Neural Networks
for
Truck Backer-Upper Control System

Jyh-Shan Chang Jenn-Huei Lin Tzi-Dar Chiueh
Department of Electrical Engineering
National Taiwan University

Abstract

Artificial neural networks have been studied for many years in the hope of achieving human-like performance in many fields. One of these fields is to use neural networks to solve highly nonlinear control systems. Multilayer feedforward neural network has proven to be a very powerful tool in this field. Nevertheless, such a network suffers from a very time-consuming training procedure. In this paper, based on radial-basis function and recurrent neural network, we develop two different neural network systems for backing up a computer simulated truck to a loading dock in a planar parking lot. In our simulations, these two neural networks are all capable of learning from the training samples and performing generalization, therefore provide viable alternatives to the multilayer feedforward neural network for real-world applications.

1 Introduction

Since the first appearance of learning theories and neural processing in the 1940s, many researchers have focused on the theoretical frameworks and mathematical foundations of neural networks, or artificial neural networks to be more precise. Having understood the computational principles, we may be able to use artificial neural network as a tool to solve currently difficult-to-solve or even unsolved problems.

Neural networks consist of many parallel processing units which usually perform some nonlinear computations. These units are known as neurons. There are connections between these neurons specified by some varying strengths known as synaptic weights. These weights are adjusted to store the knowledge acquired during a training process. The procedure used to perform the training is called a learning algorithm. The purpose of learning algorithm is to gradually update the synaptic weights of the neural networks in order to obtain the desired responses in an optimal way. An excellent introduction of neural networks is provided by R.L. Lippmann[1]. It is the abilities of learning, adaptation, generalization, and fault tolerance that make the neural networks suitable tools for real-world applications in signal processing, speech recognition, visual perception, control and robotics.

2 Neural Networks

Successful applications of neural networks cover a wide range in diverse fields. Most of the applications deal with classification or function approximation and interpolation. Interpolation involves the construction of a system which learns a mapping from the relationship of the input and output sample spaces. This system also generalizes when new inputs are presented to it.

2.1 Multilayer feedforward network

The most widely used neural network model that is used to solve problems by researchers is probably the multilayer feedforward network. Typically, this kind of networks consist of three or more layers. A multilayer feedforward network with one hidden layer is shown in Figure 1. The first layer is composed of a set of source nodes where the input samples are presented. There are one or more intermediate layers known as hidden layers, which lie between the input layer and the final output layer. These intermediate layers and the output layer are composed of computational nodes which perform predefined nonlinear operation. The knowledge of the network that has been learned from the input-output pairs during a training process is stored in the synaptic connections between these layers in the form of adjustable weights.

Typically, the adjustments of weights are learned with a well known and popular algorithm — error back-propagation algorithm. This algorithm basically has two stages: a data feedforward stage and an error back-propagation stage. In the data

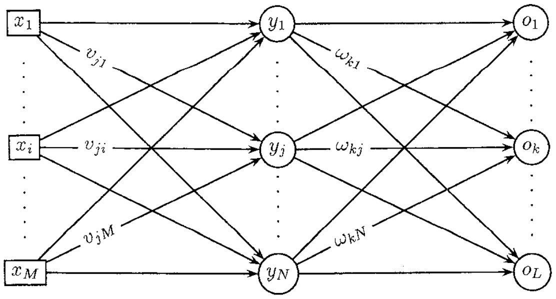


Figure 1: A multilayer feedforward network with one hidden layer.

0-7803-2646-6

feedforward stage, an input pattern is applied to the input layer, then the output of the first layer is propagated as the input to the intermediate layer through the network until it reaches the output layer and produces a network output. During the feedforward stage, all weights are fixed. It is until the backward stage that the weights get updated. The network output is compared to the desired output to generate an error signal. This error signal is crucial in the backward stage. During the error back-propagation stage, this error signal is propagated from the output layer to the intermediate layer. Each of the synaptic weights between layers along the back-propagation path is then updated or adjusted with a small amount which depends on how much it contributes to the output error. The purpose of updating the synaptic weights is to drive the network output closer to the desired output.

The error back-propagation algorithm is summarized as follows:

1. During feedforward stage, each node j in the hidden layer produces an output y_j:

$$y_j = f(net_j = \sum_{i=1}^{M} v_{ji}x_i) \tag{1}$$

   where f is an activation function. Typical examples are logistic function and hyperbolic tangent.

2. During feedforward stage, each node k in the output layer produces an output o_k:

$$o_k = f(net_k = \sum_{j=1}^{N} w_{kj}y_j) \tag{2}$$

3. Define the error function:

$$E = \frac{1}{2}\sum_{k=1}^{L}(d_k - o_k)^2 \tag{3}$$

   where d is the desired output.

4. The weights that connect the hidden layer and the output layer are then updated by gradient descent during error back-propagation stage:

$$w_{kj} = w_{kj} + \Delta w_{kj} = w_{kj} - \eta\frac{\partial E}{\partial w_{kj}}$$

$$= w_{kj} - \eta\frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}}$$

$$= w_{kj} + \eta(d_k - o_k)f'(net_k)y_j \tag{4}$$

   where the learning-rate η is a constant that determines the rate of learning.

5. The weights that connect the input layer and the hidden layer are also updated by gradient descent during error back-propagation stage:

$$v_{ji} = v_{ji} + \Delta v_{ji} = v_{ji} - \eta\frac{\partial E}{\partial v_{ji}}$$

$$= v_{ji} - \eta\sum_{k=1}^{L}\frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial net_k}\frac{\partial net_k}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial v_{ji}}$$

$$= v_{ji} + \eta\sum_{k=1}^{L}(d_k - o_k)f'(net_k)w_{kj}f'(net_j)x_i \tag{5}$$

2.2 Radial-Basis Function Networks

The radial-basis functions were first used in neural networks by Broomhead and Lowe and then improved by many other researchers[2][3]. Examples of applications are Kohonen's self organize feature map (SOFM) and restricted Coulomb energy (RCE) networks. Radial-basis function networks are quite different from other neural networks. As we can see from Figure 2, radial-basis function network contains three layers: one input layer, one hidden layer, and also one output layer. Each layer is fully connected to the one following. The first layer serves as source nodes. The hidden layer is made up of radial-basis function nodes with a predetermined centroid for each node. The output of each node is then computed as a function of the distance from the input to the centroid of this node. Typical choices of the functions are Gaussian function, thin plate spline, multiquadratic function, etc. The most popular and widely used one is probably Gaussian function, which has a peak at the center and decreases monotonically as the distance increases. This is followed by weighted summing the outputs of radial-basis function nodes to give the final network outputs. In general, a radial-basis function network is specified by three sets of parameters: the centroid c_i, the spreading parameter σ_i that controls the shape of each node, and the synaptic weights w_{ji} that connect the radial-basis layer to the output layer.

Based on the parameters required for the radial-basis function, the training algorithm can be determined in three successive steps:

1. First, the centroids are determined.

   The simplest technique is to choose randomly from the training data as the centroids. However, in such case the number of centroids must be relatively large and distributed evenly in order to cover the entire input domain. Many well-known clustering algorithms are available for determining the centroids. We have applied an improved approach known as "K-means clustering algorithm" to determine the centroids in our case. The K-means clustering algorithm finds a set of cluster centers and a partition of the training data into these clusters. Each of the training data is partitioned to the cluster with the nearest center. Each cluster center then define a RBF node in the RBF network.

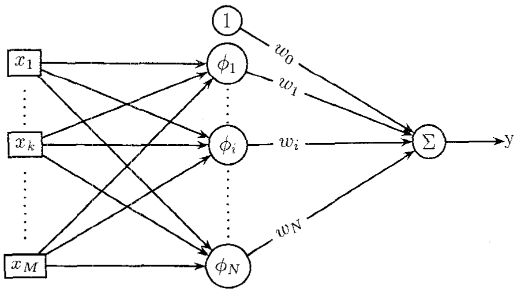


Figure 2: A Radial-Basis-Function network with one output node.

2. The spreading parameters can be determined by various methods. In our case, the P-nearest-neighbor-rule is used

$$\sigma_i = \frac{1}{P}\sum_{j=1}^{P}[\|c_i - c_j\|^2]^{\frac{1}{2}} \tag{6}$$

where c_j is the P-nearest-neighbor of c_i.

3. Having obtained the centroids and spreading parameters, the network output can then be computed as

$$y(k) = \omega_0 + \sum_{i=1}^{N}\omega_i\phi_i(\|x(k) - c_i\|) \tag{7}$$

for the input sample $x(k)$. ϕ, the kernel function, is chosen to be a Gaussian function. Let $\hat{y}(k)$ denote the desired output for the input sample $x(k)$. In our case, the standard least-mean-square algorithm will be used to adjust the weights which connect the radial-basis function layer and the output layer as

$$\omega_i = \omega_i + \frac{d\omega_i}{dt} = \omega_i + \mu e_k\phi_i(\|x(k) - c_i\|) \tag{8}$$

where μ is the learning rate and $e_k = \hat{y}(k) - y(k)$ denotes the output error.

2.3 Recurrent Neural Network

A recurrent neural network differs from a feedforward neural network in the fact that there are no restrictions on the placements of synapes in a recurrent network. This makes all kinds of feedbacks and connections possible and achieves the full computational power of neural networks. With such a general architecture, recurrent neural networks have important capabilities not found in feedforward networks, such as attractor dynamics and the ability to identify a time-varying system.

Various learning algorithms in recurrent neural networks have been proposed. Algorithms for associative memory networks which are recurrent networks settling to stable states have been proposed by Hopfield[4] and Pineda[5]. However, Jordan[6], Gallant and King[7], and Pearlmutter[8] develop algorithms to train recurrent networks to handle time-varying systems. The algorithm to be used in our paper is real-time recurrent learning algorithm for completely recurrent networks running in continually sampled time devised by R. J. Williams and D. Zipser[9].

The real-time recurrent learning algorithm exhibits the generality of the backpropagation-through-time approach without the growing memory requirement in arbitrarily long training sequence. With the feedbacks from the output layer, a small recurrent neural network can well simulate a time-varying and nonlinear system.

A typical real-time recurrent neural network is shown in Figure 3. It consists of two layers: output layer and input layer. The output layer includes output and hidden neurons. Some or all of the output/hidden neurons are delayed and fedback to the input layer. Therefore, the input layer consists of delayed output and external input. The algorithm proceeds as follows:

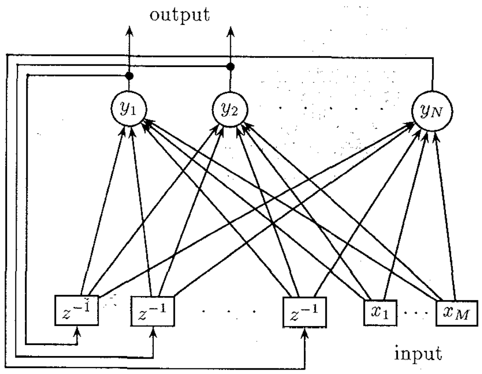


Figure 3: A Recurrent Neural Network.

1. Forward process: compute output y_j for all $j \in C$ as

$$y_j(n) = f(v_j(n)) = f(\sum_{i \in A \cup B} w_{ji}(n)u_i(n)) \tag{9}$$

2. Backward process: with

$$\Pi_{kl}{}^{j}(0) = 0$$

$$\Pi_{kl}{}^{j}(n+1) = f'(v_j(n))[\sum_{i \in B} w_{ji}\Pi_{kl}{}^{i}(n) + \delta_{kj}u_l(n)] \tag{10}$$

compute error gradient as

$$\nabla_w e(n+1) = -\sum_{j \in O} e_j(n)\Pi_{kl}{}^{j}(n+1) \tag{11}$$

3. Weight updates:

$$w_{kl}(n+1) = w_{kl}(n) - \eta\nabla_w e(n+1)$$

$$= w_{kl}(n) + \eta\sum_{j \in O} e_j(n)\Pi_{kl}{}^{j}(n+1) \tag{12}$$

where

- A: external input neurons
- B: feedback output/hidden neurons
- O: desired output neurons
- C: all output/hidden neurons
- u_i: neurons of input layer where $i \in A \cup B$
- δ_{kj}: Kronecker delta function
- η: learning rate
- w_{ji}: weight between output/hidden neuron j and input neuron i
- f: logistic function, $f(v) = \frac{1}{1+exp(-v)}$

3 Truck Backer-Upper Control System

Many of the control problems involve input-output mapping. In this paper, we are going to use the neural networks that we have mentioned in the previous sections as tools to solve a control problem. The problem we have chosen is a truck backer-upper control system. This truck control system, which corresponds to a highly nonlinear control problem, is originally proposed by Nguyen and Widrow[10][11]. We are going to develop the neural truck control system which is based on different architectures and training algorithms.

Figure 4 shows the simulated truck and loading zone. The truck is determined by three state variables ϕ, x, and y where

$$\begin{array}{rcl} 0 & \leq x \leq & 100 \\ -90 & \leq \phi \leq & 270 \\ -30 & \leq \theta \leq & 30 \end{array} \tag{13}$$

The trained neural network is going to generate a proper steering angle θ to control the truck while backing up to a loading dock from an arbitrary initial position and arbitrary angle ϕ of the truck with the horizontal axis. During the process, only backing up is allowed, and the truck stops when it hits the loading dock.

The architecture of our neural truck controller is shown in Figure 5. The controller consists of two portions. The first part is a neural steering controller which generates an output of the steering-angle to be as the input to the next part of the system. The appropriate steering-angle depends on the state variables (x, y, ϕ). The next part of the system is a truck emulator which is used to estimate the next state of the truck according to the current state of the truck and the input of steering-angle. Instead of a real emulator, it is a simple kinematic equation[12]. If the truck moves backward from (x, y) to (x', y') at an iteration, then

$$\begin{array}{rcl} x' & = & x + r\cos(\phi') \\ y' & = & y + r\cos(\phi') \\ \phi' & = & \phi + \theta \end{array} \tag{14}$$

where r denotes the fixed driving distance of the truck for all backing up movements.

We will use back-propagation network, radial-basis function network, and recurrent neural network separately as the steering controller. That is we are going to train these neural networks with some sets of input-output samples. Since the optimal steering-angle for all the combinations of states are not available for the training sets, we have developed a simple fuzzy truck controller similar to the one proposed by S.G. Kong and B. Kosko[12] with 9 fuzzy associative memory (FAM) instead of 35. This fuzzy truck controller only serves as a nonlinear function generator while the neural networks serve as function approximators. They will learn from the samples generated by the fuzzy truck controller and back up the truck appropriately.

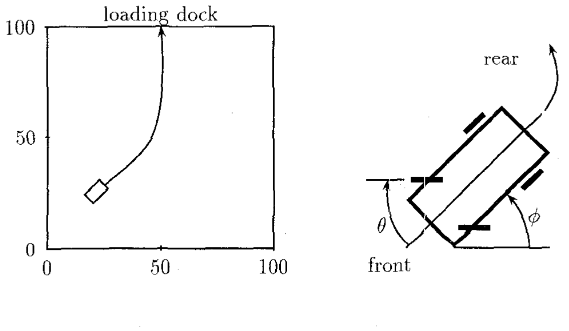


Figure 4: Simulated truck and loading zone.

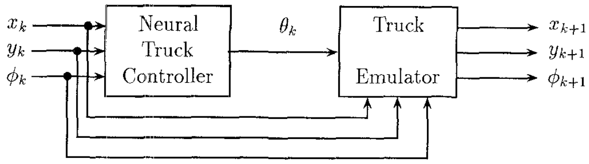


Figure 5: Neural truck controller system.

4 Simulations and Discussions

In order to train the neural network controllers, we will need training samples to teach those networks. We have generated 525 samples from eleven different initial positions. These initial positions start from $(0, 20)$ with a 5 increment in x direction until 100 is reached. At each initial position, the angle of the truck with the horizontal is changed from $-90°$ to $270°$ with a $15°$ increment. For each combination of (x, ϕ), the fuzzy controller produces an output steering-angle θ. This vector (x, ϕ, θ) constitutes a training sample for the neural network controllers. These samples defined a control surface in a three dimensional product-space. The neural networks will learn from the samples and generalize a control surface close to the fuzzy one.

The first experiment we conduct is a three layer feedforward neural network. The first layer has three nodes served as the inputs of the truck state (x, y, ϕ). Various numbers of hidden nodes are used in the hidden layer for the purpose of comparison. The final layer has one node and produces the steering-angle as its output. The error back-propagation algorithm is used to adjust the weights between layers. The 525 training samples are presented to the network in one epoch. The output error for each sample is accumulated. This error is then used to adjust the weights as mentioned in the previous section. The process is continued until the output error is converged to a desired small value. The control surfaces of the trained networks are shown in Figure 6. As we can see from these control surfaces, the multilayer feedforward network has shown its superiority in generalization. Even with a small number of hidden nodes, the network can still learn from the samples and perform generalization.

The radial-basis function neural network controller is also trained for comparison. The training samples are initially classified into a number of clusters to make close samples in the input space to be placed in the same cluster. Many well-known clustering algorithms are available. We have used the K-means algorithm in our case. The geometric mean of each cluster is taken as the centroid vector. Various methods are also available for determining the spreading parameter too. We simply use the P-nearest-neighbor rule to determine the spreading parameter for each cluster. After the centroids and spreading parameters are determined, the only work left is to adjust the weights to have the desired outputs based on the accumulated error in the output in one epoch. The adjustments of weights are continued until the error converges. The control surfaces of the trained networks are shown in Figure 7. The error curves during training are shown in Figure 8 with the curves generated by multilayer feedforward network for comparison.

Comparing Figure 7 to the control surfaces generated by the multilayer feedforward network in Figure 6, we can find out that the RBF network do require more RBF nodes than the multilayer feedforward network in order to have similar generalization property. The reason is that the RBF network is mainly composed by locally-tuned RBF nodes. Each locally-tuned node will constitute a control surface, determined by the centroid and the spreading parameter, to cover a portion of the input space. The control surface generated by the RBF network is composed by a weighted accumulation of the surface generated by each of the RBF node. Therefore, it is fundamental to have sufficient number of RBF nodes to cover the entire input space. The number of RBF nodes required in the network is a trade-off. If the number is too small, then the network is unable to learn the relevant information from the samples therefore fails to generalize. If the number is too large, the network begin to memorize and therefore sacrifice the smoothness property for generalization. We can also notice that the converge speed of these networks from Figure 8. The radial-basis function network only requires a few hundreds to converge to a certain degree while the multilayer feedforward network requires a few thousands. Moreover, in the least-mean-square sense, the time required to adjust the weights for the radial-basis function is less than the time required for error back-propagation in the multilayer feedforward network in one epoch.

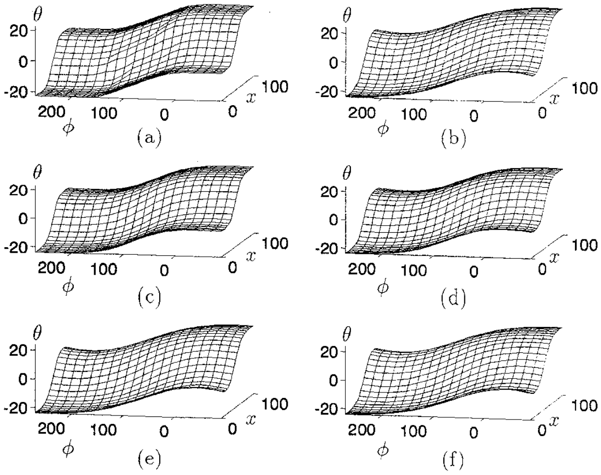


Figure 6: Control surfaces: (a) fuzzy controller; (b) BP controller with 5 hidden nodes; (c) BP controller with 10 hidden nodes; (d) BP controller with 25 hidden nodes; (e) BP controller with 50 hidden nodes; (f) BP controller with 100 hidden nodes.

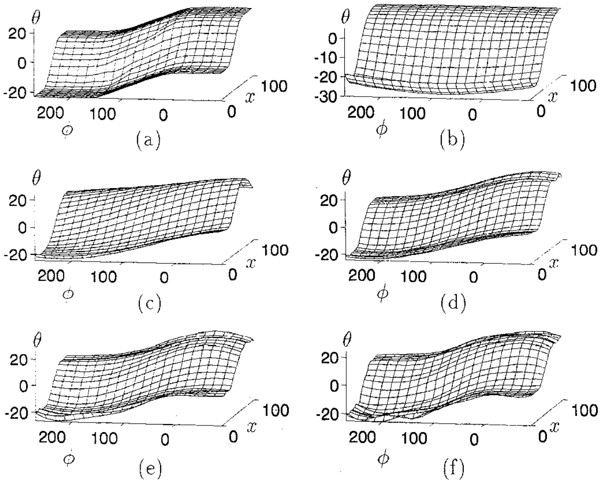


Figure 7: Control surfaces: (a) fuzzy controller; (b) RBF controller with 5 RBF nodes; (c) RBF controller with 10 RBF nodes; (d) RBF controller with 25 RBF nodes; (e) RBF controller with 50 RBF nodes; (f) RBF controller with 100 RBF nodes.

The recurrent neural network is also trained with the algorithm mentioned previously. But the result is unsatisfactory. With the same set of training samples, the error is not able to converge. This indicates that the recurrent neural network fails to learn any relevant information from the training samples. The reason is that these training samples are not sequentially relevant so that the output feedback of RNN is meaningless. Therefore, the training samples from a complete path of truck that are sequentially relevant are needed for RNN. A four output nodes RNN with all the four nodes are fed back to the input is trained in our simulation. The control surface and its error curve are shown in Figure 9. Future research will be concerned with the optimal number of nodes and feedbacks in the stand point of generalization.

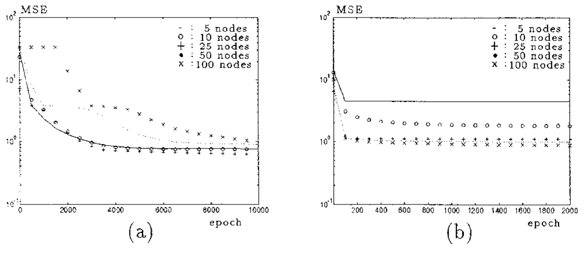
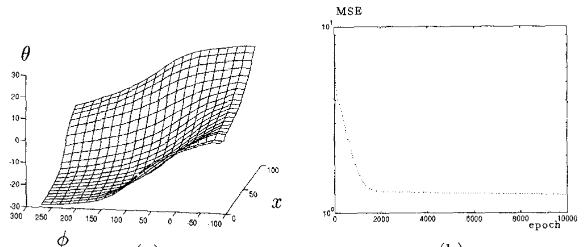


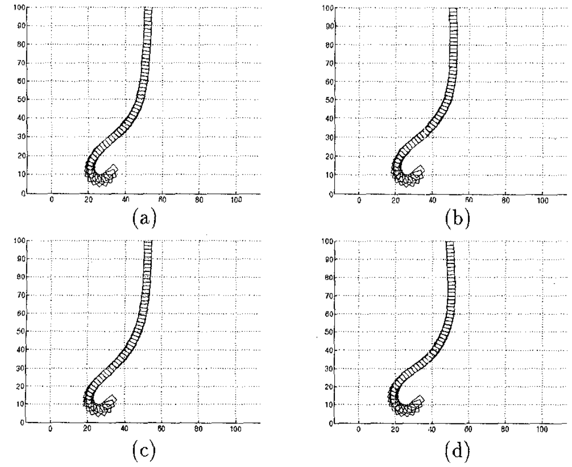
Figure 8: Error curves of (a) the multilayer feedforward network; (b) the radial-basis function network.



(a) (b)

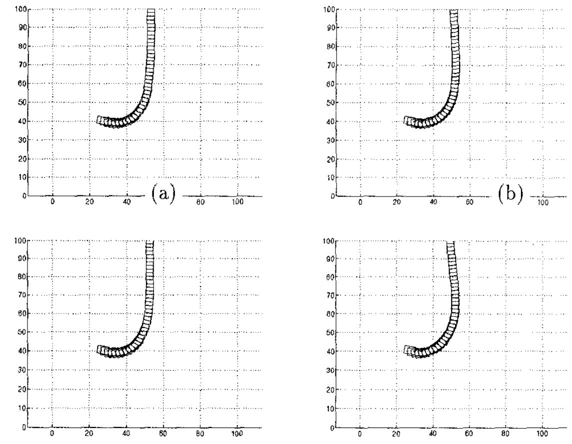
Figure 9: (a) control surface of RNN controller; (b) error curve during training.

These three different trained neural network controllers, are tested with arbitrary initial states. During the tests, the weights are fixed. The initial state is presented to the input layer. The neural controller will generalize the appropriate steering-angle according to the information it learns from the samples. The steering-angle is fed into the truck emulator to calculate the next state of the truck. And then, the next state of the truck is presented to the input layer for the appropriate steering-angle again. Two of the tests are illustrated in Figure 10 and 11. As we can see from the results, these neural network controllers are all capable of backing up the truck to the loading zone from a variety of initial positions as long as there is sufficient clearance.



(a) (b) (c) (d)

Figure 10: Truck trajectories for initial state $(30, 10, 220°)$ generated by (a) a fuzzy controller with 9 FAMs; (b) a multilayer feedforward neural controller with 10 hidden nodes; (c) a radial-basis function neural controller with 25 RBF nodes; (d) a recurrent neural controller with 4 output nodes.



(a) (b) (c) (d)

Figure 11: Truck trajectories for initial state $(30, 40, -10°)$ generated by (a) a fuzzy controller with 9 FAMs; (b) a multilayer feedforward neural controller with 10 hidden nodes; (c) a radial-basis function neural controller with 25 RBF nodes; (d) a recurrent neural controller with 4 output nodes.

5 Conclusions

In this paper we have tested the abilities of various neural networks to generalize from the training samples. For the problem we chose, the truck backer-upper control system, the results show that these neural networks are all capable of generalization. In this case, the control surface generated by the fuzzy controller is though nonlinear but not very complicated, the multilayer feedforward network is able to generalize with less hidden nodes. When the control surface gets complicated, more neurons must be needed. In contrast to the lengthy training time required for multilayer feedforward network, the radial-basis function network is an attractive alternative. RBF network offers shorter training time and even better performance with the centroids and spreading parameters chosen properly.

Though the real-time recurrent learning (RTRL) algorithm is nonlocal and suffers from a lack of stability and slow convergence speed, a very small network is sufficient for truck backer-upper control system. Such a compact network makes the training time acceptable. It is found that the desired output should be fedback for convergence while the feedback of hidden neurons can speed up the convergence and improve the generalization ability. The generalization capability of RNN is not better than BP's in our simulations. We believe this is because we only use delayed output of order one (i.e. only the latest output is fedback). With higher order delayed output, RNN is expected to perform much better. Further investigations will be conducted in the future.

References

[1] Lippmann, R.L., "An Introduction to Computing with Neural Nets," *IEEE Acoustics, Speech, Signal Proc. Mag.*, 4-22, April 1987.

[2] Moody, J., and Darken, C.J., "Fast learning in networks of locally tuned processing units," *Neural Computation,* vol. 1, 281-294, 1989.

[3] Musavi, M.T., and Ahmed, W., etc., "On the training of radial basis function classifiers," *Neural Networks,* vol. 5, 595-603, 1992.

[4] Hopfield, J.J., "Neural Networks as Physical Systems with Emergent Collective Computational Abilities," *Proceedings of the National Academy of Sciences,* 79, 2554-2558, 1982.

[5] Pineda, F.J., "Dynamics and Architecture for Neural Computation," *Journal of Complexity,* 4, 216-245, 1988.

[6] Jordan, M.I., "Attractor Dynamics and Parallelism in a Connectionist Sequential Machine," *Proceedings of the Eighth Annual Conference of the Cognitive Science Society,* 531-546, 1986.

[7] Gallant, S.I. and King, D., "Experiments with Sequential Associative Memories," *Proceeding of the Tenth Annual Conference of the Cognitive Science Society,* 40-47, 1988.

[8] Pearlmutter, B.A., "Learning State Space Trajectories in Recurrent Neural Network," *Neural Computation,* vol. 1, 1989.

[9] Williams, R.J., and Zipser, D., "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks." *Neural Computation,* vol. 1, 270-280, 1989.

[10] Nguyen, D., and Widrow, B., "The Truck Backer-Upper: An Example of Self-Learning in Neural Networks," *Proceedings of International Joint Conference on Neural Networks (IJCNN-89),* vol. II, 357-363, June 1989.

[11] Nguyen, D., and Widrow, B., "Neural networks for self-learning control systems," *IEEE Control Systems Magazine,* 18-23, April 1990.

[12] Kosko, B., *Neural Networks and Fuzzy Systems,* Prentice Hall, Englewood Cliffs, NJ, 1992.