A Novel Fast Backpropagation Learning Algorithm Using Parallel Tangent and Heuristic Line Search

PAYMAN MOALLEMI\textsuperscript{I} \hspace{2cm} S. AMIRHASSAN MONADJEMI\textsuperscript{II} \hspace{2cm} MOHSEN ASHOURIAN\textsuperscript{III}

\textsuperscript{I} Department of Electronic Engineering, Faculty of Engineering
\textsuperscript{II} Department of Computer Engineering, Faculty of Engineering
\textsuperscript{III} Department of Information Technology Engineering, Faculty of Engineering
University of Isfahan
Hezarjarib Avenue, Isfahan 81746
IRAN

Abstract: In gradient based learning algorithms, the momentum has usually an improving effect in convergence rate and decreasing the zigzagging phenomena. However it sometimes causes the convergence rate to decrease. The Parallel Tangent (ParTan) gradient is used as deflecting method to improve the convergence. From the implementation point of view, it is as simple as the momentum. In fact this method is one of the more practical implementation of conjugate gradient. ParTan tries to overcome the inefficiency of zigzagging of conventional backpropagation by deflecting the gradient through acceleration phase. In this paper, we use two learning rate, $\eta$ for gradient search direction and $\mu$ for accelerating direction through parallel tangent. Moreover, an improved version of dynamic self adaptation of $\eta$ and $\mu$ is used to improve parallel tangent gradient learning method. In dynamic self adaptation, each learning rate is adapted locally to the cost function landscape and the previous learning rate. Finally we test the proposed algorithm on various MLP neural networks including a XOR $2 \times 2 \times 1$, Encoder $16 \times 4 \times 16$ and finally Parity $4 \times 4 \times 1$. We compare the results with those of the dynamic self adaptation of gradient learning rate and momentum (DS $\eta$-$\alpha$) and parallel tangent with dynamic self adaptation (PTDS $\eta$-$\mu$). Experimental results showed that the average number of epoch is decreased to around 66% and 50% for DS $\eta$-$\alpha$ and PTDS $\eta$-$\mu$ respectively. Moreover our proposed algorithm shows a good power for avoiding from local minimum.

Key-Words: Parallel Tangent, Zigzagging, Adaptive Learning Rates, Backpropagation.

1 Introduction

Learning is one the most attractive properties of neural networks [1]. While the weight tuning of a multi layer perceptron (MLP) was a problem recently, using steepest descent method based on backpropagation learning algorithm [2], made a great progress for MLP neural networks by giving the ability to have more than one changing coefficient layer. The central idea behind this solution is that the errors for the units of the hidden layer are determined by back propagating the errors of the units of the output layer. Thus the method is often called the backpropagation learning rule.

Considering steepest descent method for MLP learning, if $W_k$ and $E(W_k)$ are the weights vector and output error of network in $k^{th}$ epoch, the weight vector can be corrected as:

$$W_{k+1} = W_k - \eta \nabla E(W_k)$$ (1)

where $\nabla E(W_k)$ is gradient of error (calculated weight change) and $\eta$ is the gradient learning rate. Generally, this procedure has problem-dependent disadvantages. Firstly, convergence is fast only if the parameter setting is optimal; therefore learning rate should be adaptively computed. Secondly, the gradient algorithm usually behaves poorly near an optimum point where zigzagging phenomena occurs [3].

1.1 Adaptive Learning Rate

Fixed learning rate results to unstable convergence near the optimum point, therefore convergence is guaranteed only if the learning rate is small enough. On the other hand, small learning rate results to slow convergence and therefore a long learning procedure. So many researches have suggested variable learning rate.
In fact, the learning rate determines what portion of calculated weight change will be used for correction. The optimal value of the learning rate depends on the characteristics of the error surface. If the surface changes rapidly, the gradient calculated just on the local information will be poor indication of the true right path. Therefore a smaller rate is desirable. On the other hand, if the surface is relatively smooth, then a large rate will converge rapidly [3].

Line search [3] is a popular method for calculating the optimal learning rate for each iteration step. In [4] a quadratic estimation of error surface was used to estimate a suitable rate in each epoch. This method could effectively reduce the number of epochs but using of the line search, the computation time of each epoch is larger than the typical one.

The dynamic self-adaptation algorithm [5] adapts itself to the local information based on the cost function \( E(W_k) \). It is a two-step procedure that first suggests two values for the step-size corresponding to the previous value and then selected the best. Assume that in the epoch \( k \), \( E(W_k) \) is the cost function of weights, \( \nabla E(W_k) \) is the search direction and \( \eta_{k+1} \) is the proposed learning rate then the new value of weight vector is computed as:

\[
\eta_{k+1} = \begin{cases} 
\eta_k \xi & \text{if } E(W_k - \eta_k \nabla E_k / \xi) < E(W_k - \eta_k \nabla E_k / \xi) \\
\eta_k / \xi & \text{else}
\end{cases}
\]

That is, out of two possibility; \( \eta \xi \) and \( \eta / \xi \); the algorithm takes the one that gives the lower value for the cost function, i.e., the one that is the best. So at each epoch, the procedure adapts itself in a simple but straightforward way to the cost function landscape. Although in principle \( \xi \) can be any number larger than 1.0, it was shown that the choice of \( \xi = 1.839 \) is nearly optimum for elliptical contours [5].

1.2 Zigzagging Phenomenon

The gradient algorithm usually behaves poorly near an optimum point where small orthogonal steps are taken, this is called zigzagging phenomenon. To illustrate the zigzagging phenomena, let us consider an objective function with concentric ellipsoidal contours as shown in the Figure 1 with minimum point \( P^* \). If the initial point for a gradient search happens to be precisely on one of the axes of the systems of ellipses, the gradient line will pass right through the optimum (peak) and the search will be over in one descent (ascent). Otherwise, the search will follow a zigzag course such as the one from P1 to P2 to P3 and so on. Finally, the zigzagging course will be near to \( P^* \) as shown in the Figure 1.

![Fig 1. Zigzagging phenomenon in ellipsoidal contours](image)

One approach for deflecting the gradient step and reducing zigzagging phenomena is the momentum direction method. It is a kind of memory that incorporates the weight change of previous step and in this way slows down the useless oscillations. More precisely, we have \( W_{k+1} = W_k + \Delta W_k \) such that [3]:

\[
\Delta W_{k+1} = -\eta \nabla E(W_k) + \alpha \Delta W_k \quad (3)
\]

\( \eta \) and \( \alpha \) are the learning rate in direction of gradient and momentum. In some problem momentum shows good performance but unfortunately sometimes it has some disadvantages and so decreases the convergence rate and increase failures. Parallel Tangent gradient, improve the speed of training and failure rate, and from point of complexity, it is comparable to the momentum method.

In the next section, we briefly introduce the parallel tangent method. Then we develop a fast learning algorithm for MLP neural network.

2 Parallel Tangent Gradients

The parallel tangent technique combines many desirable properties of the simple gradient method [6]. It makes the use of the geometric property of quadratic objective function and it works well with elliptical contours. Parallel tangent has many forms and gradient ParTan is one form which amounts to a multidimensional extension of the accelerated gradient method. This technique represents a distinct improvement over the method of steepest descent. In the Figure 1, it can be seen that the crooked path is bounded by two straight lines which intersect about at the optimum \( P^* \). This suggests that the search from point P3 be conducted, not in the gradient direction toward P4, but along the straight line from P1 through P3, as shown in the
Figure 2. In this way, the peak \( P^* \) would be located after three steps, first from \( P_1 \) to \( P_2 \) along the gradient at \( P_1 \), then from \( P_2 \) to \( P_3 \) along the gradient at \( P_2 \), and finally from \( P_3 \) along the line through \( P_1 \) to \( P_3 \). This is the two dimensional version of a method which accelerates along a ridge and usually is called parallel tangent (or ParTan) gradient [7].

```
\[ W_{k+1} = W_k - \eta_k \nabla E(W_k) + \mu_k A_k \]
```

Figure 3 shows a schematic diagram of parallel tangent gradient. Note that the points here have been numbered such that the odd-numbers after \( P_3 \) (i.e. \( P_5, P_7, P_9 \), etc.) are the result of gradient search descent, whereas the even-numbers after \( P_2 \) (that is \( P_4, P_6, P_8 \), etc.) are obtained by acceleration. In other words, the even-numbered points \( P_{2k} \) is determined by acceleration from \( P_{2k-4} \) through \( P_{2k-1} \), where \( k=2,3,...,N \) i.e.

\[ P_{2k} = \Omega(P_{2k-1}, P_{2k-4}) \quad k \geq 2 \]  

where \( \Omega \) is the acceleration function. So the acceleration is indeed the process of taking the minimum point on the line connecting \( P_{2k-1} \) and \( P_{2k-4} \).

\[ \nabla E(W_{k}) \quad \text{and} \quad A_k \] 

are calculated gradient and acceleration directions, \( \eta_k \) and \( \mu_k \) are suitable gradient and acceleration learning rates, all in epoch \( k \).

The Figure 4 shows the proposed algorithm.

```
- Initialize weights
- Compute gradient
- Adapt gradient learning rate
- Do one Gradient step
- While (Termination Condition not meet) {
   Adapt gradient learning rate
   Do one Gradient step
   Adapt acceleration learning rate
   Do one acceleration step
}
```

3. Proposed Learning algorithm

The proposed algorithm employs the acceleration direction based on parallel tangent as a deflected direction instead of momentum one. Thus to accelerate convergence, it uses an acceleration step after each gradient step. Therefore the weight vector can be updated as:

\[ W_{k+1} = W_k - \eta_k \nabla E(W_k) + \mu_k A_k \]

where \( \nabla E(W_k) \) and \( A_k \) are calculated gradient and acceleration directions, \( \eta_k \) and \( \mu_k \) are suitable gradient and acceleration learning rates, all in epoch \( k \). Considering the Figure 3 and Equation 4, acceleration direction \( A_k \) can be computed as:

\[ A_k = P_{2k-1} - P_{2k-4} \]

4. Adaptation of learning rates

In the proposed algorithm, the learning rates are continuously adapted over epochs. The adaptation is done with respect to the shape of error surfaces. For finding a suitable learning rate, we proposed a heuristic line search method for the gradient search direction as well as the acceleration direction. Suppose that at epoch \( k \), the current weight vector is \( W_k \) and \( d \) is a search direction through weight space. The minimum along the search direction then gives the next value for the weight vectors:

\[ W_{k+1} = W_k + \lambda_k d \]

where the parameter \( \lambda_k \) is chosen to minimize

\[ E(\lambda) = E(W_k + \lambda d) \]

The line search represents a one-dimensional minimization problem of error function \( E(W_k) \) in
direction of \( \mathbf{d} \) in MLP neural network, an error function calculation requires one forward propagation and hence needs \( \sim 2NW \) operations, where \( N \) is the number of patterns in data set and \( W \) is the number of weights. An error function gradient evaluation however requires a forward propagation, a backward propagation and a set of multiplications to form the derivatives. It therefore needs \( \sim 5NW \) operations, so an efficient line search can reduce total operations \cite{3}.

For each line search procedure, \( E(\lambda) \) should be firstly evaluated for three different value of \( \lambda \), for example \( \lambda=0, \lambda_1 \) and \( \lambda_2 \) as shown in the Figure 5. In normal case of minimization (like MLP learning), the direction of \( \mathbf{d} \) is such that \( E(\lambda) \) is a decreasing function of \( \lambda \), but because of some mistakes in choosing \( \lambda_1 \) and \( \lambda_2 \) or search direction of \( \mathbf{d} \), this is not true in all cases. Figures 5 through 7 show some different cases that will be described in the following.

### 4.1 \( E(\lambda_1)<E(0) \)

In this condition, \( E(\lambda) \) and \( \lambda \) are inversely proportional, so the search direction \( \mathbf{d} \) as well as \( \lambda_1 \) are proper. Three different cases can be supposed.

**Case I, \( E(\lambda_1)<E(\lambda_2) \):** The Figure 5 represents a bracket case where \( E(\lambda_1)<E(\lambda_2) \) and \( E(\lambda_1)<E(0) \). Since the error function is continuous and smooth, this ensures that there is a minimum \( \lambda_m \) somewhere in the interval \((0, \lambda_2)\). Therefore \( \lambda_m \) can be estimated by quadratic estimation of \( E(\lambda) \) as a function of \( \lambda \), \( \lambda_m \) \cite{4}:

\[
\lambda_m = -0.5 \frac{E(0)(\lambda_2^2 - \lambda_1^2) + E(\lambda_1)\lambda_2^2 - E(\lambda_2)\lambda_1^2}{E(0)(\lambda_2 - \lambda_1) + E(\lambda_2)\lambda_1 - E(\lambda_1)\lambda_2} \tag{9}
\]

**Fig 5. Bracket case when \( \lambda_m \) can be estimated accurately**

**Case II, \( E(\lambda_1)>E(\lambda_2) \):** The Figure 6 depicts this case. Like case I, there is a minimum too but it is beyond of \( \lambda_2 \) and we can estimate \( \lambda_m \) using the Equation 9. Since \( \lambda_m > \lambda_2 \), we should test the correctness of \( \lambda_m \) by checking the condition \( E(\lambda_m)<E(\lambda_2) \). If \( E(\lambda_m)=E(\lambda_2) \) then we consider \( \lambda_m = \lambda_2 \) because the \( \lambda_m \) is not computed properly.

**Fig 6. \( \lambda_m \) is larger than \( \lambda_2 \) but can be estimated using the Equation 9.**

**Case III, \( E(\lambda_1)>E(\lambda_2) \):** The Figure 7 shows this case that is similar to case II and increasing \( \lambda_1 \) or \( \lambda_2 \) can decreases \( E(\lambda) \) \( (E(\lambda_1)<E(\lambda_2)) \). In this condition, \( \lambda_m \) can not be estimated by quadratic estimation of \( E(\lambda) \), but we know that \( \lambda_m \) is larger than \( \lambda_2 \). In our proposed heuristic line search, we suggest \( \lambda_m = \lambda_1 + \lambda_2 \), but like case II, the suggested \( \lambda_m \) will be accepted only if \( E(\lambda_m) \) is smaller than \( E(\lambda_2) \). If not, \( \lambda_2 \) is better as \( \lambda_m = \lambda_2 \).

**Fig 7. \( \lambda_m \) is larger than \( \lambda_2 \) but can not be estimated by quadratic estimation, so we suppose \( \lambda_m = \lambda_1 + \lambda_2 \)**

### 4.2 \( E(\lambda_1)>E(0) \)

This condition shows that the search direction \( \mathbf{d} \) or \( \lambda_1 \) are not chosen properly because \( E(\lambda) \) increases for \( \lambda>0 \). In fact this case can be supposed as a mistake in choosing \( \lambda_1 \) or even search direction \( \mathbf{d} \). Therefore some corrections should be considered for the next epoch. In our proposed heuristic line search strategy, \( \lambda_m \) is restarted to an initializing value \( \lambda_{initialize} \), therefore \( \lambda_m = \lambda_{initialize} \).

### 4.3 Choosing \( \lambda_1 \) and \( \lambda_2 \)

The case I and then II are the best cases, because \( \lambda_m \) can be efficiently estimated. Therefore choosing \( \lambda_1 \) and \( \lambda_2 \) have an important role. However, the case I is
better than II because estimation of $\lambda_m$ is more accurate. We use the dynamic self adaptation algorithm [5] for choosing $\lambda_1$ and $\lambda_2$. This method adapts itself to the local information based on the cost function $E(W_k)$. If $\lambda_m$ was chosen for epoch $k$, $\lambda_1$ and $\lambda_2$ are computed for epoch $(k + 1)$ as:

$$
\begin{align*}
\lambda_1 &= \lambda_m + \zeta \\
\lambda_2 &= \lambda_m - \zeta
\end{align*}
$$

(10)

where $\zeta = 1.839$ is nearly optimum for the elliptical contours [5] that is a good approximation for error function of MLP neural network with Sigmoid activation functions.

4.4 Overall Adaptive Rate Algorithm

As mentioned before we have two separate learning rates, $\eta_k$ and $\mu_k$ that are for climbing in gradient direction ($-\nabla E(W_k)$) and for acceleration direction ($\lambda_k$), in $k$th epoch. The overall process of adaptation of learning rates in the Figures 5 to 7 can be explained by the Figure 8.

1-Compute $\lambda_1$ using the Equation 10.
2-If $E(\lambda_1) > E(0)$ then $\lambda_m = \lambda_{\text{Initialize}}$ and return.
3-Compute $\lambda_2$ using the Equation 10.
4-If $E(\lambda_2) > E(\lambda_1)$ (case I) then compute $\lambda_m$ using the Equation 9 and return.
5-In case II, compute $\lambda_m$ using the Equation 9, if $E(\lambda_m) > E(\lambda_2)$ then $\lambda_m = \lambda_2$ and return.
6-In case III, $\lambda_m = \lambda_1 + \lambda_2$, if $E(\lambda_m) > E(\lambda_2)$ then $\lambda_m = \lambda_2$

Fig 8. The proposed strategy for finding adaptively a suitable learning rate

5. Experimental Results

In order to evaluate the performance of the proposed learning algorithm, some experimental studies are carried out on different MLP neural network problems. Since our algorithm is a combination of Parallel Tangent and a heuristic line Search, we call it PATLIS.

We compared the results of our proposed algorithm, PATALS with dynamic self adaptation of both learning rates including gradient and momentum (DS-$\eta$-$\alpha$) and parallel tangent dynamic self adaptation (PTDS-$\eta$-$\mu$) [8]. In PTDS-$\eta$-$\mu$, we used parallel tangent direction as a deflecting direction and also used two distinct variable learning rates for gradient as well as parallel tangent based on the Equation 2.

The different MLP problems are chosen so that they cover different error surfaces. The excitation function of neurons is Sigmoid and the experiments are accomplished using batch update scheme. Since gradient based learning algorithms are sensitive to different starting point, we carried out our experiment with more than 5000 different runs from random initialization points between $-r$ to $r$ with uniform distribution.

$$
\eta = \frac{3}{\sqrt{N_l + 1}}
$$

Where $N_l$ is the number of neurons in layer $l$. All chosen MLP problems have three layers, an input layer, a hidden layer and an output one. The first chosen case is XOR which is a classical MLP problem with two input and one output layer. The MLP architecture for XOR case is $2 \times 2 \times 1$ that contains two nodes in the hidden layer. Encoders constitute well-known test cases for any learning procedure. The second chosen case is a 4bit Encoder with $16 \times 4 \times 16$ architecture. This MLP neural network has 16 learning patterns. Each pattern has only one '1' in 16 bits and other 15 bits are set to '0'. In Encoder problems, the input learning patterns and output ones are the same.

The final case for MLP learning is a 4bit Parity problem with $4 \times 4 \times 1$ architecture that is considered as an extension to the XOR case. A Parity MLP neural network has a single output that should be '1' if an odd number of inputs were '1', otherwise the output is zero. The Parity $4 \times 4 \times 1$ is a challenging problem that contains some local minima.

Some statistics results of our simulation studies are summarized in the Table 1. This statistics includes average number of epochs that were successful in the best 500 runs and reached on the preset Max-Error as well as successful rates in more than 5000 runs.

The initialized value of learning rates of gradient ($\eta$), parallel tangent ($\mu$) and momentum ($\alpha$) are all set 0.1. Considering the error function, the learning rates are adjusted during learning process. The learning process is terminated if the convergence criterion Max-Error is below a predefined threshold. The Max-Error is the maximum error of output nodes that was set to 0.447 in all tested algorithms. Moreover, at the start of each simulation, the weights are initialized to random value between $-r$ to $r$ which were chosen based on the Equation 11.

Epoch complexity is one the most important factors in each optimization algorithm. For the tested algorithms, we can estimate the epoch
complexity based on the numbers of required backpropagation and forwards computations. The complexity of each backpropagation computation that includes one forward and one backward is almost \(~5NW\) operations while \(~2NW\) operations are required only for forward one. Where \(N\) is the number of patterns in data set and \(W\) is the number of weights. Each dynamic self adaptation (the Equation 2) includes two forward computations so it needs to about \(~4NW\) operations. Therefore, the epoch complexity of DS \(\eta-\alpha\) and PTDS \(\eta-\mu\) is about \(~13NW\) which includes \(~5NW\) for backpropagation and \(~4NW\) for dynamic self adaptation of gradient rate and \(~4NW\) for dynamic self adaptation of momentum rate (in DS \(\eta-\alpha\)) or parallel tangent rate (in PTDS \(\eta-\mu\)). The proposed algorithm –PATLIS-sometimes needs to compute the error value for \(\lambda_m\) (refer to the adaptive learning in PATLIS, the Figure 8, stage 5 and 6), so the maximum epoch complexity is \(~17NW\). On the other word sometimes only one forward computation is need (refer to the Figure 8, stage 2). Our investigation shows that the epoch complexity of PATLIS is averagely around \(~15NW\) instead of maximum \(~17NW\).

As shown in Table 1, PATLIS average epochs were reduced about 50.2% and 66.8% for DS \(\eta-\alpha\) and PTDS \(\eta-\mu\) respectively. Considering the epoch complexity of all tested algorithm, PATLIS showed about 61.8% and 42.6% reduction respect to the benchmark algorithms.

The successful rate of the attempts has a very important role in practically neural network learning, especially in large neural networks. The learning algorithm with a high successful rate shows that it can avoid local minimum and therefore has more chances to find acceptable weights.

### 4 Conclusion

In this paper we introduced PATLIS algorithm for MLP neural network. To decrease zigzagging phenomena and increase speed of convergence, PATLIS used parallel tangent as a deflecting direction instead of momentum one. Moreover, a powerful adaptive learning rate computation based on a heuristic line search and some simple hypothesis, was employed. Experimental results showed that not only the average number of epochs as well as complexity was noticeably decreased, but also successful rates of attempts were increased too. Therefore, PATLIS algorithm can decrease the learning time and also properly avoids local minimum hazard compared to other algorithms.

### References:


<table>
<thead>
<tr>
<th>Problems</th>
<th>Algorithm</th>
<th>DS (\eta-\alpha)</th>
<th>PTDS (\eta-\mu)</th>
<th>PATLIS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Epochs</td>
<td>Success</td>
<td>Epochs</td>
<td>Success</td>
</tr>
<tr>
<td>XOR (2\times2\times1)</td>
<td>19.1</td>
<td>63%</td>
<td>13.6</td>
<td>46%</td>
</tr>
<tr>
<td>Encoder (16\times4\times16)</td>
<td>142.6</td>
<td>11.7%</td>
<td>49.6</td>
<td>64%</td>
</tr>
<tr>
<td>Parity (4\times4\times1)</td>
<td>150.9</td>
<td>11.5%</td>
<td>144.7</td>
<td>8.8%</td>
</tr>
<tr>
<td>Epoch Complexity</td>
<td>13NW</td>
<td></td>
<td>13NW</td>
<td></td>
</tr>
</tbody>
</table>