Chapter 2

Cascade Learning: Simultaneous Perturbation with Dynamic Tunneling

2.1 Introduction

Feedforward neural networks are powerful models for solving non-linear mapping problems. Despite advances in neural networks, determining the most appropriate network size for solving a specific task is yet to be solved. Feedforward neural network model selection techniques can be classified into three groups: (i) perform a selection through arbitrary models, (ii) begin with a complex model and then simplify, and (iii) begin with a simple model then increase its complexity.

In the first method, several arbitrary structures are tried and the one giving the best performance is selected. Such neural networks could have more hidden neurons than necessary (Chen et al., 1990a; Kamarthi and Pittner, 1999; Maeda and De Figueiredo, 1997). In the second approach, a large network is trained first:
and its size is reduced by removing redundant hidden neurons using the network pruning technique (Castellano et al., 1997; Huang and Huang, 1991; Riedmiller and Braun, 1993). Castellano et al. (1997) have described an iterative pruning algorithm to remove such redundant hidden neurons. Pruning requires advanced knowledge about what size is large for the problem at hand, but this is not a serious concern when the upper bound on the number of hidden units have been established (Huang and Huang, 1991). In the third one, start with a small network and then add additional hidden units and weights until a satisfactory solution is found. This is called constructive method (Hush and Horne, 1998; Spall, 1992; Young and Downs, 1998; Zhang and Morris, 1998). The small network to start with is assumed to have no hidden units. However if prior knowledge of the problem is available, an alternative initial state may be used. Constructive algorithm searches for small network solution. Smaller networks are more efficient in feedforward computation and can be described by a simpler set of rules. If the initial network selected is too small, it may not converge to a good solution and hence underfit the data. On the other hand selecting an initial network that is much larger than required makes training computationally expensive. Constructive algorithm will spend the majority of their time training the networks smaller than the final networks as compared to algorithms that start training with an oversize network. This method has been proved to be powerful for training feedforward neural networks.

Recently several researchers have proposed different approaches to find the structure of the neural network. Holcomb and Morari (1991) have proposed a local training method for radial basis function network training and to find the
number of hidden neurons. Wang et al. (1994) have described a procedure to find
the number of hidden layers and hidden neurons in it. Hush and Horne (1998)
have presented a constructive algorithm with piecewise linear sigmoidal nodes for
non-linear function approximation. Young and Downs (1998) have described an
algorithm that constructs a feedforward network with a single hidden layer of
threshold units which implements the task for any consistent classification prob-
lem on real valued training vectors. Treadgold and Gedeon (1999) have proposed
an algorithm in which, once a new hidden neuron is connected to the network,
all weights are trained using RPROP algorithm (Riedmiller and Braun, 1993).
Simulated annealing resilient backpropagation (SARPROP) regularization term
(Riedmiller and Braun, 1993) is added to the algorithm in order to test the effect
of regularization on a constructive algorithm. Zhang and Morris (1998) have de-
scribed a sequential orthogonal training method for single hidden layer network
with sigmoidal activation function and also for the neural networks with mixed
type of hidden neurons and hybrid models.

Phansalkar and Sastry (1994) have shown that all local minima of the sum of
least squares error are stable and other equilibrium points are unstable in the back-
propagation algorithm with momentum term. Yu and Chen (1995) have shown the
condition which leads to a sufficient local minima free condition for the backprop-
agation learning. The condition is if there are $P$ nonconsistent input patterns to
learn and a two-layered feedforward neural network having $P - 1$ sigmoidal hidden
neuron and one dummy hidden neuron is used for the learning, then the subopti-
mal equilibrium point of the corresponding error surface is unstable in the sense
of Lyapunov. A method for globally searching to find the good minima has been proposed by Zhang and Chen (2000).

Training feedforward neural networks can be viewed as a nonlinear optimization problem in which the goal is to find a set of network weights that minimize an error function. The backpropagation approach is a well known approach for supervised learning in neural networks to minimize the error function. One of the major problems with backpropagation is local minimum entrapment. The dynamic tunneling technique (Barhen et al. 1997; Chowdhury et al. 1999) can be applied to detrap the local minima. Maeda and De-Figueiredo (1997) have used simultaneous perturbation to train a neuro-controller for controlling a robot arm. They have used neural network as a neuro controller to learn an inverse system of a plant without any information about the sensitivity function of the objective plant. Maeda et al. (1999) have proposed a learning rule for neural networks via simultaneous perturbation and an analog feedforward neural network circuit. The learning rule described is stochastic gradient-like algorithm via simultaneous perturbation. There is no need to do backward calculation of the backpropagation method. The basic idea of simultaneous perturbation method was proposed by Spall (1992). This is an extension of the Kiefer-Kwokowitz stochastic approximation method (spall, 1992). Spall has proved that the algorithm using that method converges to a minimum of a regression function with probability 1 under certain condition. Spall has applied this approach to various problems such as optimization, stochastic approximation, adaptive control and neural network controller and their applications (Spall, 1999). Spall has also pointed out that simultaneous per-
turbation type of methods are superior to the conventional optimization techniques. Chowdhury et al. (1999) have proposed a new method for efficient training of multilayered perceptrons by combining error backpropagation to find local minima and a dynamic tunneling technique to detrap the local minima.

In this chapter, we propose an efficient technique by combining simultaneous perturbation and dynamic tunneling technique for training single hidden layer feed-forward networks by cascade learning. A single hidden neuron is added to the single layer of the neural network after training. Then the cascaded network is trained. Simultaneous perturbation approach is used to train the network and to minimize the cost function, and the dynamic tunneling is used to detrap the local minimum which are trapped during training. The efficiency of the proposed method is demonstrated for the selected problems namely neuro-controller, encoder, adder, demultiplexer, XOR and LT character recognition problem.

2.2 Training neural network

We consider a single hidden layer neural network as shown in fig.1.1, in which $X = [x_1, x_2, ..., x_n]^T$ is the input vector. Superscript $T$ denotes transpose. $W_i = [w_{i1}, w_{i2}, ..., w_{in}]^T$ is the weight vector associated with the $i^{th}$ hidden neuron including thresholds as weights with input value 1. $W = [W_1, W_2, ..., W_k]^T$ is a weight matrix associated with the hidden layer neurons. Similarly $V = [V_1, V_2, ..., V_l]^T$ denotes the weight matrix associated with the output layer neuron. Subscript $k$ denotes number of hidden neurons and $l$ denotes number of output layer neurons. The
sigmoidal function

\[ f(x) = \frac{1}{1 + e^{-x}} \]

is used for the hidden neuron to represent its input-output characteristic. Therefore the output of the \( j^{th} \) neuron corresponding to the input is

\[ H_j = \frac{1}{1 + e^{-\sum_{k=1}^{n} w_{jk} x_k}}. \]

\([H_1, H_2, \ldots, H_k]^T\) is a vector which corresponds to the output of the hidden layer neurons.

\[ y_m = \sum_{j=1}^{k} v_{mj} H_j \]

is the system output of \( m^{th} \) output neuron for the \( i^{th} \) input pattern.

\[ y_{m, d} = \sum_{j=1}^{k} v_{mj} H_j + R_i \]

is the desired output of \( m^{th} \) output neuron for the \( i^{th} \) input pattern, where \( R_i \) is the model residual. Define \( e_i(t) \), the error function(vector) at time \( t \), as the difference between the desired output and the actual output of the network:

\[ e_i(t) = Y_{d,i}(t) - Y_i(t). \]

One of the commonly used criterion for the cost function is, sum of squared error (SSE) or mean squared error (MSE), defined as:

\[ J = \frac{1}{2} \sum_i (e_i(t))^2 \]  \hspace{1cm} (2.1)

\[ J = E[\sum_i (e_i(t))^2] \]  \hspace{1cm} (2.2)

where \( E \) is the statistical expectation operator. Here every hidden neuron is used to model the relationship between input data and model residuals. For the first hidden
neuron the model residual is the desired output as the initial network structure is assumed to be empty. The weights corresponding to the hidden neurons and the output neurons are modified by the learning rule via simultaneous perturbation technique.

2.2.1 Learning rule and simultaneous perturbation

The learning rule via simultaneous perturbation technique proposed by Maeda and De Figueiredo (1997) is used for weights updation in cascade learning. The $i^{th}$ component of the modifying quantity corresponding to the $t^{th}$ iteration, of the weights $\Delta w_i^t$ is defined as follows:

$$\Delta w_i^t = \frac{J(Y(W_t + C_t, V_t + D_t)) - J(Y(W_t, V_t))}{c_t^i} \tag{2.3}$$

where $C_t = [c_t^1, c_t^2, ..., c_t^t]^T$ and $D_t = [d_t^1, d_t^2, ..., d_t^t]^T$ are the perturbation vectors with uniformly distributed random numbers in the interval [-0.01, 0.01] except [-0.001, 0.001] for adding small disturbances to all weights, $J(Y(W_t, V_t))$ is the error function of the neural network model for the weights associated with the connections, and $J(Y(W_t + C_t, V_t + D_t))$ is the error function when all the weights of the connections in the network are disturbed, that is added, simultaneously by perturbation vectors. The properties of the perturbation vectors are assumed to be as described by Maeda and De Figueiredo (1997). Weights of the network are updated using the following learning rule,

$$w_{i+1} = w_i - \alpha \Delta w_i \tag{2.4}$$

where $\alpha$ is a positive learning coefficient. The error function is measured using
forward operation of the neural network for all input patterns. Then add small
perturbation to all weights simultaneously and observe the value of the error func-
tion using forward operation of the neural network for all input patterns. Detailed
strict convergence conditions of this simultaneous perturbation algorithm have
been described by Spall (1992).

2.2.2 Simultaneous perturbation training

2.2.2.1 Training single hidden unit

Select one hidden neuron unit as in Figure 2.1 for training. This neuron is
used to model the relationship between input value and model residual. The
weights are uniformly distributed random numbers in [-1,1]. The output of the
feedforward neural network and its error function for all training patterns are
calculated and accumulated. The random numbers for the perturbation vectors
$C_t$ and $D_t$ are generated and added correspondingly with the weights $W_t$ and
$V_t$ simultaneously. Again the output of the feedforward neural network and its
error function for all desired input patterns and the total error are computed. If
the mean squared error (MSE) of the network after perturbation is less than the
MSE of the network before perturbation then the learning rule is used to find the
modified weight and the weights are updated using the learning rules (2.3) and
(2.4) respectively. Otherwise dynamic tunneling technique is employed until the
MSE after perturbation is less than the MSE before perturbation. If the MSE of
the network after weight updation is greater than the MSE of the network before
weight updation then do the dynamic tunneling technique.
2.2.2.2 Training cascaded neural network

Initially a single hidden unit structure is trained. Then a newly created hidden unit will be cascaded with the already created neural network structure. After cascading the new hidden unit, the whole structure will be trained by the simultaneous perturbation technique. During training after perturbation if the MSE of the network is greater than the MSE of the network before perturbation then perform dynamic tunneling. Using the learning rules (2.3) and (2.4) find the modified weight and update the weights of the neural network. If the MSE of the network after updation is greater than the MSE of the network before updation then do the dynamic tunneling until the MSE of the network after updation is less than MSE of the network before updation. Selection of new hidden unit and cascading process will be continued until the training of the cascaded neural network structure results in giving small enough error.
2.2.3 Dynamic tunneling technique

The dynamic tunneling technique is an implementation of direct search method and it can be treated as a modified Hookes-Jeeves pattern search method (Deb, 1995). The concept of dynamic tunneling is based on the fact that any particle placed at small perturbation from the point of equilibrium will move away from the current point to another within a finite amount of time (Chowdhury et al., 1999). It is implemented by solving the following differential equations.

\[
\frac{dW_i}{dt} = \alpha (W_i - W)^{\frac{1}{3}} \quad (2.5)
\]

\[
\frac{dV_i}{dt} = \alpha (V_i - V)^{\frac{1}{3}} \quad (2.6)
\]

Here \( \alpha \) represents the strength of learning. The value of \( W_i \) is \( w_{ij} + \epsilon_{ij} \) for all \( i, j \); \( V_i \) is \( v_{ik} + \epsilon_{ik} \) for all \( j, k \) where \( |\epsilon_{ij}| \ll 1 \) and \( |\epsilon_{ik}| \ll 1 \) and the integration for (2.5) and (2.6) will be carried out for a fixed amount of time \( t \) with a small time step \( \Delta t \). The systems (2.5) and (2.6) are of type

\[
\frac{du}{dt} = u^{\frac{1}{3}}, \quad (2.7)
\]

which has an equilibrium point at \( u = 0 \), which violates the Lipschitz condition at \( u = 0 \) since

\[
|\frac{d}{du}(\frac{du}{dt})| = |(1/3)u^{-\frac{1}{3}}| \to \infty \quad \text{as} \quad u \to 0
\]

This equilibrium point of the system is an attracting equilibrium point since the system reaches to 0 from any given initial condition. Similarly if the right hand side function of equation (2.7) is \( u^{-\frac{1}{3}} \) then the system has unstable repelling equilibrium point, at \( u = 0 \) (Chowdhury et al., 1999). To detrap the local minima, dynamic
tunneling has been employed on the synaptic weights of the output layer and hidden layer alternatively.

2.2.4 Training algorithm

- Simultaneous perturbation

1. Generate random numbers for the synaptic weights $W$ and $V$ and for the perturbation matrices $C$ and $D$.

2. $W_1 = W + C$ and $V_1 = V + D$.

3. For each set of input pattern find the error function corresponding to the synaptic weights and the perturbed weights respectively.

4. Find $\text{diff} = \text{MSE}(W_1, V_1) - \text{MSE}(W, V)$.

5. If ($\text{diff} < 0$) then $W = W_1; V = V_1$; else do Dynamic tunneling.

6. Update the weights using equation (2.4) to obtain $W'$ and $V'$.

7. If $\text{MSE}(W', V') > \text{MSE}(W, V)$ then do Dynamic tunneling and then set $W = W'; V = V'$.

8. Train the network using steps 2 to 7 until the network is trained to the desired accuracy.

- Dynamic tunneling

Do the following for the synaptic weights $V$ and $W$ alternatively.
9. For $j = k$(number of neurons in the layer) down to 1 do
    begin
    10. Generate uniformly distributed random numbers $\epsilon_{ij}$ in the range $[-1,1]$.
    11. Find $v_{1,i} = v_{ij} + \epsilon_{ij}$ for all $i$.
    12. For $l = 1$ to $n$(where $n$ is a user defined value) do
        begin
        13. Integrate $\frac{dv_{1,i}}{dt} = \alpha(v_{1,i} - v_{i})^{\frac{1}{2}}$ for all $i$.
        14. Find $\text{diff} = \text{MSE}(W_{1}, V_{1}) - \text{MSE}(W, V)$.
        15. If $\text{diff} < 0$ then $W = W_{1}; V = V_{1}$; exit tunneling;
        end $l$; end $j$;

2.2.5 Complexity of the algorithm

In this subsection the worst case time complexity of the proposed algorithm is obtained in terms of the number of iterations performed. First we consider the training of a single hidden unit structure. In a cycle, the total number of iterations for training all the $N$ patterns will be $N$. Let $R \times \frac{\Delta t}{\delta t}$ be the maximum number of times tunneling will occur for a particular neuron. In the training process the number of output neurons will be always constant. The number of function evaluations using Runge-Kutta fourth order method is 4 per step. Let $M$ be the sum of number of outputs and one hidden neuron in the structure. Therefore the total number of iterations required during tunneling in training single hidden
unit structure is \( R \times \frac{1}{\Delta t} \times 4 \times M \). If \( K \) hidden neurons are required to reach the termination condition then \( K \) different single hidden unit structures are trained. Therefore the total number of iterations required in training all single hidden unit structures is

\[
4 \times K \times N \times M \times R \times \frac{t}{\Delta t}
\]

In cascading phase, the whole network will be trained. Cascading will occur \( K - 1 \) times. The number of iterations required for training the cascaded network will be \( N \times 4 \times Q \times S \times \frac{t}{\Delta t} \times (M + 1) \), where \( Q \) is the maximum number of times the structure is trained repeatedly after cascading and \( S \times \frac{t}{\Delta t} \) is the maximum number of times the tunneling occurs in the cascading phase. The total number of iterations required in the training of whole network will be

\[
Q \times 4 \times N \times S \times \frac{t}{\Delta t} \times (M + 1) + Q \times 4 \times N \times S \times \frac{t}{\Delta t} \times (M + 2) + \ldots + Q \times 4 \times N \times S \times \frac{t}{\Delta t} \times (M + K - 1) \\
= Q \times 2 \times N \times S \times \frac{t}{\Delta t} \times \left(K^2 + (2 \times M - 1) \times K - 2M\right)
\]

The computational effort of the proposed algorithm will be

\[
4 \times K \times N \times M \times R \times \frac{t}{\Delta t} + 2 \times Q \times N \times S \times \frac{t}{\Delta t} \times \left(k^2 + (2 \times M - 1) \times K - 2M\right).
\]

2.3 Simulation results and discussion

The performance of the proposed training scheme has been measured using the selected example problems, namely two-link planar arm, encoder, adder, demul-
tiplerex, XOR and L-T character recognition problem. In the examples two-link planar arm, encoder, adder and demultiplexer the activation function $1/(1 + e^{-x})$ has been used for hidden and output layers. The convergence of the learning process is measured by using SSE for the two-link planar arm problem. For encoder, adder and demultiplexer MSE is used. The activation function $(1 - e^{-x})/(1 + e^{-x})$ is used for hidden layer and $1/(1 + e^{-x})$ for output layer in the examples XOR and L-T character recognition. Here we use SSE. In all the examples the learning parameter $\alpha$ is fixed as 0.005. All the example problems considered were executed with five different initial weights and the results are tabulated in Table 2.1, and 2.2. Table 2.3 shows the comparison of results for encoder, adder and demultiplexer between the proposed simultaneous perturbation with dynamic tunneling (SPDT) algorithm and error backpropagation with dynamic tunneling (EBPDT) of Chowdhury et al. (1999). Comparison of results of examples XOR and L-T character recognition between the proposed algorithm and the conjugate gradient with line search and backpropagation with weight extrapolation (BPWE) methods of Kamarthi and Pittner (1999) are shown in Table 2.4.

2.3.1 Two-link planar arm

First we consider a static problem of neuro-controller for two-link planar arm. A neural network as neuro-controller can learn an inverse of a plant without any information about the sensitivity function of the objective plant. We have prepared 10 positions of the top of the arm to be learned. Top of the arm $(x, y)$ is represented
as follows using arm lengths $l_1, l_2$ and the angles $\theta_1, \theta_2$:

$$x = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2)$$

$$y = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2)$$

where $\theta_1$ and $\theta_2$ are angles as shown in Figure 2.2.

![Diagram of a two-link planar arm with labels $l_1$, $l_2$, $\theta_1$, $\theta_2$, and desired positions.](image)

Figure 2.2: Two link planar arm

We have used the above training method to find the output as $\theta_1$ and $\theta_2$ for the network input $x_1$ and $x_2$. The characteristic of each output neuron is the ordinary sigmoid function. The network has been constructed with two different data sets. It can be observed from Table 2.1 that to reach SSE of 0.048084 for the first data set, the network requires 4 neurons and 7 neurons to reach the SSE of 0.049704 for the second set of data. Maeda and De Figueiredo (1997) have used two hidden
<table>
<thead>
<tr>
<th></th>
<th># hidden neuron</th>
<th>SSE</th>
<th>Termination condition</th>
<th>Time step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two-link planar arm</td>
<td>2</td>
<td>0.048187</td>
<td>0.05</td>
<td>0.005</td>
</tr>
<tr>
<td>I data set</td>
<td>4</td>
<td>0.049983</td>
<td>0.05</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.048084</td>
<td>0.05</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.049446</td>
<td>0.05</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.049981</td>
<td>0.05</td>
<td>0.005</td>
</tr>
<tr>
<td>Two-link planar arm</td>
<td>2</td>
<td>0.046292</td>
<td>0.05</td>
<td>0.005</td>
</tr>
<tr>
<td>II data set</td>
<td>4</td>
<td>0.047966</td>
<td>0.05</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>0.049704</td>
<td>0.05</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.042975</td>
<td>0.05</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.043825</td>
<td>0.05</td>
<td>0.005</td>
</tr>
</tbody>
</table>

Table 2.1: Simulation results for two-link planar arm for different data sets

Figure 2.3: Actual points and predicted points for the first data set of the two link planer arm
Figure 2.4: Actual points and predicted points for the second data set of the two-link planar arm

layer neural networks with 10 neurons in each hidden layer for learning the 10 desired positions of the top of the arm. The actual positions of the top of the arm and the positions predicted by the proposed training algorithm and the positions predicted by Maeda and De Figueiredo (1997) algorithm are shown in Figures 2.3 and 2.4. It can be seen that the positions predicted by the proposed algorithm are almost close to the actual positions.

2.3.2 Encoder, Adder and Demultiplexer

Two types of encoders were simulated in this example. Training pattern for the first type is 0001 → 0001; 0010 → 0010; 0100 → 0100; 1000 → 1000; and for the second type is 00 → 1000; 01 → 0100; 10 → 0010; 11 → 0001.
<table>
<thead>
<tr>
<th>No. of hidden neurons</th>
<th>MSE/SSE</th>
<th>Epoch</th>
<th>Termination condition</th>
<th>Time step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoder 2-1-4</td>
<td>0.000096</td>
<td>150</td>
<td>0.0001</td>
<td>0.005</td>
</tr>
<tr>
<td>Encoder 2-1-4</td>
<td>0.000098</td>
<td>96</td>
<td>0.0001</td>
<td>0.005</td>
</tr>
<tr>
<td>Encoder 2-1-4</td>
<td>0.000094</td>
<td>131</td>
<td>0.0001</td>
<td>0.005</td>
</tr>
<tr>
<td>Encoder 2-1-4</td>
<td>0.000082</td>
<td>161</td>
<td>0.0001</td>
<td>0.005</td>
</tr>
<tr>
<td>Encoder 2-1-4</td>
<td>0.000056</td>
<td>139</td>
<td>0.0001</td>
<td>0.005</td>
</tr>
<tr>
<td>Encoder 4-1-4</td>
<td>0.000055</td>
<td>135</td>
<td>0.0001</td>
<td>0.0005</td>
</tr>
<tr>
<td>Encoder 4-1-4</td>
<td>0.000090</td>
<td>148</td>
<td>0.0001</td>
<td>0.0005</td>
</tr>
<tr>
<td>Encoder 4-1-4</td>
<td>0.000080</td>
<td>127</td>
<td>0.0001</td>
<td>0.0005</td>
</tr>
<tr>
<td>Encoder 4-1-4</td>
<td>0.000074</td>
<td>134</td>
<td>0.0001</td>
<td>0.0005</td>
</tr>
<tr>
<td>Encoder 4-1-4</td>
<td>0.000090</td>
<td>125</td>
<td>0.0001</td>
<td>0.0005</td>
</tr>
<tr>
<td>Half adder</td>
<td>0.000098</td>
<td>101</td>
<td>0.0001</td>
<td>0.05</td>
</tr>
<tr>
<td>Half adder</td>
<td>0.000098</td>
<td>109</td>
<td>0.0001</td>
<td>0.05</td>
</tr>
<tr>
<td>Half adder</td>
<td>0.000092</td>
<td>90</td>
<td>0.0001</td>
<td>0.05</td>
</tr>
<tr>
<td>Half adder</td>
<td>0.000047</td>
<td>133</td>
<td>0.0001</td>
<td>0.05</td>
</tr>
<tr>
<td>Half adder</td>
<td>0.000076</td>
<td>85</td>
<td>0.0001</td>
<td>0.05</td>
</tr>
<tr>
<td>Full adder</td>
<td>0.000092</td>
<td>79</td>
<td>0.0001</td>
<td>0.05</td>
</tr>
<tr>
<td>Full adder</td>
<td>0.000078</td>
<td>69</td>
<td>0.0001</td>
<td>0.05</td>
</tr>
<tr>
<td>Full adder</td>
<td>0.000098</td>
<td>97</td>
<td>0.0001</td>
<td>0.05</td>
</tr>
<tr>
<td>Full adder</td>
<td>0.000092</td>
<td>146</td>
<td>0.0001</td>
<td>0.05</td>
</tr>
<tr>
<td>Full adder</td>
<td>0.000092</td>
<td>141</td>
<td>0.0001</td>
<td>0.05</td>
</tr>
<tr>
<td>Demultiplexer</td>
<td>0.000093</td>
<td>226</td>
<td>0.0001</td>
<td>0.05</td>
</tr>
<tr>
<td>Demultiplexer</td>
<td>0.000086</td>
<td>194</td>
<td>0.0001</td>
<td>0.05</td>
</tr>
<tr>
<td>Demultiplexer</td>
<td>0.000094</td>
<td>148</td>
<td>0.0001</td>
<td>0.05</td>
</tr>
<tr>
<td>Demultiplexer</td>
<td>0.000067</td>
<td>230</td>
<td>0.0001</td>
<td>0.05</td>
</tr>
<tr>
<td>Demultiplexer</td>
<td>0.000084</td>
<td>173</td>
<td>0.0001</td>
<td>0.05</td>
</tr>
<tr>
<td>XOR</td>
<td>0.000080</td>
<td>226</td>
<td>0.0001</td>
<td>0.0005</td>
</tr>
<tr>
<td>XOR</td>
<td>0.000086</td>
<td>238</td>
<td>0.0001</td>
<td>0.0005</td>
</tr>
<tr>
<td>XOR</td>
<td>0.000088</td>
<td>276</td>
<td>0.0001</td>
<td>0.0005</td>
</tr>
<tr>
<td>XOR</td>
<td>0.000081</td>
<td>225</td>
<td>0.0001</td>
<td>0.0005</td>
</tr>
<tr>
<td>XOR</td>
<td>0.000054</td>
<td>184</td>
<td>0.0001</td>
<td>0.0005</td>
</tr>
<tr>
<td>L-T</td>
<td>0.000098</td>
<td>162</td>
<td>0.0001</td>
<td>0.005</td>
</tr>
<tr>
<td>L-T</td>
<td>0.000082</td>
<td>147</td>
<td>0.0001</td>
<td>0.005</td>
</tr>
<tr>
<td>L-T</td>
<td>0.000089</td>
<td>124</td>
<td>0.0001</td>
<td>0.005</td>
</tr>
<tr>
<td>L-T</td>
<td>0.000093</td>
<td>115</td>
<td>0.0001</td>
<td>0.005</td>
</tr>
<tr>
<td>L-T</td>
<td>0.000099</td>
<td>149</td>
<td>0.0001</td>
<td>0.005</td>
</tr>
</tbody>
</table>

Table 2.2: Simulation results for selected example problems
The network structure used for the first type is 4-1-4 and for the second type is 2-1-4. Chowdhury et al. (1999) have used the structure 4-2-4 and 2-1-4-4 for the first and second type of encoders. The training pattern used for half adder are 00 → 00; 01 → 01; 10 → 01; 11 → 10; and for the full adder are 000 → 00; 001 → 01; 010 → 01; 011 → 10; 100 → 01; 101 → 10; 110 → 10; 111 → 11. The network architecture used for the half adder is 2-1-2 and for the full adder is 3-1-2 where as Chowdhury et al. (1999) have used 2-2-2 and 3-3-2 respectively.

Figure 2.5: Learning curve based on MSE and epoch for demultiplexer

For demultiplexer, one input, two selection and four output lines were considered. The output line is controlled by selection lines. The input pattern ($xyz$) should be read as: $x = $ data input, $y, z =$ selection lines. If output pattern is represented by ($abcd$) then the selection lines $yz = 10$ indicate that the output $c$ will be the same as the data input $x$, while other outputs are maintained at one.
The proposed algorithm requires 3-1-4 network structure whereas 3-3-4 structure has been used in Chowdhury et al. (1999). Figure 2.5 shows the plot of epoch versus MSE of the network.

<table>
<thead>
<tr>
<th># hidden neurons</th>
<th>Encoder 2-1-4</th>
<th>Encoder 4-1-4</th>
<th>Half adder</th>
<th>Full adder</th>
<th>Demultiplexer</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBPDT</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>SPDT</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MSE</th>
<th>EBPDT</th>
<th>SPDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBPDT</td>
<td>0.001</td>
<td>0.000082</td>
</tr>
<tr>
<td>SPDT</td>
<td>0.005</td>
<td>0.000074</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Epoch</th>
<th>EBPDT</th>
<th>SPDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBPDT</td>
<td>871</td>
<td>161</td>
</tr>
<tr>
<td>SPDT</td>
<td>216</td>
<td>154</td>
</tr>
</tbody>
</table>

Table 2.3: Comparison table for the proposed method SPDT with EBPDT for the encoder, adder and demultiplexer problems

Table 2.3 shows the total number of epochs and neurons required to reach the termination condition using the proposed algorithm and the results reported by Chowdhury et al. (1999). It can be observed from the table that there is substantial improvement in number of neurons, errors and epochs.

2.3.3 XOR and L-T Character Recognition

The training pattern used for XOR are 00 → 0; 01 → 1; 10 → 1; 11 → 0. The proposed algorithm needs 2-2-1 structure for SSE of 0.000088 with 276 epochs whereas Kamarthi and Pittner (1999) have used 2-3-1 network structure for BPWE with SSE of 0.00008 and 4886 epochs and they have further shown that
the conjugate gradient with line search method is able to train the network to an error value not smaller than 0.012, that is, it could not train the network below this error for the termination condition of $10^{-4}$.

<table>
<thead>
<tr>
<th></th>
<th>Conjugate gradient with line search</th>
<th>Backpropagation with weight extrapolation</th>
<th>Proposed algorithm SPDT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XOR</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># hidden neurons</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>SSE</td>
<td>0.012</td>
<td>0.00008</td>
<td>0.000088</td>
</tr>
<tr>
<td>Total epochs</td>
<td>14</td>
<td>4886</td>
<td>276</td>
</tr>
<tr>
<td><strong>L-T</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># hidden neurons</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SSE</td>
<td>0.011</td>
<td>0.0001</td>
<td>0.000098</td>
</tr>
<tr>
<td>Total epochs</td>
<td>5</td>
<td>1811</td>
<td>162</td>
</tr>
</tbody>
</table>

Table 2.4: Comparison table for the proposed method SPDT with conjugate gradient with line search and BPWE for XOR and L-T problems.

In the letters L and T character recognition problem, the training set consisted of a 3 X 3 pixel binary image for each letter with its four orientations, in total eight patterns. The letters L and T are indicated by the target values 0.05 and 0.95 respectively for the output unit. The proposed SPDT algorithm requires 9-1-1 network structure to converge to SSE of 0.000098 using 162 epochs with a tolerance of $10^{-4}$ for termination. Kamarthi and Pittner (1999) had 9-2-1 structure to converge to 0.0001 in 1811 epochs using the BPWE method whereas they have pointed out that their structure did not converge using conjugate gradient with line search method, and was locked in a local minima after reaching the SSE of 0.011 for the termination condition $10^{-5}$. Figure 2.6 shows the decrease in the
Figure 2.6: Learning curve based on SSE and epoch of L-T character recognition error corresponding to epochs of the network. Table 2.4 shows the comparative results of the proposed algorithm with BPWE and conjugate gradient with line search methods, for the example problems XOR and L-T character recognition.

2.4 Conclusion

Simultaneous perturbation with the dynamic tunneling approach for building single hidden layer networks is proposed. Sigmoidal single hidden neurons after training are cascaded to the network in such a way that it models the relationship between input data and model residual and then the whole network is trained. Simultaneous perturbation has been used to find the local minima and to minimize the cost function. Dynamic tunneling has been employed to detrap the local minima. It has been observed that the selected examples require a smaller number
of neurons and epochs than the related results in the literature.

The proposed algorithm is very simple and needs less number of manipulations in learning. The value of the cost function, number of epochs and hidden neurons are significantly reduced in comparison to the error backpropagation with dynamic tunneling, conjugate gradient with line search and backpropagation with weight extrapolation methods. The proposed algorithm overcomes the local minimum point in learning using simultaneous perturbation procedure.

During simulation it has been noted that the number of epochs needed to converge to the desired result depends on the initial choice of the weights. The results obtained for different initial weights show the robust learning of the proposed algorithm.