Dynamic Tunneling Technique for Efficient Training of Multilayer Perceptrons
Pinaki Roy Chowdhury, Y. P. Singh, and R. A. Chansarkar

Abstract—A new efficient computational technique for training of multilayer feedforward neural networks is proposed. The proposed algorithm consists of two learning phases. The first phase is a local search which implements gradient descent, and the second phase is a direct search scheme which implements dynamic tunneling in weight space avoiding the local trap thereby generating the point of descent. The repeated application of these two phases alternately forms a new training procedure which results into a global minimum point from any arbitrary initial choice in the weight space. The simulation results are provided for five test examples to demonstrate the efficiency of the proposed method which overcomes the problem of initialization and local minimum point in multilayer perceptrons.

Index Terms—DTT, Lipschitz condition, MLP.

I. INTRODUCTION

MULTILAYER perceptrons (MLP’s) form a class of feedforward neural networks (FFNN’s). They have found a wide variety of applications in diverse areas viz. pattern recognition, classification, function approximation, prediction of currency exchange rates, maximizing the yields of chemical processes, identification of precancerous cells [1]. The most commonly used training method in MLP is error backpropagation (EBP) algorithm [2]. EBP has been tested successfully for different kinds of tasks. However it lacks in many aspects like, slow convergency, getting trapped in local minimum point [3]. A lot of progress has taken place in EBP learning theory [4]–[12]. Efforts have also been made to modify EBP training method using variable step-size [10], layer-by-layer optimization [11], and using linear block optimization by using least square techniques [12]. Second-order methods have also been studied for MLP training and they include Newton method [5], the Broyden–Fletcher–Goldfarb–Shanno method [6], the Levenberg–Marquardt modification [7], conjugate-gradient [8], and scaled conjugate-gradient [9].

In this paper a new method is developed for efficient training of MLP by combining EBP and dynamic tunneling technique [13]. The EBP is used here to find a local minimum point and the dynamic tunneling technique (DTT) is employed to detrap the local minimum. Thus the application of DTT results in the point of next descent. This technique along with EBP applied alternately in the weight space results in the point of global minimum, as demonstrated using various case studies. In all these experiments various initial conditions in the weight space were selected to demonstrate its efficacy to get global minimum point overcoming the problem of initialization in MLP.

In this work Section II will discuss the architecture of MLP. Proposed learning technique is discussed in Section III. Section IV gives the computational steps along with the complexity of the algorithm, whereas Section V deals with the simulation results and discussions, confirming good performance of the proposed learning methodology. To end, Section VI gives the conclusions.

II. MLP ARCHITECTURE

The general architecture of MLP is shown in Fig. 1. It has an input layer, arbitrary number of hidden layers, and an output layer. Input is fed to each of the input layer (layer 1) neurons, the outputs of input layer feed into each of the layer 2 neurons, and so on as shown in Fig. 1. In this work the layered structure having 1 input, 1 output, and \(L-2\) hidden layers will be termed as a \(L\) layer network. The model of a typical neuron in MLP shown in Fig. 2. The output from a neuron can be expressed...
mathematically as

\[ f(y) = (1 + e^{-\beta y})^{-1} \]  (1)

where \( \beta \) indicates the gain of the sigmoid and \( y \) is the sum of the weighted response from the neurons in the preceding layer. In Fig. 2, \( s \) indicates sigmoid nonlinearity.

### III. Training Method

This section describes the proposed learning method as weight updation scheme. The computational scheme of the proposed technique starts as follows.

An initial point is chosen in the weight space at random and then it is slightly perturbed. The new point is tested for either gradient descent or tunneling phase in the following manner.

Let the random point chosen on the weight space be denoted by \( \mathbf{W} \), where \( \mathbf{W} \in \mathbb{R}^{n} \) where \( n \) denotes the dimension of the search space. The new point is represented by \( \mathbf{W} + \varepsilon \), where \( \varepsilon \) has the same dimension as \( \mathbf{W} \), and each component of \( \varepsilon \) is a small quantity (\( \ll 1 \)). Depending on the relative value of mean squared error (mse), which is the mean of the squared error given by (2) below, at \( \mathbf{W} \) and \( \mathbf{W} + \varepsilon \); i.e., if \( \text{mse}_{\mathbf{W}} \leq \text{mse}_{\mathbf{W} + \varepsilon} \) then learning in gradient descent is initiated, else tunneling takes place in the weight space. If a gradient descent phase is initiated then it will converge to local minimum point. If tunneling is initiated then it finds a point for the gradient descent. After this, the algorithm automatically enters either of the two phases alternately and weights are updated according to the modification rule of the respective phases explained later. The learning procedure continues till the global optimal weights are obtained with minimum mse.

#### A. Phase 1—EBP

The most popular learning algorithm in MLP is the EBP and is described in brief with the following notations.

- \( u_{j}^{l} \) Output of the \( j \)th node in layer \( l \).
- \( w_{j,k}^{l} \) Weight connecting \( j \)th node in layer \( l \) to \( k \)th node in layer \( l - 1 \).
- \( x_{p} \) \( p \)th training sample.
- \( d_{j}(x_{p}) \) Desired response of the \( j \)th output node for the \( p \)th training sample.
- \( N^{l} \) Number of nodes in layer \( l \).
- \( L \) Number of layers.
- \( P \) Number of training patterns.

In the above notations \( u_{0}^{l} = 1 \) and \( w_{j,0}^{l} \) represents the bias weights, where \( l \neq 1 \).

EBP implements a gradient search technique to find the network weights, that minimizes the squared error function given below

\[ E(W) = 1/2^{L} \left( \sum_{l=1}^{L} \sum_{q=1}^{N^{l}} (u_{q}^{l}(x_{p}) - d_{q}(x_{p}))^{2} \right). \]  (2)

The weights of the network are updated iteratively according to

\[ u_{j,k}^{l}(t+1) = u_{j,k}^{l}(t) - \eta \frac{\partial E(W)}{\partial u_{j,k}^{l}} \]  (3)

where \( \eta \) is a positive constant, called the learning rate, and \( t \) represents the index of iteration. Thus, application of gradient descent in weight space results in

\[ \frac{du_{j,k}^{l}}{dt} = -\eta \frac{\partial E(W)}{\partial u_{j,k}^{l}}. \]  (4)

The dynamics of the error function is then given by

\[ \frac{dE(W)}{dt} = -\eta \sum_{l=1}^{L} \sum_{j=1}^{M^{l}} (\frac{\partial E(W)}{\partial u_{j,k}^{l}})^{2}. \]  (5)

Thus for a nonnegative \( \eta \) the total error \( E(W) \) will be a nonincreasing function of time. This proves the fact that there will be a global decrease in the value of \( E(W) \) with time.

#### B. Phase II—The Dynamic Tunneling Technique

The dynamic tunneling technique is an implementation of direct search method and it can be treated as a modified Hooke–Jeeves pattern search method [15], which is discussed below. An equilibrium point \( u_{eq} \) of the dynamical system

\[ \frac{du}{dt} = g(u) \]  (6)

is termed an attractor (repeller) if no (at least one) eigenvalue of the matrix \( A \)

\[ A = \frac{\partial g(u_{eq})}{\partial u} \]  (7)

has a positive real part [13]. Typically, dynamical system such as (6) obey Lipschitz condition

\[ |\partial g(u_{eq})/\partial u| < \infty \]  (8)

which guarantees the existence of a unique solution for each initial condition \( u_{0} \). Usually such systems has infinite relaxation time to an attractor and escape time from a repeller. Based on the violation of Lipschitz condition at equilibrium points, which induces singular solutions such that each solution approaches an attractor, or escapes from a repeller in finite time. To exemplify the above statement, consider a system given by

\[ \frac{du}{dt} = -u^{1/3}. \]  (9)

The system represented by (9) has an equilibrium point at \( u = 0 \), which violates the Lipschitz condition at \( u = 0 \), since

\[ |d/du(du/dt)| = |1/3u^{-2/3}| \rightarrow \infty, \quad \text{as } u \rightarrow 0. \]  (10)

The equilibrium point of the above mentioned system is termed as attracting equilibrium point, since from any initial condition \( u_{0} \neq 0 \), the dynamical system in (9) reaches the equilibrium point \( u = 0 \) in a finite time \( t_{1} \) given by

\[ t_{1} = -\int u^{-1/3} du = 3/2u_{0}^{2/3}. \]  (11)

Similarly, the dynamical system

\[ \frac{du}{dt} = u^{-1/3} \]  (12)

has a repelling unstable equilibrium point at \( u = 0 \) which violates the Lipschitz condition. Any initial condition which
is infinitesimally close to the repelling point \( u = 0 \) will escape the repeller, to reach point \( u_0 \) in a finite time given by

\[
t_1 = \int u^{-1/3} \, du = 3/2 \cdot 6^{2/3}.
\]

(13)

The concept of dynamic tunneling algorithm is based on the violation of Lipschitz condition at equilibrium point, which is governed by 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 as discussed above. The tunneling is implemented by solving the differential equation given below

\[
du_{j,k}^{t}/dt = \rho(u_{j,k}^{t} - u_{j,k}^{*})^{1/3}.
\]

(14)

Here \( \rho \) represents the strength of learning, \( u_{j,k}^{*} \) represents the last local minimum for \( u_{j,k}^{t} \). It is obvious from (6) that the local minimum point \( W^{*} \) is also the point of equilibrium of the tunneling system. The value of \( u_{j,k}^{t} \) is \( u_{j,k}^{*} + \varepsilon_{j,k}^{t} \), where \( |\varepsilon_{j,k}^{t}| \leq 1 \) and (14) is integrated for a fixed amount of time \( (\Delta t) \), with a small time-step \( (\Delta t) \). After every time-step, \( \text{mse}_W \) is computed with the new value of \( u_{j,k}^{*} \) keeping remaining components of \( W \) same as \( u_{j,k}^{*} \). Tunneling comes to a halt when \( \text{mse}_W \leq \text{mse}_W^m \) where \( \text{mse}_W^m \) indicates last local minimum (the condition for descent), and initiates the next gradient descent. If this condition of descent is not satisfied, then this process is repeated with all the components of \( u_{j,k}^{*} \) until the above condition of descent is reached. If for no value of \( u_{j,k}^{*} \) the above condition is satisfied, then the last local minimum is the global minimum point. In this way by repeated application of gradient descent and tunneling in weight space, may lead to the point of global minimum. In this work, to avoid the condition of saturation of neurons, a corrective term [16] is introduced as

\[
f'(\alpha) = f'(\alpha) + 0.1.
\]

(15)

Finally, a general expression for the dynamical system discussed separately in two phases can be written as

\[
du_{j,k}^{t}/dt = -\eta \partial E(W)/\partial u_{j,k}^{t} \Theta[1 - \Theta[\text{diff}]] + \rho(u_{j,k}^{t} - u_{j,k}^{*})^{1/3} \Theta[\text{diff}]
\]

(16)

here \( \Theta \) is a heaviside step function defined as

\[
\Theta[x] = \begin{cases} 
0 & x \leq 0 \\
1 & x > 0
\end{cases}
\]

and diff is defined as \( \text{mse}(W) - \text{mse}(W^{*}) \). The implementation issues of (16) is as discussed earlier.

IV. COMPUTATIONAL SCHEME

In this section the proposed learning algorithm is expressed as computational steps as given below.

A. Algorithm

1) Initialize \( W^{*} \) — any small random weights.
2) \( \varepsilon \leftrightarrow \text{small value } \ll 1 \) [this means, \( \varepsilon \) is of same dimension as \( W^{*} \) and each component of \( \varepsilon \ll 1 \)].
3) \( W \leftarrow W^{*} + \varepsilon \).
4) Compute \( \text{diff} \leftarrow \text{mse}(W) - \text{mse}(W^{*}) \).
5) If \( \text{diff} \leq 0 \) then EBP; else Dynamic Tunneling
6) EBP

6.1) Choose the tolerance limit (tol) at which gradient descent will come to a halt.

6.2) Repeat

6.2.1) Feed Forward
6.2.2) Compute Gradient
6.2.3) Update Weight

6.3) Until termination condition is satisfied.

Now Steps 6.2.1–6.2.3 will be explained as computational models [17] in brief.

6.2.1.1) \( k \leftarrow 2; \)
6.2.1.2) \( \forall j \mid j \leq N^k \rightarrow (u_{j}^{t} \leftarrow f(\sum_{m=0}^{N^k} u_{m}^{k-1} w_{j,m}) od2 od1) \).
6.2.1.3) \( k \leftarrow L; \)
6.2.2.1) \( \forall j \mid j \leq N^k \rightarrow (u_{j}^{t} \leftarrow \eta \cdot \frac{\partial E(W)}{\partial u_{j}^{t}} + \frac{\partial E(W)}{\partial u_{j}^{t}} \cdot (1 - \eta \cdot \frac{\partial E(W)}{\partial u_{j}^{t}}) od2 od1) \).
6.2.2.2) \( \forall j \mid j \leq N^k \rightarrow (u_{j}^{t} \leftarrow \eta \cdot \frac{\partial E(W)}{\partial u_{j}^{t}} \cdot (1 - \eta \cdot \frac{\partial E(W)}{\partial u_{j}^{t}}) od2 od1) \).
6.2.3.1) \( k \leftarrow 1; \)
6.2.3.2) \( \forall j \mid j \leq N^k \rightarrow (u_{j}^{t} \leftarrow \eta \cdot \frac{\partial E(W)}{\partial u_{j}^{t}} \cdot (1 - \eta \cdot \frac{\partial E(W)}{\partial u_{j}^{t}}) od2 od1), \) here \( \eta \) represents the learning rate.

B. Complexity of the Algorithm

In this section the worst case complexity of the proposed method will be analyzed in terms of the number of iterations performed both in EBP and DTT. To proceed with the analysis,
Table I

<table>
<thead>
<tr>
<th>Problem</th>
<th>NOE</th>
<th>mse</th>
<th>tol</th>
<th>steps</th>
<th>η</th>
<th>Δt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a)</td>
<td>1061</td>
<td>0.0006</td>
<td>10^{-3}</td>
<td>10^{-13}</td>
<td>0.5</td>
<td>5*10^{-5}</td>
</tr>
<tr>
<td>b)</td>
<td>1031</td>
<td>0.0008</td>
<td>10^{-3}</td>
<td>10^{-11}</td>
<td>0.5</td>
<td>5*10^{-5}</td>
</tr>
<tr>
<td>c)</td>
<td>516</td>
<td>0.0005</td>
<td>10^{-3}</td>
<td>10^{-11}</td>
<td>0.5</td>
<td>5*10^{-5}</td>
</tr>
<tr>
<td>d)</td>
<td>888</td>
<td>0.0006</td>
<td>10^{-3}</td>
<td>10^{-11}</td>
<td>0.5</td>
<td>5*10^{-5}</td>
</tr>
<tr>
<td>e)</td>
<td>2017</td>
<td>0.0006</td>
<td>10^{-3}</td>
<td>10^{-11}</td>
<td>0.5</td>
<td>5*10^{-5}</td>
</tr>
<tr>
<td>4-2-4 Encoder</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a)</td>
<td>146</td>
<td>0.006</td>
<td>10^{-4}</td>
<td>10^{-25}</td>
<td>0.5</td>
<td>5*10^{-5}</td>
</tr>
<tr>
<td>b)</td>
<td>186</td>
<td>0.005</td>
<td>10^{-4}</td>
<td>10^{-25}</td>
<td>0.5</td>
<td>5*10^{-5}</td>
</tr>
<tr>
<td>c)</td>
<td>216</td>
<td>0.005</td>
<td>10^{-4}</td>
<td>10^{-25}</td>
<td>0.5</td>
<td>5*10^{-5}</td>
</tr>
<tr>
<td>2-1-4-4 Encoder</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a)</td>
<td>46</td>
<td>0.001</td>
<td>10^{-3}</td>
<td>10^{-22}</td>
<td>0.5</td>
<td>5*10^{-4}</td>
</tr>
<tr>
<td>b)</td>
<td>871</td>
<td>0.001</td>
<td>10^{-3}</td>
<td>10^{-22}</td>
<td>0.5</td>
<td>5*10^{-4}</td>
</tr>
<tr>
<td>Full Adder</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a)</td>
<td>371</td>
<td>0.0015</td>
<td>10^{-4}</td>
<td>10^{-30}</td>
<td>0.5</td>
<td>5*10^{-4}</td>
</tr>
<tr>
<td>b)</td>
<td>311</td>
<td>0.0020</td>
<td>10^{-4}</td>
<td>10^{-30}</td>
<td>0.5</td>
<td>5*10^{-4}</td>
</tr>
<tr>
<td>c)</td>
<td>396</td>
<td>0.0017</td>
<td>10^{-4}</td>
<td>10^{-30}</td>
<td>0.5</td>
<td>5*10^{-4}</td>
</tr>
</tbody>
</table>

EBP is considered first. Symbols used carry the interpretation as stated earlier.

Total number of iterations (TNOI) for one pattern

in 6.2.1 = \sum_{j=2,\ldots,L} N^j. \quad (17)

Similar results can be computed for 6.2.2) and 6.2.3). For \( P \) training patterns the total number of iterations in one cycle of EBP is given as below

\[ P^* \left( \sum_{j=2,\ldots,L} (N^j * (2N^{j-1} + 3)) \right). \quad (18) \]

Assuming number of cycles required using all the training patterns for EBP to converge to a local minimum as \( C_1 \), then total number of iterations is given by TEBP and can be expressed as

\[ TEBP \approx O(P x C_1 x N^j x N^{j-1}), \quad 2 \leq j \leq L. \quad (19) \]

The number of times tunneling will occur, for a particular variable = \( t/\Delta t \).

\[ (20) \]

The number of variables (\( v \))

\[ = \sum_{j=2,\ldots,L} (N^j + (N^{j-1} + 1)) \quad (21) \]

if each time tunneling occurs, the condition of descent is satisfied in the last phase of tunneling, and in the last variable, then the total number of iterations required during tunneling is given by

\[ f_e = vt/\Delta t. \quad (22) \]

During the process of integration by RK45 method, the number of function evaluations can be given approximately by

\[ f_e^{RK45} = (1 + 2 + 3 + \cdots + m)\Delta t \quad (23) \]

where \( m = t/\Delta t \).

The \( f_e^{RK45} \) can be easily reduced to

\[ f_e^{RK45} = t(t + \Delta t)/2\Delta t \approx (t^2/\Delta t). \quad (25) \]
Total number of iterations during the tunneling phase
\[ \approx v(\Delta t^3/\Delta t^2), \]  
(26)
So the total number of function evaluations during one phase of EBP and tunneling phase is given by \( T_{fe} \) as
\[ T_{fe} \approx O(N^3 \times N^{j-1} \times t^3) + O(P \times C1 \times N^j \times N^{j-1}), \]  
(27)
If number of such cycles is \( C_2 \), then the worst case analysis of our proposed method can be represented by \( T_{fe} \) given by
\[ T_{fe} \approx v \times C_2(\Delta t^3/\Delta t^2) + C_2 \times TEBP \]
which can be expressed as
\[ T_{fe} \approx C_2[O(N^j \times N^{j-1} \times t^3) \]
\[ + O(P \times C1 \times N^j \times N^{j-1})]. \]
(28)
So, the expected computational effort in our proposed method is cubic in the problem variables \( t \), squared in problem variable \( N^j \).

V. SIMULATION RESULTS AND DISCUSSIONS

To assess the performance of the proposed training scheme, experiments were conducted on standard problems of 1) parity; 2) encoder 3) adders 4) demultiplexer; and 5) character recognition. The setup in terms of number of layers and nodes for the above mentioned problems are discussed below.

A. Parity

The Parity problem is a standard problem where the output of the network is required to be “1” if the input pattern contains an odd number of “1’s” and “0” otherwise. In this problem the most similar patterns which differ by a single bit require different answer. The XOR problem [18] is a parity problem with input pattern of size 2. For this problem a three-layer network of size 4-4-1 is considered. Five different initial conditions are considered in the weight space for training the network and the result is provided in Table I. The results confirm the robustness of the algorithm in overcoming the problem of initialization.

B. Encoder

In this example two types of encoders are considered. In the first case a set of orthogonal input patterns are mapped to a set of orthogonal output patterns [19]. For this work a three-layer network of size 4-2-4 was employed. The next problem taken up in this section was that of a four-layer network of dimension 2-1-4-4. For the first case the network was trained starting from three different initial conditions, and whereas in the second case it was trained using two different initial conditions. The results of these are presented in Table I. From Fig. 3 it can be concluded that there is a continuous decrease of error for each epoch. From the performance of the network it can be concluded that the training process is not dependent on the initial choice of the weight. The number of epochs taken to arrive at the global minimum are comparatively less than reported in [19]. The training pattern for the first case is
\[ 0001 \rightarrow 0001; \quad 0010 \rightarrow 0010; \]
\[ 0100 \rightarrow 0100; \quad 1000 \rightarrow 1000. \]
Whereas the coding scheme for the second case is
\[ 00 \rightarrow 1000; \quad 01 \rightarrow 0100; \quad 10 \rightarrow 0010; \quad 11 \rightarrow 0001. \]

C. Adders

In this example both full-adder and half-adder is investigated. A full-adder has three inputs and two outputs. For this case study a 3 layer network of size 3-3-2 was used. The next
TABLE II

<table>
<thead>
<tr>
<th></th>
<th>NOE</th>
<th>mse</th>
<th>tol</th>
<th>steps</th>
<th>(\eta)</th>
<th>(\Delta t)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Half Adder</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a)</td>
<td>406</td>
<td>0.0035</td>
<td>(10^4)</td>
<td>(10^{-5})</td>
<td>0.5</td>
<td>(5 \times 10^5)</td>
</tr>
<tr>
<td>(b)</td>
<td>221</td>
<td>0.0035</td>
<td>(10^4)</td>
<td>(10^{-5})</td>
<td>0.5</td>
<td>(5 \times 10^5)</td>
</tr>
<tr>
<td><strong>Demultiplexer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a)</td>
<td>181</td>
<td>0.003</td>
<td>(10^4)</td>
<td>(10^{-5})</td>
<td>0.5</td>
<td>(5 \times 10^5)</td>
</tr>
<tr>
<td>b)</td>
<td>131</td>
<td>0.003</td>
<td>(10^4)</td>
<td>(10^{-5})</td>
<td>0.5</td>
<td>(5 \times 10^5)</td>
</tr>
<tr>
<td>c)</td>
<td>141</td>
<td>0.003</td>
<td>(10^4)</td>
<td>(10^{-5})</td>
<td>0.5</td>
<td>(5 \times 10^5)</td>
</tr>
<tr>
<td>d)</td>
<td>136</td>
<td>0.003</td>
<td>(10^4)</td>
<td>(10^{-5})</td>
<td>0.5</td>
<td>(5 \times 10^5)</td>
</tr>
<tr>
<td>e)</td>
<td>296</td>
<td>0.003</td>
<td>(10^4)</td>
<td>(10^{-5})</td>
<td>0.5</td>
<td>(5 \times 10^5)</td>
</tr>
<tr>
<td><strong>Character Recognition</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a)</td>
<td>691</td>
<td>0.0557</td>
<td>(10^4)</td>
<td>(10^{-5})</td>
<td>0.1</td>
<td>(5 \times 10^5)</td>
</tr>
<tr>
<td>b)</td>
<td>1151</td>
<td>0.0620</td>
<td>(10^4)</td>
<td>(10^{-5})</td>
<td>0.07</td>
<td>(5 \times 10^5)</td>
</tr>
<tr>
<td>c)</td>
<td>566</td>
<td>0.0452</td>
<td>(10^4)</td>
<td>(10^{-5})</td>
<td>0.1</td>
<td>(5 \times 10^5)</td>
</tr>
<tr>
<td>d)</td>
<td>1426</td>
<td>0.0501</td>
<td>(10^4)</td>
<td>(10^{-5})</td>
<td>0.1</td>
<td>(5 \times 10^5)</td>
</tr>
<tr>
<td>e)</td>
<td>1146</td>
<td>0.0544</td>
<td>(10^4)</td>
<td>(10^{-5})</td>
<td>0.1</td>
<td>(5 \times 10^5)</td>
</tr>
<tr>
<td>f)</td>
<td>921</td>
<td>0.0423</td>
<td>(10^4)</td>
<td>(10^{-5})</td>
<td>0.1</td>
<td>(5 \times 10^5)</td>
</tr>
</tbody>
</table>

problem in the adders section is half-adder, which has two input bits and two output bits. A three-layer network of size 2-2-2 was used for this problem. For full-adder the network was trained starting from three different initial conditions and whereas in half-adder two different initial conditions were taken. The result is again presented in Tables I and II, respectively, indicating the robustness of the algorithm toward initial choice of weight. The training patterns used for full-adder are

\[
\begin{align*}
000 & \rightarrow 00; & 001 & \rightarrow 01; & 010 & \rightarrow 01; & 011 & \rightarrow 10; \\
100 & \rightarrow 01; & 101 & \rightarrow 10; & 110 & \rightarrow 10; & 111 & \rightarrow 11.
\end{align*}
\]

The training patterns for half-adder are

\[
\begin{align*}
00 & \rightarrow 00; & 01 & \rightarrow 01; & 10 & \rightarrow 01; & 11 & \rightarrow 10.
\end{align*}
\]

D. Demultiplexer

A Demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines. The selection of a specific output line is controlled by the bit value of \(n\) selection lines. For this particular case study one input, two selection, and four output lines were considered. For training purpose a three-layer network of size 3-3-4 was used. Again five different initial conditions were taken for training the network. The result as presented in Table II confirms the efficacy of the algorithm to overcome the judicious selection of initial weights. Also from Fig. 4 it can be observed that the error decreases continuously with number of epochs, thereby indicating the trend of global decrease. The input pattern \((xyz)\) in Table I should be read as: \(x\) = data input, \(y\), \(z\) = selection lines. If output pattern is represented by \((abde)\) then, the selection lines \(yz = 10\), indicates that the output \(c\) will be same as the data input \(x\), while other outputs are maintained at one.

E. Character Recognition

In this problem the training set is constituted by the ten digits and 26 letters (first 13 lower and upper case) of the English alphabet. Each of them is represented by a matrix
of size $7 \times 5$ black and white pixels. Each character has a specified target given by a seven bit ASCII code. Character I has an ASCII code of $1001001$ and is represented as

00000
01110
00100
00100
00100
01110
00000.

The network used for training has 35 input nodes, ten hidden nodes, and seven nodes in the output layer. The results presented in Table II along with Fig. 5 confirms the accuracy and robustness of the algorithm.

Finally the comparison of the proposed method along with EBP presented in Table III underlines the fact that the proposed method can attain a lower value of MSE in less number of epochs. Table III along with Figs. 3–5 establishes the fact that dynamic tunneling ensures the traversal of only those state spaces which should guarantee a continuous decrease in the value of objective function thereby eliminating the problem of initialization in MLP.
VI. CONCLUSIONS

In this paper a new algorithm based on combining the gradient descent method and dynamic tunneling technique for training the MLP is proposed. The new algorithm is tested on many test cases and the results obtained confirms the robust learning using the proposed algorithm and overcomes the problem of initialization and local minimum point in learning using EBP algorithm. This algorithm usually leads to the point of global minimum, keeping in view the termination condition of the gradient descent phase. The algorithm reaches the global minimum in polynomial time. This algorithm is sensitive to the initial choice of weight only from the perspective of number of epochs. A good initial point will converge to the global minimum very rapidly whereas a poor choice of the starting condition takes more time. This algorithm is computationally efficient as it requires evaluation of the first-order derivative only. However, second-order methods can also be investigated along with dynamic tunneling technique.

REFERENCES


Pinaki RoyChowdhury, photograph and biography not available at the time of publication.

Y. P. Singh, photograph and biography not available at the time of publication.

R. A. Chansarkar, photograph and biography not available at the time of publication.