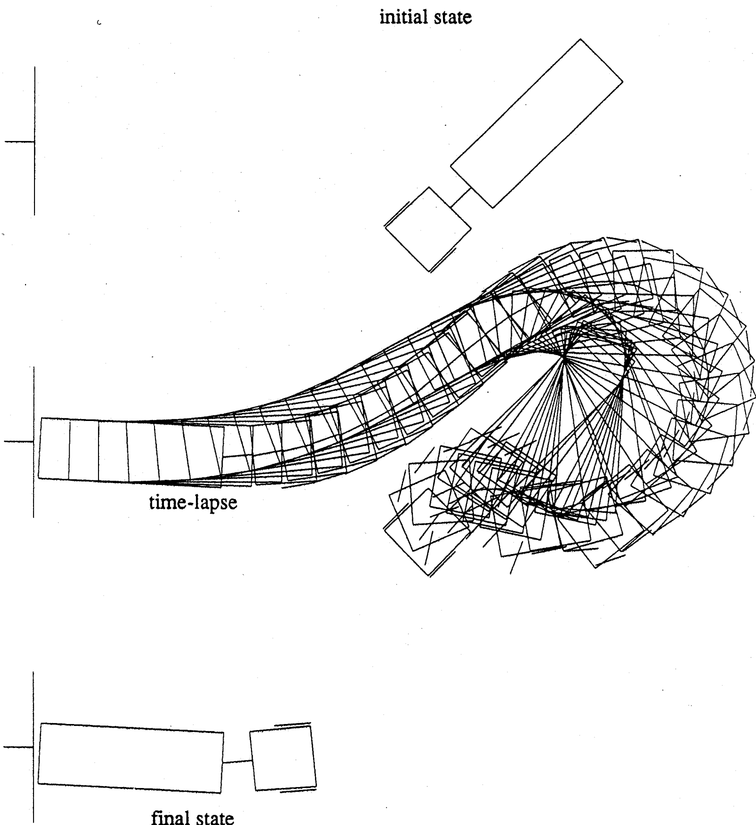

Figure 8. Backing up while minimizing control energy.

The Broom-Balancer Revisited

The algorithm described, which was used to successfully train a controller for the trailer truck, was used to train a controller for the cart/pendulum system to keep the cart near the center of the track and the pendulum vertical at the same time. The overall picture was the same as before (Figure 3). In this case however, the multistage training algorithm was used with the objective of all four state variables equaling zero. The system was run for 15 time steps or until it crashed, whichever came first; then the weights of the controller were updated using the multistage training algorithm. The system was then run for 15 more time steps and training repeated, and so on. The training process for the controller required

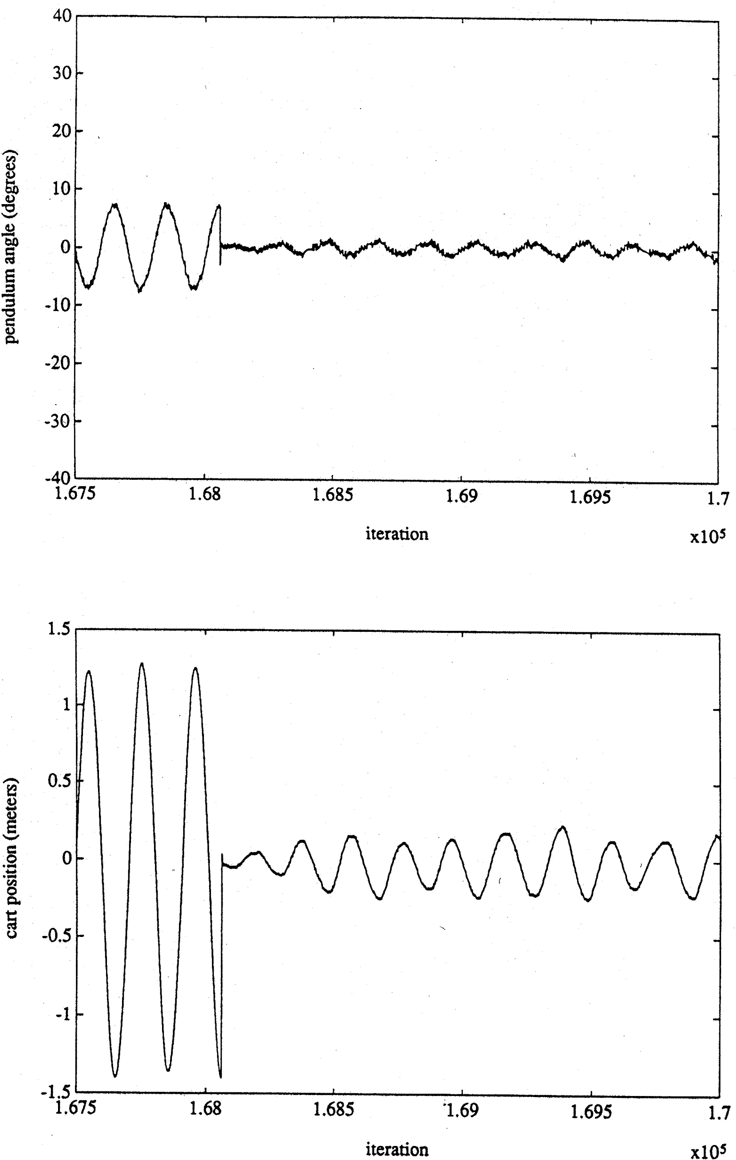

Figure 9. Control of broom-balancer position and angle.

about 200,000 time steps. Figure 9 shows the cart position and pendulum angle as a function of time near the end of training. It can be seen that after training, the controller established a limit cycle, keeping the pendulum from falling and the cart near $x = 0$ indefinitely.

SUMMARY

The multistage training algorithm for neural network controllers is a very powerful algorithm. It presents us with a way of controlling nonlinear systems that are difficult to work with using other methods. The truck emulator in the form of a neural network was able to represent the trailer and truck when jackknifed, in line, or in any condition in between. Nonlinearity in the emulator was essential for accurate modeling of the kinematics. The angle between truck and trailer was not small, and thus $\sin \theta$ could not be represented approximately as θ. Controlling the nonlinear kinematics of the truck and trailer required a nonlinear controller, implemented by another neural network. Self-learning processes were used to determine the parameters of both the emulator and the controller. Thousands of backups were required to train these networks, requiring an hour or so on a workstation. Without the learning process, however, substantial amounts of human effort and design time would have been required to devise the controller.

The multistage controller learns to solve sequential decision problems. The control decisions made early in the control process have substantial effects upon final results. Early moves may not always be in a direction to reduce error, but they position the plant for ultimate success. In many respects, the truck backer-upper learns a control strategy that is like a dynamic programming problem solution. The learning is done in a layered neural network. Connecting signals from one layer to another corresponds to the idea that the final state of a given backing-up cycle is the same as the initial state of the next backing-up cycle.

Future research will be concerned with

- Determination of complexity of emulator as related to complexity of the system being controlled
- Determination of complexity of controller as related to complexity of emulator
- Determination of convergence and rate of learning for emulator and controller
- Proof of robustness of control scheme
- Analytic derivation of nonlinear controller for truck backer-upper, and comparison with self-learned controller
- Relearning in the presence of movable obstacles
- Exploration of other areas of application of self-learning neural networks

ACKNOWLEDGMENTS

This research was sponsored by SDIO Innovative Science and Technology Office and managed by ONR under contract #N00014-86-K-0718, by the Department of the Army Belvoir R D & E Center under Contract #DAAK70-89-K-0001, by NASA Ames under Contract #NCA2-389, by Rome Air Development Center under Subcontract #E-21-T22-S1 with the Georgia Institute of Technology, and by grants from the Thomson CSF Company and the Lockheed Missiles and Space Company.

This material is based on work supported under a National Science Foundation Graduate Fellowship. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Anderson, C. W. (1989). Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine, 3,* 31–37.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics, 5,* 834–846.

Bryson, A. E., & Ho, Y. (1975). *Applied optimal control.* Washington, DC: Hemisphere.

Jordan, M. I. (1988). *Supervised learning and systems with excess degrees of freedom* (COINS Report 88-27). Cambridge, MA: Massachusetts Institute of Technology.

Parker, D. B. (1985). *Learning logic* (Technical Report TR-47). Cambridge, MA: Massachusetts Institute of Technology, Center for Computational Research in Economics and Management Science.

Psaltis, D., Sideris, A., & Yamamura, A. A. (1988). A multilayered neural network controller. *IEEE Control Systems Magazine, 3,* 17–21.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Volume 1: Foundations.* Cambridge, MA: MIT Press.

Werbos, P. J. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* Unpublished doctoral dissertation. Cambridge, MA: Harvard University.

Widrow, B. (1986). Adaptive inverse control. *Proceedings of the 2nd International Federation of Automatic Control.*

Widrow, B., Gupta, N. K., & Maitra, S. (1973). Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man, and Cybernetics, 5,* 455–465.

Widrow, B., & Stearns, S. D. (1985). *Adaptive signal processing.* Englewood Cliffs, NJ: Prentice-Hall.

DEVELOPMENTS IN CONNECTIONIST THEORY
David E. Rumelhart, Editor

Gluck/Rumelhart · Neuroscience and Connectionist Theory

Ramsey/Stich/Rumelhart · Philosophy and Connectionist Theory

Chauvin/Rumelhart · Backpropagation: Theory, Architectures, and Applications

BACKPROPAGATION

Theory, Architectures, and Applications

Edited by
Yves Chauvin
Stanford University
and Net-ID, Inc.
David E. Rumelhart
Department of Psychology
Stanford University

LAWRENCE ERLBAUM ASSOCIATES, PUBLISHERS
1995 Hillsdale, New Jersey Hove, UK