Improving Gradient Descent Method for Training Feed Forward Neural Networks

Luma. N. M. Tawfiq

Department of Mathematics, College of Education for science/Ibn- Al-Haitham, University of Baghdad

*Author to whom correspondence should be addressed; E-Mail: drlumanaji@yahoo.com

Article history: Received 20 May 2013, Accepted 24 June 2013, Published 25 June 2013.

Abstract: In this paper, many modified and new algorithms have been proposed for training feed forward neural networks, many of them having a very fast convergence rate for reasonable size networks. In all of these algorithms we use the gradient of the performance function (energy function, error function) to determine how to adjust the weights such that the performance function is minimized, where the back propagation algorithm has been used to increase the speed of training. The above algorithms have a variety of different computation and thus different type of form of search direction and storage requirements, and all the above algorithms applied in approximation problem.

Keywords: ANN, FFNN, training back propagation.

1. Introduction

Back propagation (BP) process can train multilayer feed forward neural network (FFNN). With differentiable transfer functions, to perform a function approximation to continuous function $f \in \mathbb{R}^N$, pattern association and pattern classification. The term of back propagation to the process by which derivatives of network error with respect to network weights and biases, can be computed. This process can be used with a number of different optimization strategies.

2. Training Algorithms for Neural Networks

Any non-linear optimization method, a local or global one, can be applied to the optimization of feed-forward neural networks weights. Naturally, local searches are fundamentally limited to local
solutions, while global ones attempt to avoid this limitation. The training performance varies depending on the objective function (energy function or error function) and underlying error surface for a given problem and network configuration.

Since the gradient information of error surface is available for the most widely applied network configurations, the most popular optimization methods have been variants of gradient based back-propagation algorithms. Of course, this is sometimes the result of an inseparable combination of network configuration and training algorithm which limits the freedom to choose the optimization method.

Widely applied methods are, for example, modified back-propagation [1], back propagation using the conjugate-gradient approach [2], scaled conjugate-gradient and its stochastic counterpart [3], the Marquadt algorithm [4], and a concept learning based back-propagation [5]. Many of these gradient based methods are studied and discussed even for large networks in [6]. Several methods have been proposed for network configurations where the gradient information is not available, such as simulated annealing for networks with non-differentiable transfer functions [7].

In many studies only small network configurations are considered in training experiments. Many gradient based methods and especially the Levenberg-Marquadt method are extremely fast for small networks (few hundreds of parameters), thus, leaving no room or motivation for discussion of using evolutionary approaches in the cases where the required gradient information is available. The problem of local minima can be efficiently avoided for small networks by using repeated trainings and randomly initialized weight values. Nevertheless, evolutionary based global optimization algorithms may be useful for validation of an optimal solution achieved by a gradient based method.

For large FFNN’s, consisting of thousands of neurons, the most efficient training methods (Levenberg – Marquadt , Quasi – Newton, etc.) demand an unreasonable amount of computation due to their computational complexity in time and space. One possibility could be a hybrid of traditional optimization methods and evolutionary algorithms as studied in [8]. Unfortunately, it seems that none of the contemporary methods can offer superior performance over all other methods on all problem domains. It seems that no single solution appears to be available for the training of artificial neural networks.

2.1. Gradient (Steepest) Descent (taringd)

A standard back propagation algorithm is a gradient descent algorithm (as in the Widrow-Hoff learning rule). For the basic steepest (gradient) descent algorithm, the weights and biases are moved in the direction of the negative gradient of the performance function.

For the method of gradient descent, the weight update is given by:
\[ W_{k+1} = W_k + \eta_k (-g_k) \]

where \( \eta_k \) is a parameter governing the speed of learning, named learning rate, controlling the distance between \( W_{k+1} \) and \( W_k \) and \( g_k \) is the gradient of the error surface at \( W_k \). \( W_k \) is the weight at iteration \( k \) [9], [10].

The convergence condition is satisfied by choosing: 
\[ 0 < \eta_k < \frac{1}{2\lambda_{\max}} \]

where \( \lambda_{\max} \) is the largest eigen value of weight matrix.

2.2. Gradient Descent With Momentum (traingdm) [11]

There is another training algorithm for FFNN that often provides faster convergence. The weight update formula for gradient descent with momentum is given by:

\[ W_{k+1} = W_k + \eta_k (-g_k) + \mu (W_k - W_{k-1}) \]

That is:
\[ W_{k+1} = W_k + \eta_k (-g_k) + \mu \Delta W_k \]

i.e.
\[ \Delta W_{k+1} = \eta_k (-g_k) + \mu \Delta W_k , \]

where the momentum parameter \( \mu \) is constrained to be in the range \((0, 1)\). Momentum allows the ANN to make reasonably large weight adjustments, while using a smaller learning rate to prevent a large response to the error from any one of training pattern.

The gradient is constant ( \( g_k = \text{const} \) ). Then, by applying iteratively (2):
\[ \Delta W = - \eta g_k (1 + \mu + \mu^2 + \ldots) \approx - \frac{\eta}{1 - \mu} g_k \]

(because \( \mu \in (0,1) \) and then \( \lim_{n \to \infty} \mu^n = 0 \)), i.e., the learning rate effectively increases from \( \eta \) to \( \frac{\eta}{1 - \mu} \).

Remark

There are several issues on gradient descent training algorithms:

1) When the learning rate \( \eta \) is too small, the learning algorithm converges very slowly. However, when \( \eta \) is too large, the algorithm becomes unstable and diverges.

2) Another peculiarity of the error surface that impacts the performance of the gradient descent training algorithm is the presence of local minima [12]. It is undesirable that the learning algorithm stops at a local minimum if it is located far above a global minimum.

3) Neural network may be over-trained by using gradient descent algorithms and obtain worse generalization performance. Thus, validation and suitable stopping methods are required in the cost function minimization procedure.
4) Gradient-based training is very time-consuming in most applications.

The aim of this paper is to solve the above issues related with gradient-based algorithms and propose an efficient training algorithm for FFNNs

3. Improve Gradient Descent Method

Let consider the Taylor series development of error $E$ around a point $W_k$, note that here $W$ is being seen as a vector:

$$E(W_{k+1}) = E(W_k) + g_k^T (W_{k+1} - W_k) + \frac{1}{2} (W_{k+1} - W_k)^T H W_k (W_{k+1} - W_k)$$

Then the gradient of $E$ with respect to $W$ may be approximated from (3) as:

$$g_{k+1} = g_k + H_k (W_{k+1} - W_k)$$

We want to improving the gradient descent method by making $g_{k+1}$ orthogonal with the previous gradient $g_k$, then we must choose $\eta$ such that:

$$g_{k+1}^T g_k = 0$$

From adjusts weights by gradient descent method we have:

$$W_{k+1} - W_k = - \eta_k g_k$$

and substituting in (4) we get:

$$g_{k+1} = g_k - \eta_k H_k g_k$$

Multiply (6) from the left by $g_k^T$ and use (5), then we get:

$$0 = g_k^T g_k - \eta_k g_k^T H_k g_k$$

Then we get new form of learning rate:

$$\eta_k = \frac{g_k^T g_k}{g_k^T H_k g_k}$$

Then (7) give better approach to choose $\eta$ at each iteration $k$.

We implement (7) in a new package using MATLAB 7, named 'traingdak'.

Now consider the Taylor series about $k = 0$, then:

$$E(W) = E(W_0) + g_0^T (W-W_0) + \frac{1}{2} (W-W_0)^T H_0 (W-W_0)$$

Then the gradient of $E$ with respect to $W$ may be approximated as:

$$g = g_0 + H_0 (W-W_0)$$

Because the Hessian is a symmetric matrix then it is possible to find an orthonormal set of eigenvectors $\{ u_i \}$ with the corresponding eigen values $\{ \lambda_i \}$ (i=1,2,...,Nw , where Nw is the number of weights ) such that:
\[ \text{Hu}_i = \lambda_i u_i \quad ; \quad u_i^T u_j = \delta_{ij} = u_i^T u_j = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \] (10)

(\lambda_i and u_i being calculated at the minimum of \(E\), given by the point \(W_0\)).

By writing \((W - W_0)\) and \(g_0\) in terms of \{\(u_i\}\)

\[ \begin{align*}
\text{W} - W_0 &= \sum_{i=1}^{N_v} \alpha_i u_i \\
g_0 &= \sum_{i=1}^{N_v} \gamma_i u_i
\end{align*} \] (11)

(where \(\alpha_i\) and \(\gamma_i\) are the coefficients of the \{\(u_i\)\} development) and replacing in (9) respectively:

\[ \begin{align*}
g &= \sum_{i=1}^{N_v} \gamma_i u_i + H_0 \sum_{i=1}^{N_v} \alpha_i u_i = \sum_{i=1}^{N_v} \gamma_i u_i + \sum_{i=1}^{N_v} \alpha_i H_0 u_i = \sum_{i=1}^{N_v} (\gamma_i + \alpha_i \lambda) u_i
\end{align*} \] (13)

From (11):

\[ \text{W}_{k+1} - W_k = (\text{W}_{k+1} - W_0) - (W_k - W_0) = \sum_{i=1}^{N_v} (\alpha_{i(k+1)} - \alpha_{i(k)}) u_i \] (14)

Since \(\text{W}_{k+1} - W_k = -\eta g_k\) & from (13) & (14) we get:

\[ \sum_{i=1}^{N_v} (\alpha_{i(k+1)} - \alpha_{i(k)}) u_i = -\eta \sum_{i=1}^{N_v} (\gamma_i + \alpha_i \lambda) u_i \]

Then:

\[ \sum_{i=1}^{N_v} (\alpha_{i(k+1)} - \alpha_{i(k)}) + \eta \gamma_i + \eta \alpha_{i(k)} \lambda_i u_i = 0 \] (15)

Since \(u_i^T\) is linearly independent, then \(u_i^T \neq 0\), we get:

\[ \alpha_{i(k+1)} - \alpha_{i(k)} + \eta \gamma_i + \eta \alpha_{i(k)} \lambda_i = 0, \quad \forall i \]

Then:

\[ \alpha_{i(k+1)} = (1 - \eta \lambda_i) \alpha_{i(k)} - \eta \gamma_i, \quad \forall i \] (16)

iteratively \(k+1\) step (16) gives:

\[ \alpha_{i(k+1)} = (1 - \eta \lambda_i)^k \alpha_{i(0)} - \eta \gamma_i \sum_{j=0}^{k} (1 - \eta \lambda_i)^j \] (17)

If \(\alpha_i(0) = 0\) \(\forall i\) , and substituting in (17) we get :

\[ \alpha_{i(k+1)} = -\eta \gamma_i \sum_{j=0}^{k} (1 - \eta \lambda_i)^j \] (18)
Compute (11) for \( k+1 \) we have: 
\[ W_{k+1} - W_0 = \sum_{i=1}^{N} \alpha_{i(k+1)} u_i \]

and substituting (18) in above equation we get:
\[ W_{k+1} = W_0 - \eta \sum_{i=1}^{N} \gamma_i \sum_{j=0}^{k} (1-\eta \lambda_i)^j u_i \]

(19)

By (19) we can calculate adjust weight \( W_{k+1} \) for any iterative \( k \) in one-epoch only and depending on \( W_0 \) only.

Then the improvement gradient descent method, hold by calculating \( M \) (any integer number) iterative in one-epoch instead of calculate one iterative in one-epoch.

**Remark**

- When we choose \( M \) very large then the time of calculating becomes very large.
- In practical of this paper we choose \( M = 5000 \) and we use the MATLAB package named 'traingdrl' also, \( M = 1000 \) and we use the MATLAB package named 'traingdrl2'.

Also, we speeding (19) by reinstating geometrical series into converge point (limit point) of the series by the following:
\[ W_{k+1} = W_0 - \sum_{i=1}^{N} \sum_{j=0}^{k} \gamma_i (1-\eta \lambda_i)^j u_i \]

In practical of this paper we use the MATLAB package named 'traingdl'.

### 4. Learning Parameter and Convergence for Gradient Descent Method

Considering the quadratic approximation of the error function in the neighborhood of minima, from (8), when \( g_0 = 0 \), we have:
\[ E(W) = E(W_0) + \frac{1}{2} (W - W_0)^T H_0 (W - W_0) \]

Then:
\[ g = (W - W_0)^T H_0 \]

substituting (8):
\[ g = (\sum_{i=1}^{N} \alpha_i u_i)^T H_0 \]

and from (10), we get:
\[ g = \sum_{i=1}^{N} \alpha_i \lambda_i u_i \]

(20)

\[ W_{k+1} = W_k + \sum_{i=1}^{N} \alpha_{i(k+1)} u_i - \sum_{i=1}^{N} \alpha_{i(k)} u_i \]

from (10)
Then: \( \Delta W_{(k+1)} = \sum_{i=1}^{N_w} \Delta \alpha_i u_i \), where \( \Delta \alpha_i = \alpha_{(i+1)} - \alpha_{(i)} \) \( (21) \)

Since: \( \Delta W_{(k+1)} = - \eta g_k \), then: \( \Delta W_{(k+1)} = \sum_{i=1}^{N_w} \Delta \alpha_i u_i = - \eta \sum_{i=1}^{N_w} \alpha_{(i(k))} \lambda_i u_i \)

And because the eigenvectors \( u_i \) are orthonormalated, then:
\[
\Delta \alpha_i = - \eta \lambda_i \alpha_{(k)},
\]
then by applying \((3.23)\) we get:
\[
\alpha_{(i(k+1))} = (1 - \eta \lambda_i) \alpha_{(i(k))}
\]

\( (22) \)

Again from \((11)\), by multiplying with \( U_i^T \) to the left, and from the orthonormalation of \( \{u_i\} \):
\[
U_i^T (W - W^*) = \alpha_{(i)}
\]
i.e., \( \alpha_{(i)} \) represents the distance (into the weights space) to the minimum, along the direction given by \( u_{(i)} \).

After \( K \) steps the iterative usage of \((22)\) gives:
\[
\alpha_{(K)} = (1 - \eta \lambda_i)^K \alpha_{(0)}
\]

\( (23) \)

A convergence process means that \( \lim_{k \to \infty} W_{(k)} = W^* \), i.e., \( \lim_{k \to \infty} \alpha_{(i(k))} = 0 \) (by considering the significance of \( \alpha_{(i)} \) as discussed above). Then, to have a convergent training, formula \((23)\) shows that it is necessary to impose the condition:
\[
|1 - \eta \lambda_i| < 1, \ \forall i \ \rightarrow \ 0 < \eta < \frac{2}{\lambda_{\max}}
\]

\( (24) \)

Where \( \lambda_{\max} \) is the biggest eigenvalue of the Hessian matrix \( H_0 \).

### 5. Initialization and Termination of Training

Usually the weights are initialized with random values to avoid problems due to weight space symmetry. However there are two restrictions:

- If the initial weights are too big then the activation functions \( f \) will have values into the saturation region (e.g. sigmoidal activation function) and their derivatives \( f' \) will be small, leading to a small error gradient as well, i.e. an approximatively at error surface and, consequently, a slow training.
- If the initial weights are too small then the activation functions \( f \) will be linear and their derivatives will be quasi-constant, the second derivatives will be small and then the Hessian will be small meaning that around minimums the error surface will be approximatively at end, consequently, a slow training.

Another way to improve network performance is to train multiple instances of the same network, but with a different set of initial weights, and choosing among those who give best results. This method is called committee of networks.
The criteria for stopping the training process may be one of the following:

- Stop after a fixed number of steps.
- Stop when the error function had become smaller than a specified amount.
- Stop when the change in the error function ($\Delta E$) had become smaller than a specified amount.
- Stop when the error on an (independent) validation set begins to increase.

6. Application

We applied multilayer FFNN with ridge basis function have linear output units and a single hidden layer of hyperbolic tangent hidden units (nodes). The number of hidden nodes in all problems is $2N+1$, where $N$ is number of input nodes. The training problems used problem domains function approximation and we training each problems 9 different times and the weights of the networks computed by back propagation algorithm with training algorithm: 'traingd', 'traingddr', 'traingddrl2',

**Problem 1**

$$F(x) = 3x (x - 0.6)(x + 1.17); \quad \text{where} \quad 0 \leq x \leq 1$$

The numerical results of ridge basis function FFNN with network structure 1–3–1, introduced in table 1, figure 2, illustrate the target function of FFNN training by "traingddl".

Figure 1 illustrates the deviation $\Delta F(x)$ of the approximate results by using traingdrl and traingddrl2 training algorithms from the exact function.

![Figure 1: The deviation $\Delta F(x)$ of the approximate results by using "traingdl" and "traingddrl" from the exact function](image-url)
Figure 2: Illustrate the target function of FFNN training by "traingdrl"

Table 1: Training results over several independent trials for problem 1

<table>
<thead>
<tr>
<th>NO.</th>
<th>TrainFcn</th>
<th>PerfTest</th>
<th>Time/S</th>
<th>Epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>traingdrl</td>
<td>0.00033101</td>
<td>1.312</td>
<td>243</td>
</tr>
<tr>
<td>2</td>
<td>traingdrl</td>
<td>0.00055644</td>
<td>27.735</td>
<td>5394</td>
</tr>
<tr>
<td>3</td>
<td>traingdrl</td>
<td>0.00055679</td>
<td>21.797</td>
<td>12481</td>
</tr>
<tr>
<td>4</td>
<td>traingdl</td>
<td>0.00055863</td>
<td>37.344</td>
<td>7561</td>
</tr>
</tbody>
</table>

The parameters

<table>
<thead>
<tr>
<th>Lr</th>
<th>0.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mc</td>
<td>0.1</td>
</tr>
<tr>
<td>lr_inc</td>
<td>1.1</td>
</tr>
<tr>
<td>lr_dec</td>
<td>0.4</td>
</tr>
<tr>
<td>max_perf_inc</td>
<td>1.02</td>
</tr>
<tr>
<td>delta0</td>
<td>0.13</td>
</tr>
<tr>
<td>delt_inc</td>
<td>1.4</td>
</tr>
<tr>
<td>delt_dec</td>
<td>0.7</td>
</tr>
<tr>
<td>delt_max</td>
<td>50</td>
</tr>
<tr>
<td>Epochs</td>
<td>500000</td>
</tr>
<tr>
<td>Time</td>
<td>inf</td>
</tr>
<tr>
<td>Show</td>
<td>500</td>
</tr>
<tr>
<td>Goal</td>
<td>0.0005</td>
</tr>
</tbody>
</table>

The initial Weight and Bias value

<table>
<thead>
<tr>
<th>net.IW{1,1}</th>
<th>net.IW{1,1}</th>
<th>net.IW{1,1}</th>
<th>net.IW{1,1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.4</td>
<td>-0.028035063</td>
<td>-8.4</td>
<td>-0.08706467</td>
</tr>
<tr>
<td>-8.4</td>
<td>0.782597932</td>
<td>4.2</td>
<td></td>
</tr>
<tr>
<td>8.4</td>
<td>0.524193666</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Problem 2

\[ F(x) = (x_1 - x_2)^3 + 1.8x_1x_2 - x_1 + 7x_2 \; ; \; \text{where} \; 0 \leq x_1 \leq 1, \; 0 \leq x_2 \leq 1. \]

The numerical results of ridge basis function FFNN with network structure 2 – 5 – 1, introduced in table 2.

Figure 3 illustrates the exact function for problem 2 and the target function of FFNN training by "traingdrl" illustrated by figure 4.
Figure 3: Exact function for problem 2

Figure 4: Illustrate the target function of FFNN training by "traingdld"

Table 2: Training results over several independent trials for problem 2

<table>
<thead>
<tr>
<th>NO.</th>
<th>TrainFcn</th>
<th>PerfTest</th>
<th>Time/S</th>
<th>Epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>traingdal</td>
<td>3.39E-05</td>
<td>0.406</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>traingdak</td>
<td>0.00010437</td>
<td>12.203</td>
<td>1594</td>
</tr>
<tr>
<td>3</td>
<td>Traingdak1</td>
<td>0.00010647</td>
<td>182.813</td>
<td>20677</td>
</tr>
<tr>
<td>4</td>
<td>Traingdak2</td>
<td>0.00026303</td>
<td>10.906</td>
<td>1535</td>
</tr>
</tbody>
</table>

The initial Weight and Bias value

<table>
<thead>
<tr>
<th></th>
<th>net.IW{1,1}</th>
<th>net.LW{2,1}</th>
<th>net.B{1,1}</th>
<th>net.B{2,1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.883296494</td>
<td>5.557571531</td>
<td>-0.620692505</td>
<td>-7.350929181</td>
<td>-0.69825405</td>
</tr>
<tr>
<td>6.260773688</td>
<td>0.052084796</td>
<td>-0.613137687</td>
<td>-4.721676826</td>
<td></td>
</tr>
<tr>
<td>-5.739059144</td>
<td>2.502638636</td>
<td>0.364446447</td>
<td>1.618210254</td>
<td></td>
</tr>
<tr>
<td>5.826811435</td>
<td>-2.286326472</td>
<td>-0.394471198</td>
<td>-0.205894898</td>
<td></td>
</tr>
<tr>
<td>-3.286975626</td>
<td>-5.328770143</td>
<td>0.083347708</td>
<td>1.177377716</td>
<td></td>
</tr>
</tbody>
</table>

The parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Lr</td>
<td>0.24</td>
</tr>
<tr>
<td>Mε</td>
<td>0.4</td>
</tr>
<tr>
<td>lr_inc</td>
<td>1.01</td>
</tr>
<tr>
<td>lr_dec</td>
<td>0.8</td>
</tr>
<tr>
<td>max_perf_inc</td>
<td>1.01</td>
</tr>
<tr>
<td>delta0</td>
<td>0.2</td>
</tr>
<tr>
<td>delt_inc</td>
<td>1.3</td>
</tr>
<tr>
<td>delt_dec</td>
<td>0.7</td>
</tr>
<tr>
<td>delt_max</td>
<td>50</td>
</tr>
<tr>
<td>Epochs</td>
<td>500000</td>
</tr>
<tr>
<td>Time</td>
<td>inf</td>
</tr>
<tr>
<td>Show</td>
<td>1000</td>
</tr>
<tr>
<td>Goal</td>
<td>0.00005</td>
</tr>
</tbody>
</table>
7. Conclusions

A trained multilayer feed forward neural network is expected to capture the functional relationship among the input-output pattern pairs for the given training data. It is implicitly assumed that the mapping function corresponding to the data is a smooth one. But, due to the limited number of training samples, the problem becomes an ill-posed problem, in the sense that there will be many solutions satisfying the given data, but none of them may be the desired correct one. This happens if the number of free parameters (weights) of the network is very large. Such a situation results in a large error when some other (test) samples are given to validate the network model for the function. This is called "poor generalization" by the network. On the other hand, fewer number of the free parameters may result in a large error even for the training data, and hence a poor approximation to the desired function. The function approximation interpretation of a multilayer feed forward neural network enables us to view different hidden layers of the network performing different functions. For example, the first hidden layer can be interpreted as capturing some local features in the input space. The second hidden layer can be interpreted as capturing some global features. This two-stage approximation has been shown to realize and continuous vector-valued function. The universal approximation theorem seems to suggest that even a single layer of non-linear units would suffice to realize any continuous function. But this result assumes that a hidden layer of unlimited size is available and that the continuous function to be approximated is also available. But it is not useful to realize the function by training a single hidden layer network.

References


