Efficient Training of Backpropagation Neural Networks

Mohammed A. Otair
Department of Management Information Systems
Al-ahllya Amman University
Amman, Jordan
Email Address: otair2005@hotmail.com

Walid A. Salameh
Department of Computer Science-RSS
Princess Summaya University for Science and Technology
11941 Al-Jubaiha, Amman, Jordan
Email Address: walid@psut.edu.jo

Abstract

This paper focuses on gradient-based backpropagation algorithms that use either a common adaptive learning rate for all weights or a separate adaptive learning rate for each weight. The learning-rate adaptation is based on descent techniques and estimates of the local constant that are obtained without additional error function and gradient evaluations. This paper proposed three algorithms to improve the different versions of backpropagation training in terms of both convergence rate and convergence characteristics, such as stable learning and robustness to oscillations. The new modification consists of a simple change in the error signal function. Experiments are conducted to compare and evaluate the convergence behavior of these gradient-based training algorithms with three training problems: XOR, encoding problem, and character recognition, which are a popular training problems.

Key Words

Neural Networks, Backpropagation, Momentum, Delta-Bar-Delta, Optical Backpropagation.

1. Introduction

The Backpropagation (*BP*) algorithm [22, 23] is perhaps the most widely used supervised training algorithm for multi-layered feedforward neural networks. However, in some cases, the standard Backpropagation takes unendurable time to adapt the weights between the units in the network to minimize the mean squared errors between the desired outputs and the actual network outputs [4].

A variety of approaches adapted from numerical analysis have been applied in an attempt to use not only the gradient of the error function but also the second derivative in constructing efficient supervised training algorithms to accelerate the learning process. The research usually focuses on heuristic methods for dynamically adapting the learning rate during training to accelerate the convergence.

There are many researches, which have been proposed to improve this algorithm; some of these researches were developed to speeding up training process [5,8,12,13,26,27]. Other researches have investigated the effect of adaption of momentum factor [14,28,29].

This paper presents three versions of an optical backpropagation (OBP) algorithm, with analysis of its benefits. An OBP algorithm is designed to overcome some of the problems associated with standard BP training using non-linear function, which applied on the output units. One of the important aspects of the proposed algorithm is its ability to escape from local minima with high speed of convergence during the training period. In order to evaluate the performance of proposed algorithm OBP, experiments are carried out on two problems: Xor, and optical character recognition. The results are compared with results obtained from classical BP.

This paper is divided into four sections. In section 2, introduce the different versions of the Backpropagation (*BP*). In section 3 the proposed algorithms are presented. Experimental results are presented in section 4 to evaluate and compare the performance of these algorithms with several other BP methods. Section 5 presents the conclusions.

2. Backpropagation Algorithms

2.1 Standard Backpropagation (BP)

The Backpropagation *BP* learns a predefined set of output example pairs by using a two–phase propagate adapts cycle. As seen in figure 2.1, after an input pattern has been applied as a stimulus to first layer of network units, it is propagated through each upper layer until an output is generated. This output pattern is then compared to the desired output, and an error signal is computed for each output unit.

The signals are then transmitted backward from the output layer to each unit in the intermediate layer that contributes directly to the output. However, each unit in the intermediate layer receives only a portion of the total error signal, based roughly on the relative contribution the unit made to the original output. This process repeats, layer by layer, until each unit in the network has received an error signal that describes its relative contribution to the total error.

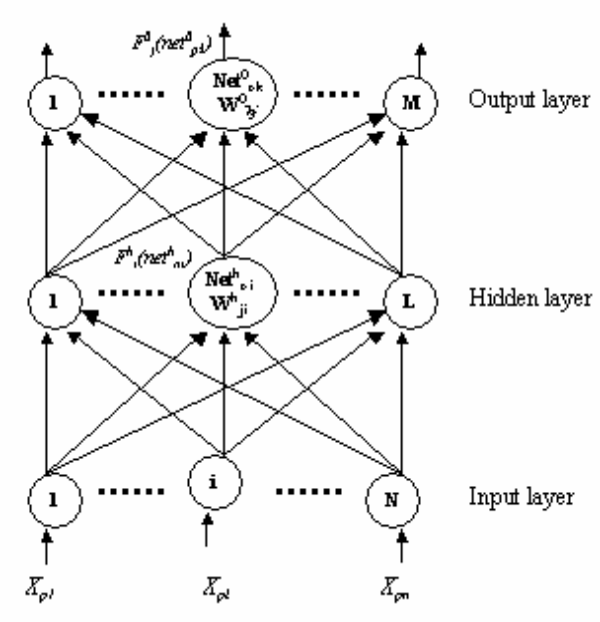


Figure 2.1 - The Three-layer BP Architecture

Freeman and Skapura's book [4] describe the procedure of training feedforward neural networks using the backpropagation algorithm. The detailed formulas are described as follow:

Assume there are *m* input units, *n* hidden units, and *p* output units.

1. Apply the input vector, $X_p=(X_{p1}, X_{p2}, X_{p3}, \ldots, X_{pN})^t$ to the input units .
2. Calculate the net- input values to the hidden layer units:
$$net^h_{pj}=\left(\sum_{i=1}^{N} W^h_{ji} \bullet X_{pi}\right) \quad \textbf{(2.1)}$$
3. Calculate the outputs from the hidden layer:
$$i_{pj} = f^h_j(net^h_{pj}) \quad \textbf{(2.2)}$$
4. Move to the output layer. Calculate the net-input values to each unit:
$$net^o_{pk}=\left(\sum_{j=1}^{L} W^o_{kj} \bullet i_{pj}\right) \quad \textbf{(2.3)}$$
5. Calculate the outputs:
$$O_{pk} = f^o_j(net^o_{pk}) \quad \textbf{(2.4)}$$
6. Calculate the error terms for the output units:
$$\delta^o_{pk} = (Y_{pk} - O_{pk}) \bullet f^{o\prime}_k(net^o_{pk}) \quad \textbf{(2.5)}$$
Where,
$$f^{o'}_k(net^o_{pk}) = f^o_k(net^o_{pk}) \bullet (1 - f^o_k(net^o_{pk})) \quad \textbf{(2.6)}$$
7. Calculate the error terms for the hidden units
$$\delta^h_{pj} = f^{h'}_j(net^h_{pj}) \bullet \left(\sum_{K=1}^{M} \delta^o_{pk} \bullet W^o_{kj}\right) \quad \textbf{(2.7)}$$
Notice that the error terms on the hidden units are calculated *before* the connection weights to the output-layer units have been updated.
8. Update weights on the output layer
$$W^o_{kj}(t+1) = W^o_{kj}(t) + (\eta \bullet \delta^o_{pk} \bullet i_{pj}) \quad \textbf{(2.8)}$$
9. Update weights on the Hidden layer
$$W^h_{ji}(t+1) = W^h_{ji}(t) + (\eta \bullet \delta^h_{pj} \bullet X_i) \quad \textbf{(2.9)}$$

2.2 Backpropagation With Momentum (BPM)

The error Backpropagation training algorithm (BP) which is an iterative gradient descent algorithm is a simple way to train multilayer feedforward neural networks [23]. The BP is based on the gradient descent rule:

$$\Delta W_{ij(n)} = (\eta \bullet \delta_j \bullet X_i) + \mu \bullet (\Delta W_{ij}(n)) \quad \textbf{(2.10)}$$

Where W_{ij} is the weights from unit *i* to *j*, η is the learning rate, X_{ji} is the value of the input into the weight, μ is the momentum factor, and δ_j is the back-propagated error term.

Momentum is neural networks behave similarly to momentum in physics science, in [25] momentum is defined:

> "To see the effect of this momentum term, consider that the gradient descent search trajectory is analogous to that of (momentumless) ball rolling down the error surface. The effect of μ is to add momentum that tends to keep the ball rolling down error surface "

Many algorithms have been proposed to deal with momentum factor [3,6,12,14,18,20] to speed up standard backpropagation. Momentum factor is used speed up training process. However, in many cases, when an improper selection of the momentum factor causes a network to diverge or falls in local minima [12].

2.3 Delta–Bar–Delta

Robert A. Jacobs uses a local learning rate adaptation [6]. In contrary to the former approaches his delta–bar–delta algorithm controls the learning rates by observing the sign changes of an exponential averaged gradient. He increases the learning rates by adding a constant value instead of multiplying it:

1. Choose some small initial value for every learning rate.
2. Adapt the learning rates.

$$\Delta\eta_{ij}(n+1)=\begin{cases}\eta_{ij}(n)+k & ,\text{If } \bar{\Delta}_{ij}(n-1)*\Delta_{ij}(n)>0\\ (1-\gamma)*\eta_{ij}(n) & ,\text{If } \bar{\Delta}_{ij}(n-1)*\Delta_{ij}(n)<0\\ \eta_{ij}(n) & ,\text{Otherwise}\end{cases} \quad \textbf{(2.11)}$$

γ is the exponential decay factor and *K* is the linear increment factor.

The delta value for each neuron is calculated as:

$$\Delta_{ij}=-\delta_j * x_i \quad \textbf{(2.12)}$$

Where δ_j is the error at a single output were defined in equations: (2.5) for output layer, (2.7) for hidden layer.

The bar-delta value for each neuron is calculated as:

$$\bar{\Delta}_{ij}(n)=(1-\beta)*\Delta_{ij}(n)+\beta*\bar{\Delta}_{ij}(n-1) \quad \textbf{(2.13)}$$

$\bar{\Delta}_{ij}(n)$ is the derivative of the error surface, β is the smoothing constant.

3. Update the weights using equations (2.8), (2.9).

3. Proposed Algorithms

3.1 Optical Backpropagation (OBP)

The backpropagation algorithm as described encounters the following difficulty. When the actual value $f^{o}{}_{k}(net^{o}{}_{pk})$ approaches either extreme value, the factor $f^{o}{}_{k}(net^{o}{}_{pk})\bullet(1-f^{o}{}_{k}(net^{o}{}_{pk}))$ in equation (2.6) makes the error signal very small. This implies that an output unit can be maximally wrong without producing a strong error signal with which the coupling strengths could be significantly adjusted. This retards the search for a minimum in the error. For instance, this occurs when some of the output units are pushed towards the wrong extreme value by competition in the network, thereby not increasing their error signal but instead decreasing it.

The proposed algorithms focus on this delay of the convergence is caused by the derivative of the activation function. Unfortunately, any saturating response function is bound to have this property: near the saturation points the derivative vanishes. We will show, however, that a slightly modified error signal function of the backpropagation algorithm resolves this shortcoming and indeed greatly accelerates the convergence to a solution. This applies not only to the initial approach of the desired values, but also to the final convergence process when the response is already nears the target vector.

In this section, the adjustment of the new algorithm *OBP* [13], will be described at which it would improve the performance of the *BP* algorithm. The convergence speed of the training process can be improved significantly by *OBP* through maximizing the error signal, which will be transmitted backward from the output layer to each unit in the intermediate layer.

In *BP*, the error at a single output unit is defined as:

$$\delta^{o}{}_{pk}=(Y_{pk}-O_{pk}) \quad \textbf{(3.1)}$$

Where the subscript "*P* " refers to the p_{th} training vector, and "*K* " refers to the k_{th} output unit. In this case, Y_{pk} is the desired output value, and O_{pk} is the actual output from k_{th} unit, then δ_{pk} will propagate backward to update the output-layer weights and the hidden-layer weights.

While the error at a single output unit in adjusted *OBP* will be as:

$$New\delta_{pk}=(1+e^{(Y_{pk}-O_{pk})^2})\bullet f'(\sum w_{ij}x_i)$$

$$, \text{ if } (Y-O)>\text{zero}. \quad \textbf{(3.2a)}$$

$$New\delta_{pk}=-(1+e^{(Y_{pk}-O_{pk})^2})\bullet f'(\sum w_{ij}x_i)$$

$$, \text{ if } (Y-O)<\text{zero}. \quad \textbf{(3.2b)}$$

$$New\delta_{pk}=0$$

$$, \text{ if } (Y-O)=\text{zero}. \quad \textbf{(3.2c)}$$

An *OBP* uses three forms of $New\delta^{\circ}{}_{pk}$, because the *exp* function always returns *zero* or *positive* values (and the adapts operation for many output units need to decrease the actual outputs rather than increasing it). This $New\delta^{\circ}{}_{pk}$ will minimize the errors of each output unit more quickly than the old $\delta^{\circ}{}_{pk}$, and the weights on certain units change very *large* from their starting values.

The $New\delta_{pk}$ will propagate backward to update the output-layer weights and the hidden-layer weights. This $New\delta_{pk}$ will minimize the errors of each output unit more quickly than the old δ_{pk}, and the weights on certain units change very *large* from their starting values.

Notice that the equations in step 6,7,8,9 change, but all other equations and steps will be as outlined in section 2 remain unchanged

The steps of an OBP:

1. Apply the input example to the input units.

2. Calculate the net-input values to the hidden layer units.
3. Calculate the outputs from the hidden layer.
4. Calculate the net-input values to the output layer units.
5. Calculate the outputs from the output units.
6. Calculate the error term for the output units, using the *Newδ°pk (using equations (3.2a) (3.2b).)* instead of *δ°pk*.
7. Calculate the error term for the hidden units, through applying *Newδ°pk*, also.

$$New\delta^{h}{}_{pj} = f^{h'}{}_{j}(net^{h}{}_{pj}) \bullet (\sum_{K=1}^{M} New\delta^{o}{}_{pk} \bullet W^{o}{}_{kj}) \quad \textbf{(3.3)}$$

8. Update weights on the output layer.

$$W^{o}kj(t+1) = W^{o}kj(t) + (\eta \bullet New\delta^{o}pk \bullet i_{pj}) \quad \textbf{(3.4)}$$

9. Update weights on the hidden layer.

$$W^{h}ji(t+1) = W^{h}ji(t) + (\eta \bullet New\delta^{h}pj \bullet X_{i}) \quad \textbf{(3.5)}$$

10. Repeat steps from step **1** to step **9** until the error ($Y_{pk} - O_{pk}$) is acceptably small for each training vector pairs.

The proposed algorithm as classical BP is stop when the squares of the differences between the actual and target values summed over the output units and all patterns are acceptably small.

3.2 Optical Backpropagation with Momentum Factor (OBP-M)

The proposed algorithm OBP-M applied OBP algorithm with momentum factor to speed up the training process with standard backpropagation with momentum (BPM).

OBP-M uses the same techniques and equations, which used in OBP except the equation (3.4) in step 8 which is changed to be as follows:

$$\Delta W^{o}{}_{kj}(n) = -\eta * New\delta^{o}{}_{pk} * i_{pj} + (\alpha * \Delta W_{ij}(n-1)) \quad \textbf{(3.6)}$$

and equation (3.5) in step 9 which is changed to be as follows:

$$\Delta W^{h}{}_{ji}(n) = -\eta * New\delta^{h}{}_{ji} * x_{i} + (\alpha * \Delta W_{ji}(n-1)) \quad \textbf{(3.7)}$$

The OBP-M algorithm applies the *momentum* term, which is added to the weight change and is proportional to the previous weight change.

The convergence speed of the training process can be improved significantly by *OBP* through adjusting the error especially when a momentum factor is used.

3.3 Enhanced Version of Delta-Bar-Delta (EVDBD)

An Enhanced version of Delta-Bar-Delta (EVDBD) algorithm is an extension of the Delta-Bar-Delta algorithm as a natural outgrowth from Jacob's work [6], EVDBD is the same as Delta-Bar-Delta which introduced by Jacobs as outlined in section 2.3 except that the proposed algorithm uses an Optical Backpropagation OBP rather than BP network (i.e. that EVDBD uses all steps from 1 to 10 in section 3.1 to updates its weights rather than classical BP).

4. Experimental Evaluation

In this section , eight training algorithms are implemented which are: Backpropagation (BP) [22], Backpropagation with momentum (BPM) [22], QuickProp (QP) [3], Delta-Bar-Delta (DBD) [6], Resilient Backpropagation (RPROP) [20] , Optical Backpropagation (OBP) [13] , Optical Backpropagation with momentum (OBPM) [14] , and Enhanced Version of Delta-Bar-Delta (EVDBD) [15]. Most algorithms have been experimented the following neural network problems:

4.1 XOR Problem (XOR 2-2-1)

The implement of the OBP algorithm to solve the XOR problem is very important because the XOR problem requires hidden layers and many other difficult problems involve an XOR as a subproblem.

The XOR problem will be solved using neural network which consists of two input units, two hidden units, and single output unit, with biases for hidden unit and the output unit, without direct connection from input to the output layers, (this network is labeled as 2-2-1). To train this network all initial weights will start as shown in figure 4.1 for all training processes with for this network [16].

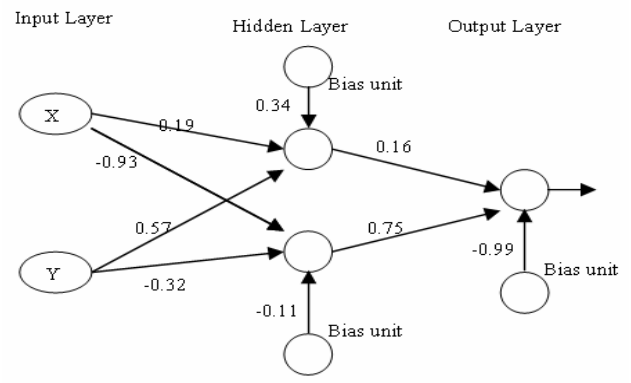


Figure 4.1: *Initial Weights for XOR (2-2-1) problem*

Table 6.3 shows the parameters to solve the XOR (2-2-1) problem using the following algorithms: BP, BPM, QP, DBD, OBP, EVDBD, and OBPM respectively.

Table 4.1: *The parameters to solve the XOR (2-2-1) problem*

Alg. / Parameters	BP	BPM	QP	DBD	OBP	EVDBD	OBPM
Initial Weights	-0.5 ~ +0.5	-0.5 ~ +0.5	-0.5 ~ +0.5	-0.5 ~ +0.5	-0.5 ~ +0.5	-0.5 ~ +0.5	-0.5 ~ +0.5
Learning rate η	0.1 ~ 1	0.1 ~ 1	0.1 ~ 1	0.1 ~ 1	0.1 ~ 1	0.1 ~ 1	0.1 ~ 1
Epoch Limit	50000	25000	10000	10000	5000	5000	5000
MSE	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
Momentum μ	N/A	0.5 ~ 0.8	N/A	N/A	N/A	N/A	0.5 ~ 0.8
β	N/A	N/A	N/A	0.6	N/A	0.6	N/A
γ	N/A	N/A	N/A	0	N/A	0.001	N/A
κ	N/A	N/A	N/A	0	N/A	0.001	N/A
MaxFactor	N/A	N/A	0.64	N/A	N/A	N/A	N/A
Shrink Factor	N/A	N/A	1.75	N/A	N/A	N/A	N/A
Epsilon	N/A	N/A	0.55	N/A	N/A	N/A	N/A
Number of Parameters	4	5	7	7	4	7	5

4.1.1 Solve XOR (2-2-1) problem using BP and OBP :

The results of the training processes using the OBP and BP algorithms (targeted to reach *MSE* less than or equals to 0.001) will be explained in table 4.2.

Table 4.2: *Solve XOR (2-2-1) problem Using OBP and BP*

η	OBP	BP
0.1	1640	21304
0.2	911	10339
0.3	713	6772
0.4	656	5018
0.5	659	3980
0.6	671	3295
0.7	651	2810
0.8	593	2448
0.9	521	2169
1	443	1947

Table 4.2 and figure 4.2 show that the OBP algorithm is more efficient that the BP algorithm because; it can speed up the convergence rate. Also all experiments have shown that the performance of the OBP algorithm is better when the network is trained with smaller learning rate.

It is well known that if a small learning rate were used, then this will be able the network to escape from local minima. However, those will leads to slow down the training process. For that the OBP algorithm helps to speeds up the training process with the usage of a very small learning rate (For example, if we take the number of epochs from table 4.2 using the BP throw a learning rate of 1, we will find that it equals to 1947, while the number of epochs will be 1640 if we used the OBP using learning rate equals to 0.1).

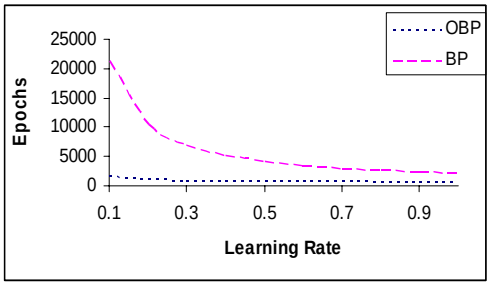


Figure 4.2: *Solve XOR (2-2-1) problem Using OBP and BP*

4.1.2 Solve XOR (2-2-1) problem using OBPM and BPM:

As a result of this experiment, more attention was given to the effect of the momentum factor's on the training process using BPM. Table 4.3 shows all the results.

Table 4.3: *Solve XOR (2-2-1) problem Using BPM and BPM*

η	BPM 0.5	BPM 0.6	BPM 0.7	BPM 0.8
0.1	13702	11653	10165	8894
0.2	6850	5833	5086	4450
0.3	4566	3893	3393	2969
0.4	3423	2923	2546	2229
0.5	2737	2342	2039	1784
0.6	2280	1654	1700	1488
0.7	1953	1678	1459	1277
0.8	1708	1470	1277	1118
0.9	1518	1309	1137	995
1	1365	1180	1024	896

Table 4.3 showed the positive effect of the momentum's usage on the training process. Whereas the larger momentum, means the faster of training process will be.

The table 4.4 shows the required number of epochs using the OBP algorithm and the effect of the momentum with this algorithm. It notice that when use a (0.8) as value for momentum and a learning rate equals 1, the number of epochs will be 43, but using the BP with the same parameters described in the table 4.3. It can notice the difference between the two algorithms in term of number of epochs which is 896.

Table 4.4: *Solve XOR (2-2-1) problem Using OBPM and BPM*

η	OBPM 0.5	OBPM 0.6	OBPM 0.7	OBPM 0.8
0.1	862	592	450	379
0.2	463	309	228	191
0.3	330	221	154	129
0.4	263	184	118	97
0.5	224	167	96	79
0.6	199	156	83	66
0.7	181	147	73	58
0.8	169	140	67	51
0.9	160	135	64	46
1	153	132	67	43

The larger momentum also means the faster of training process goes on, taking into consideration the number of the epochs using the two algorithms.

Figure 4.3 illustrates tables; 4.3 and 4.4. It is noticeable that the curves are divided into of classes: The first class needs a large number of epochs which represents BPM algorithm, while the second class of these curves needs a few number of epochs which represent the OBPM algorithm.

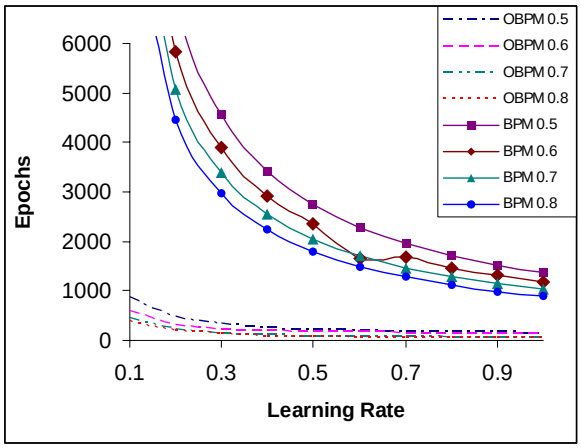


Figure 4.3: *Solve XOR (2-2-1) problem Using OBPM and BPM*

4.1.3 Solve XOR (2-2-1) problem using Seven Algorithms:

In this experiment, seven different algorithms have been compared in term of number of epochs. Three of them were proposed, as been mentioned in the section 3: EVDBD, OBPM and OBP, while the rest of these algorithms are: DBD, QP, BPM and BP. (The parameters to solve the XOR (2-2-1) problem for all algorithms mentioned in table 4.1).

η	BP	BPM 0.8	QP	DBD	OBP	EVDBD	OBPM 0.8
0.1	21304	8894	8022	981	1640	457	379
0.2	10339	4450	2995	956	911	450	191
0.3	6772	2969	2008	930	713	456	129
0.4	5018	2229	1492	905	656	472	97
0.5	3980	1784	1210	881	659	484	79
0.6	3295	1488	985	857	671	479	66
0.7	2810	1277	870	834	651	454	58
0.8	2448	1118	868	812	593	416	51
0.9	2169	995	1025	791	521	373	46
1	1947	896	990	770	443	326	43

Table 4.5: *Solve XOR (2-2-1) problem using Seven Algorithms*

Notice that with the DBD and EVDBD algorithms there is no constant value for the learning rate, instead the learning rate is initialized with values for each training process as shown in the first column in table 4.5 then these learning rates values are adapted through the training epochs.

Figure 4.4 represents the previous table. As you can see, the OBPM (with momentum of 0.8) is the fastest proceeded by the EVDBD which is faster than the OBP. The three of them are the proposed ones. Then, comes the DBD with a close number of epochs to the OBP.

According to the last three algorithms which took a larger number of epochs, they are the QP, BPM and BP respectively. As seen in table 4.1, the OBP and OBPM took less number of parameters (which are problem dependent); besides the two of them are speeding up the training process. Here comes the tradeoff between speed up the training process and the number of parameters which varies from one problem to another, so the best algorithm here is the OBPM.

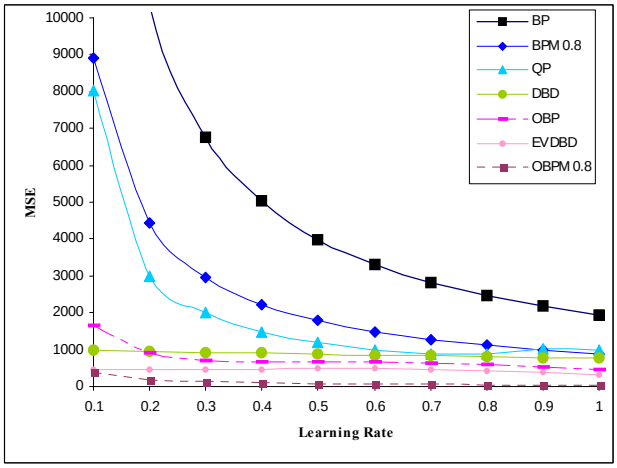


Figure 4.4: *Solve XOR (2-2-1) problem using Seven Algorithms*

4.2 XOR problem (4 bits)

The second problem to be described is the XOR problem with 4-bits (*4-16-1*). The network consists of 4 units in the input layer, and only 1 unit in the output layer, and a hidden layer of 16 units, respectively.

Table 4.6 shows the parameters to solve the XOR (4 bits) problem using the following algorithms: BP, BPM, OBP, OBPM, and RPROP respectively.

Table 4.6: *The parameters to solve the XOR (2, 3 and 4 bits) problem*

Parameters	BP	BPM	OBP	OBPM	RPROP
Initial Weights	-0.1 ~ +0.1	-0.1 ~ +0.1	-0.1 ~ +0.1	-0.1 ~ +0.1	-0.1 ~ +0.1
η	0.05 ~ +0.95	0.05 ~ +0.95	0.05 ~ +0.95	0.05 ~ +0.95	**η+** 1.05
Epoch Limit	75000	75000	25000	15000	50000
MSE	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}
Momentum μ	N/A	0.05 ~ 0.95	N/A	0.05 ~ 0.95	N/A
Δ_0	N/A	N/A	N/A	N/A	0.1
Δ_{max} /Δ_{min}	N/A	N/A	N/A	N/A	$50/10^{-6}$
η-	N/A	N/A	N/A	N/A	0.05 ~ +0.95
# of Parameters	**4**	**5**	**4**	**5**	**7**

Where η+ denotes the increase factor, η- denotes the decrease factor, Δ_0 denotes the initial update weight, Δ_{max} denotes the upper limit, and Δ_{min} denotes the lower limit for RPROP algorithm.

4.2.1 Solve XOR problem (4 bits) using BP, RPROP, and OBP

Table 4.7 represents the training process for this problem using BP, RPROP and OBP, with different values for learning rate between 0.1 and 0.9.

In this experiment again, one of the proposed algorithms was tested, which is the OBP to solve this problem. That is because OBP could solve the XOR problem using different architectures (as mentioned in section 4.1).

In addition, the aim from this experiment was to concentrate on one of the most important variations of the BP which is RPROP, according to the used parameters that have been described in table 4.6. All training processes of the RPROP in table 4.7 used 1.05 as value of the η+ (i.e. fixed). But values of the η- were between 0.1 and 0.9.

Table 4.7: *Solve XOR problem (4 bits) using BP, RPROP, **and OBP***

η	BP 4-bit	RPROP 4-bit	OBP 4-bit
0.1	60245	128	56
0.2	29922	120	45
0.3	20561	110	33
0.4	16168	103	27
0.5	12818	98	24
0.6	10540	83	16
0.7	9084	75	13
0.8	7213	66	9
0.9	7188	64	8

As seen in the previous table, the OBP algorithm needs less number of epochs in comparing with BP and RPROP especially when using a large value for learning rate. With respect to the number of the required parameters by each algorithm, that is obvious from table 4.6. The RPROP needs seven parameters, while the OBP needs only four parameters like the BP (just in term number of parameters).

4.2.2 Solve XOR problem (4 bits) using BPM and OBPM

Through this experiment the BPM has been tested. Many tests have been done with the same values of the learning rate and the momentum (i.e. the learning rate and momentum have been used equally and increasingly from 0.1 to 0.9 as in table 4.8).

Table 4.8: *Solve XOR problem (4 bits) using BPM **and OBPM***

η	μ	BPM 4-bit	OBPM 4-bit
0.1	**0.1**	16880	33
0.2	**0.2**	15874	19
0.3	**0.3**	14325	11
0.4	**0.4**	13293	10
0.5	**0.5**	12286	7
0.6	**0.6**	11555	6
0.7	**0.7**	10279	5
0.8	**0.8**	10404	5
0.9	**0.9**	9588	4

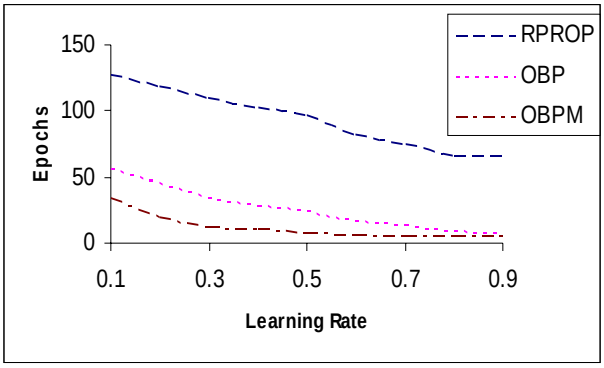
As seen in the previous table, if the large value of learning rate parallel with momentum were used, then obviously the small required number of epochs gradually will be.

From the previous table and as mentioned in sections; 4.1.2 and 4.1.3, the OBPM is faster than BPM. It has solved the XOR problem with a very few number of epochs, about 4 with learning rate equals to 0.9.

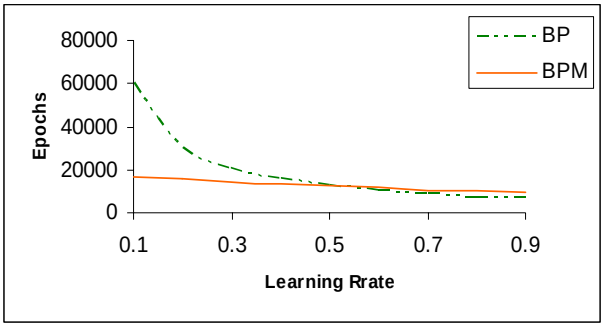
4.2.3 Compare five algorithms to solve XOR (4 bits)

Figure 4.5 summarizes the training process of the five algorithms to solve the XOR problem (4 bits). Figure 4.5(a) shows that the OBPM achieves the training process really faster than the OBP. It can see the huge differences in the number of epochs between these two algorithms and the RPROP.

Figure 4.5 (b) shows the performance of the BPM on the BP. Taking into consideration, when the large value for learning rate with momentum factor are used, then there is no positive affect on the speed of the training process, despite the importance of the momentum factor to escape from local minima.



(a)



(b)

Figure 4.5: *Compare five algorithms to solve XOR (4 bits)* ***(a)*** *Solve XOR (4 bits) using OBPM, OBP and RPROP* ***(b)*** *Solve XOR (4 bits) using BPM and BP*

4.3 Encoder Problem

Encoder problem is a feed-forward neural network with *N* input units, *M* hidden unites, and *N* output units (i.e. *N-M-N* networks). Training these networks can be very challenging when *M*<*N* [19]. They are trained so their output values (approximately) match their input values on a training set.

Peter Anderson [1] proposed a new approach to train encoder feed-forward neural networks and apply it on many classes of problems such as *N-4-N*. Anderson's approach is to initially train the network with a related, relatively easy-to-learn problem, and then gradually replace the training set with harder problems, until the network learns the problem he originally intended to solve. In several cases, this method allowed us to train networks that otherwise fail to train. Table 4.9 shows the parameters to solve the Encoder problem using the following algorithms: BP and OBP respectively.

Table 4.9: *The parameters to solve the Encoder problem*

Parameters	BP	OBP
Initial Weights	-0.1 ~ +0.1	-0.1 ~ +0.1
Learning rate η	0.05 ~ 0.95	0.05 ~ 0.95
Epoch Limit	50000	25000
MSE	10^{-4}	10^{-4}

4.3.1 16-M-16 Encoder Problem

A 16-*M-16* problem is to compress a signal of 16 values into one of *M* values. This network is useful for feature extraction. Each network consists of 16 units in both the input layer and output layer, and a hidden layer of *M* (6 to 11) units. In this experiment, the input patterns are equal to the target patterns such as the following sample pattern formats: Note, the size of gab between two ones ≥6.

Table 4.10: Five of the forty training patterns for 16-M-16 network

1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0

1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0

1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0

The following table shows the training process of the network 16-M-16, various values of *M* have been applied from 6 to 11. It is clear that the more number of units in the hidden layer, causes the network will be able to escape from local minima, so acceleration the training process.

Table 4.11: *Solve 16-M-16 Encoder using OBP with Training Wheel's technique*

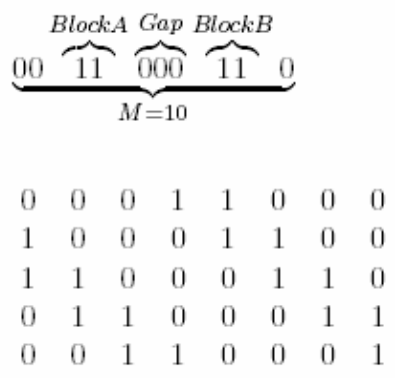
η \ M	11	10	9	8	7	6
0.1	125	145	181	∞	212	332
0.2	107	126	∞	171	204	224
0.3	97	111	∞	121	97	122
0.4	79	92	100	98	124	∞
0.5	65	∞	84	97	∞	155
0.6	62	62	64	74	97	∞
0.7	56	54	57	∞	88	212
0.8	51	50	59	58	78	97
0.9	64	62	58	71	75	∞

From the previous table it can notice that the best values achieved when a hidden layer size equals to 11. Meaning that the large hidden layer size helps to generalize and accelerate the training process.

4.3.2 M-4-M Encoder Problem (Double dots)

This experiment has tested *M-4-M* network to compress a signal of *M* vales into 4 values where *M*=7, 9, 10, 16, and 28 [1]. Each network consists of *M* units in both the input layer and output layer, and a hidden layer of 4 units. In this experiment, the input patterns are equal to the target patterns such as the following sample pattern formats:

Table 4.12: Five of the ten training patterns for 10-4-10 network where gab size = 3 and block size =2



This experiment has examined M-4-M through Training Wheel's technique. There are more than one architecture were used. Table 4.13 shows that there are many cases have been tested (for example, in the third column the architecture in this experiment is 9-4-9, the gap between each two ones is 2 and the size of each block is 1).

Table 4.13: *Solve M-4-M Encoder using OBP with Training Wheel's technique*

N	7	9	10		16			28
Gap	1	2	3		5			10
Block Size / η	1	1	2	1	3	2	1	4
0.1	57	180	45	110	97	1223	174	309
0.2	56	181	32	89	68	1082	122	216
0.3	48	113	22	82	64	692	96	203
0.4	41	101	26	75	61	570	124	120
0.5	33	97	28	67	56	452	111	108
0.6	42	95	27	56	57	413	118	87
0.7	44	85	29	71	56	297	106	84
0.8	49	107	34	79	65	477	134	76
0.9	59	123	39	90	93	575	180	87

From the previous table, it can noticeable that the minimum number of epochs was when the architecture 10-4-10 and the value of the gap =2 and the block size is 3 with learning rate equals to 0.3. In addition, the number of epochs with all learning rates in this case is less than all other cases.

From the previous table, it can be realized that using a medium size of the hidden layer helps in accelerating the training process. Meaning that using a large size of the hidden layer may slow down the training process or may leads the network to overfitting or even falling into local minima.

To compare BP and OBP, the best results using OBP have been taken from table 4.13, while the best results for the same experiment using BP taken from [1]. Take into consideration that using the same parameters with each. Table 4.14 shows this comparison.

Table 4.14: *Comparing between OBP and BP to Solve M-4-M Encoder Problem with Training Wheel's technique*

M-4-M	Gap	Block	BP	OBP
7	1	1	3669	33
9	2	1	3598	85
10	3	2	4504	22
		1	4653	56
16	5	3	3319	56
		2	3635	297
		1	4640	96
28	10	4	185998	76

It can see from the previous table that there are great differences between OBP and BP, which surely approaches towards the OBP with respect to the convergence time. In addition to that, the ability of the OBP to deal with large size of gap and block (e.g. using a gap=10, block =4 with the network architecture of 28-4-28).

4.4. Optical Character Recognition (OCR) Problem

In this section, the following algorithms; OBP, OBPM, BP, and BPM are implemented. In addition, five neural networks are developed and trained to recognize handwritten characters. Two algorithms were used, classical *BP* and *OBP* for the first two networks, while only OBP were used to test the last three networks because the classical algorithm takes too long time to trains and the comparative is insufficient.

4.4.1 48-8-4 Neural Network to Solve OCR Problem

4.4.1.1 Methodology

In addition, to study the performance of the *OBP* and *BP*, the neural networks were applied on handwritten English character recognition. Binary images of the characters were represented with bipolar values [-1, 1], and given to the input layer. As shown in figure 4.5, ten images are used in the training set an *8X6* binary image for the neural network with 48-8-4 architecture. The three-layered neural network is implemented with different learning parameters.

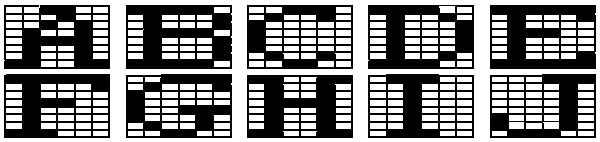


Figure 4.5: *The Training set of OCR (48-8-4 neural Network)*

Binary vectors of size 4 represent the output values. As shown in table 6.32, character *A* is represented with zero, while *J* is represented with 9. Each character image is mapped to its corresponding binary code. To train the network for larger set of characters a large output vector can be used (as will be in sections 4.4.5.2 and 4.4.5.3).The size of the hidden layer for this network is 8 producing *48-8-4* network. Small values for the learning rates were used to avoid local minima, the value ranges from 0.1 to 0.4. In addition, several sets with different random initial weights between –0.5 to +0.5 were used in each training process with various values for the learning rate. The training processes were terminated when the *MSE* (Mean Square Error) is less than or equal to 0.0001.

4.4.1.2 Solve OCR problem (48-8-4) Using BP and OBP:

Table 4.15 shows training summary for the *48-8-4* neural network using the *OBP* and *BP* with different values of the learning rate

Table 4.15: *Solve OCR problem (48-8-4) using OBP and BP*

η	OBP	BP
0.1	359	53188
0.2	226	32222
0.3	213	18063
0.4	155	13354

From the previous table, it can see the great difference between the two algorithms which greatly approaches towards OBP

4.4.1.3 Solve OCR problem (48-8-4) Using BPM and OBPM with Biases and without Biases:

This experiment refers to the training of multilayer feedforward neural networks with the architecture which described in the previous section, to recognize 8x6 pixel handwritten character from *A* to *J*. The network has 48 input units, 8 hidden units, and 4 output units, which represent the output of each character in binary code (e.g. output for A is 0000, and J is 1001).

The convergence criterion was set at 0.0001 MSE. 20 experiment runs of each learning rate were made with random initial weights setting over the interval [-0.5, 0.5]. All experiments was reached to MSE less than or equal to 0.0001.

The proposed algorithm OBP-M and BPM were tested in optical character recognition problem (OCR) with two networks .The first experiment refers to the training of network consists of 48-8-4 without biases and using Binary inputs [1, 0] instead of Bipolar inputs [-1, 1] and constant momentum was set 0.3, while the second network has the same architecture but with biases for hidden and output units (i.e. uses Bipolar input), and 0.1 for momentum factor.

This network was trained without momentum factor in [17]. Table 4.16 shows summary of parameters to solve the OCR problem using the following algorithms: BPM and OBPM respectively.

Table 4.16: *Parameters to solve the OCR problem (48-8-4) using OBPM and BPM*

Parameters	BPM	OBPM
Initial Weights	-0.5 ~ +0.5	-0.5 ~ +0.5
Learning rate η	0.1 ~ 0.4	0.1 ~ 0.4
Epoch Limit	25000	5000
MSE	10^{-4}	10^{-4}
Momentum μ	0.1, 0.3	0.1 , 0.3

4.4.1.3.1 Solve OCR problem (48-8-4) Using BPM and OBPM without Biases:

Table 4.17 shows the comparison results of standard BPM algorithm and proposed OBP-M algorithm using different learning rates. The first column represents the learning rate. Experiment results for the 48-8-4 OCR without biases for the two algorithms, and bipolar input values were used [-1, 1]. Max, Min and Avg. are the maximum, minimum and the average of number of epochs, respectively. S.D. is the standard deviation of the number of epochs. Remember that the number of trials is 20 with each learning rate.

Table 4.17: *Solve OCR problem (48-8-4) without biases Using OBPM and BPM*

η	Algo.	Max	Min	Avg	S.D.
0.1	BPM	20647	2283	7454.2	7481.4
	OPBM	63	11	22.8	16.3
0.2	BPM	10252	1145	3252.8	3276.3
	OPBM	28	6	11.6	6.7
0.3	BPM	4940	764	1639.43	1334.5
	OPBM	18	4	10.05	8.2
0.4	BPM	2548	576	1063.7	819.7
	OPBM	9	3	6.1	1.8

From the previous table it can see that the high speed of OBP in all training processes with the different values of the learning rate. The minimum number of required epochs with OBP was 3, when learning rate equals to 0.4, while in BP was 576.

4.4.1.3.2 Solve OCR problem (48-8-4) Using BPM and OBPM with Biases:

Table 4.18 shows the comparison results of standard BPM algorithm and proposed OBPM algorithm using different learning rates. Experiment results for the 48-8-4 OCR with; biases for the two algorithms, momentum factor equal to 0.1, and binary input values were used [0, 1]. Max, Min and Avg. are the maximum, minimum and the average of number of epochs, respectively. S.D. is the standard deviation of the number of epochs. Remember that the number of trials is 20 with each learning rate.

Table 4.18: *Solve OCR problem (48-8-4) with biases Using OBPM and BPM*

η	Algo.	Max	Min	Avg	S.D.
0.1	BPM	69006	52704	62121.7	6507.1
	OPBM	423	380	400.5	22
0.2	BPM	77170	25330	32862.3	12754.1
	OPBM	327	195	349.8	244.5
0.3	BPM	40328	17006	21108.75	5900.9
	OPBM	355	145	202.67	156.1
0.4	BPM	22024	11143	15766.3	3102.5
	OPBM	320	107	257.2	211.2

The aim from this experiment was to concentrate on how much using biases can affect on the proposed algorithm OBPM. From table 4.18, it can see that the effectiveness of the OBPM did not change although the number of epochs has increased, but these have been accompanied with the rise number of epochs using the BPM algorithm. That means that OBPM has the ability to deal with the biases. The effect of using a large learning rate has appeared which accelerates the training process. Through the OBPM a minimum number of the epochs reached to 107 while in BPM was 11143 and that was with learning rate of 0.4.

4.4.2 96-16-4 Neural Network to Solve OCR Problem

From this experiment a larger network has been used by using a larger size for each image which each one of them represents one of the capital letters from A-J. Each character represents with 12 rows and 8 columns which means that the number of pixels which represent each character is 96; this number represents the size of the input layer. 16 units were used to the size of the hidden layer, and the size of the output layer is 4 for all experiments.

In this experiment, the network was trained several times until reached to MSE equals to 0.001 through several learning rates as in table 4.19. The aim of this test is to compare the algorithms OBP and BP through a network that has a larger architecture from those that were used in the previous section.

Table 4.19: *Solve OCR problem (96-16-4) Using OBP and BP*

η	OBP	BP
0.1	83	4702
0.2	31	2479
0.3	25	1744
0.4	23	1191

From the previous table, it can see that OBP is still keeping its qualities in speeding up the training process especially with using a larger learning rate. The focus should always be through the number of epochs that gets from each algorithm.

4.4.3 Solve OCR problem (400-L-4) Using OBP

In this test, the architecture of the network was 400-N-4, where N=20, 40 and 60. The aim of this network is to recognize the characters from A-J. Each letter have been represented by 20 rows and 20 columns which makes the input layer size equals to 400 (the inputs are binary with biases). The aim of this test is to know the effect of the size of the input layer on the performance of the proposed algorithm OBP, and the effect of maximizing the hidden layer size on speeding up training using this algorithm. The following table shows the results of this test (note that MSE=0.0001).

Table 4.20: *Solve OCR problem (400-L-4) Using OBP*

η / HLS	0.1	0.2	0.3	0.4
20	142	75	52	43
40	109	53	42	43
60	56	46	39	28

HLS: refers to the Hidden Layer size.

It is clear that the minimum number of epochs was when the size of hidden layer equals to 60 especially when use larger learning rate which indicates that maximizing the size of the hidden layer and the learning rate helps in speeding up the training process. The size of the hidden layer may be maximizing just to a certain size [30], [7], that is because the network could not generalize when used larger than 60 units for the hidden layer size.

4.4.4. Solve OCR problem (900-L-4) Using OBP

This network is like other networks were built to recognize the characters from A-J. But here, each character have been represented by 30 rows and 30 columns, which makes the size of the input layer equals to 900 units (The inputs are binary with biases). Various values for the size of hidden layer have been used such as 25, 50, 75 and 100 units. The training process for this network has given the following results (Notice that MSE=0.0001):

Table 4.21: *Solve OCR problem (900-L-4) Using OBP*

HLS \ η	0.1	0.2	0.3	0.4
25	115	82	52	40
50	69	43	34	34
75	51	31	30	29
100	44	29	28	25

This experiment has also assured that OBP has the ability to generalize even with maximizing the input layer size. From figure 6.26 we notice that the hidden layer size maximized until reaches to a point where more maximization has no positive effect (may be overfitting). It even might expose the network to a prevention from generalize or memorization (for example, it can see from the previous table that the size of hidden layer equals to 75 and 100 leads to the same performance approximately).

4.4.5. Solve OCR Problem (10000-L-M neural network)

From the previous experiments, different sizes have been examined for input layer, but in this test the size of the input layer has been maximized to be compatible with real problems. Each character has been represented with 100 rows and 100 columns, so that the size of the input layer becomes very large which is 10000 units. The inputs of this network are binary with biases. The training for each network proceeded till the MSE becomes 0.0001 through different learning rates.

4.4.5.1 Solve OCR problem (10000-L-4) Using OBP:

This network has been constructed to recognize the characters from A-J with different sizes of hidden layer 40, 50, and 60 units with many learning rates. The next table represents the results of this network:

Table 4.22: *Solve OCR problem (10000-L-4) Using OBP*

HLS \ η	0.1	0.2	0.3	0.4
40	98	62	44	40
50	66	50	41	23
60	48	39	27	25

The row that represents the hidden layer size equals to 60 is the best row but it should say that using hidden layer size greater than 60 will prevent the network from generalization. So using a medium hidden layer size is better for generalization and acceleration of the training process.

4.4.5.2 Solve OCR problem (10000-L-5) Using OBP:

The new thing about this experiment is maximizing the output layer and that is to help this network to recognize the charters from A-Z (for example, the target output for the character A is 00000, and for Z is 11001). The number of units (N) in the hidden layer was 30, 40, 50, 60, 70 and 80 units. The following table summarizes the results of this experiment. The number of epochs that have been put in the table gave a better result from the four trails that have been got with each hidden layer using a certain learning rate.

Table 4.23: *Solve OCR problem (10000-L-5) Using OBP*

HLS \ η	0.1	0.2	0.3	0.4
30	151	72	70	55
40	112	72	60	37
50	85	55	38	33
60	81	38	32	29
70	60	31	27	27
80	54	25	24	23

From the previous, it can notice that OBP could solve this problem too and through the different sizes of hidden layer. For this test and the previous tests using hidden layer size of 60 and 70 is better because it is close to the results of hidden layer size equals to 80, but it reduces from falling into local minima or overfitting.

4.4.5.3 Solve OCR problem (10000-L-6) Using OBP:

The aim of this network is recognizing a large number of symbols. The size of the output layer has been maximized to 6. So this network has the ability to recognize the capital letters from A-Z, small letters from a-z beside the digits from 0-9. According to the size of the hidden layer it was 80, 90 and 100 units as in the following table:

Table 4.24: *Solve OCR problem (10000-L-6) Using OBP*

HLS \ η	0.1	0.2	0.3
80	74	67	42
90	51	51	39
100	33	27	22

From the results of the previous table, the new result is that when the size of the training set (i.e. number of symbols) increases it becomes very important to use a larger size of the hidden layer. In addition, the required number of epochs becomes less.

Note: w have not concentrated on the comparison between OBP and BP in the last few experiments. That is because the comparison is not logical and the training process needs a long time with BP. In addition, the great difference between these algorithms especially when dealing with large complicated problems; the concentrated was only on OBP and its ability to solve many problems and because the comparison has been done early between the two algorithms and the OBP executed on all. To clarify that, we take these two examples:

Example 1: To train the network that have the architecture 900-100-4 with learning rate=0.4 to reach MSE =0.0001 that is to recognize characters from A-J. This network is

considered kind as small network, we got the following results:

- OBP needed 25 epochs.
- BP needed 1665 epochs

Example 2: To train the network that have the architecture 10000-30-5 (that is to recognize characters from A-Z) with learning rate 0.1 with the same conditions of the training process in the first example we got the following results:

- OBP needed 151 epochs.
- BP needed 15600 epochs and more than 20 hours training on Dell machine (Pentium IV).

5. Conclusion

This paper introduced a three algorithms: OBP, OBPM and EVDBD respectively which has been proposed for the training of multilayer neural networks. The proposed algorithms provide stable learning, robustness to oscillations, and improved convergence rate.

An OBP is enhanced the version of the Backpropagation BP algorithm. The study shows that OBP is beneficial in speeding up the learning process, because it can adapts all weights with optical time. A modified error signal function would enhance the whole process in terms of learning speed, convergence. Comparing OBP approach with the classical one, somebody, will notice that the enhancement is, really, noticeable. The new approach has been applied on: (different neural network architectures, different input vectors (bipolar values without biases, binary with biases), different momentum factor, and different learning rate) and it has shown tangible improvements. The experiment results confirmed these observations.

The use of a momentum term with OBP accelerates the BP training. The algorithm was superior to other methods in terms of requiring less number of epochs to converge as well as less parameters are used (i.e. it does not uses many parameters which problem dependent like many algorithms mentioned above).

Appendix A

Notations

The notation described below were used throughout section 2.1 and section 3.1

X_{pi} net input to the *i*th input unit

net^{h}_{pj} net input to the *j*th hidden unit

W^{h}_{ji} weight on the connection from the *i*th input unit to *j*th hidden unit .

i_{pj} net input to the *j*th hidden unit

net^{o}_{pk} net input to the *k*th output unit

W^{o}_{kj} weight on the connection from the *j*th hidden unit to *k*th output unit .

O_{pk} actual output for the *k*th output unit .

Y_{pk} desired output for the *k*th output unit .

f (Sigmoid) activation function

f' derivative of activation function

δ^{o}_{pk} signal error term for the *k*th output unit.

δ^{h}_{pj} signal error term for the *j*th hidden unit.

η learning rate

References

[1] Anderson G. Peter, *Training Wheels for Encoder Networks*, Proc. of the Int. ICSC Symposium on Intelligent Industrial Automation(IIA 96) and Soft Computing (SOCO 96), Reading, U.K. , Academic Press.1996.

[2] Carling, A., Alison, Back Propagation. *Introducing Neural Networks*, 1992, 133-154.

[3] Fahlman, S. E., Faster-learning variations on Backpropagation: an empirical study. Proceedings of the 1988 Connectionist Models Summer School, 38-51.

[4] Freeman, J. A., Skapura, D. M., Backpropagation. *Neural Networks Algorithm Applications and Programming Techniques*, 1992, 89-125.

[5] F. M. Silva and L. B. Almeida,"Speeding up backpropagation", in Advanced Neural Computers, 1990,151-158.

[6] Jacobs, R. A., Increased rates of convergence through learning rate adaptation, Neural Networks, 1988,1,169-180

[7] John K. Kruschke, Javier R. Movellan. *Benefits of gain: Speed learning and minimal hidden layers in backpropagation networks*. IEEE Trans on Systems, 21(1), 1991, 273-280.

[8] J. Leonard and M. A. Kramer, "Improvement of the backpropagation algorithm for training neural networks", Computer Chem. Engng., 1990, 14(3),337-341.

[9] Lau, Clifford G.Y., Neural networks: theoretical foundations and analysis. IEEE Press, 1992.

[10] Martin T. Hagan, Howard B. Demuth, Neural Networks Design, 1996, 11.1-12.52

[11] Maureen Caudill, and Charles Butler, Understanding Neural Networks: Computer Explorations, Volume 1,1993,.155-218

[12] Minai, A.A., Williams, R.D., Acceleration of back-propagation through learning rate momentum adaptation, Proceedings of the International Joint Conference on Neural Networks, 1990, 1676-1679.

[13] M.A. Otair and W.A. Salameh, An Improved Back-Propagation Neural Networks using a Modified Non-linear Function, Proceedings of the IASTED International Conference, 2004,442-447

[14] M.A. Otair and W.A. Salameh, Optical Back-Propagation Neural Networks with a Momentum Factor –A Case Study-, Accepted for publication in WSEAS Transactions, June 12, 2004

[15] M.A. Otair and W.A. Salameh, *Enhanced Version of Delta-Bar-Delta*, under preparation, 2004.

[16] M.A. Otair and W.A. Salameh, *Solving The Exclusive-OR Problem Using An Optical Backpropagation Algorithm*, under preparation, 2004.

[17] M.A. Otair and W.A. Salameh, *Online Handwritten Character Recognition Using An Optical Backpropagation Neural Networks*, Proceedings of The 2004 International Research Conference on Innovations in Information Technology, 2004, 334-341.

[18] M. Hagiwara, "Theoretical derivation of momentum term in backpropagation", in Int. Joint Conf. On Neural Networks, 1992,682-686.

[19] Paul Bakker , Steven Philps, Janet Wiles, *The N-2-N encoder: a matter of representation* , Proc. of The Int. Conf. On Artificial Neural Networks, Amsterdam, The Netherlands, 1993.

[20] Riedmiller, M and Braun, H., A direct adaptive method for faster backpropagation learning the PROP algorithm. Proceedings of the IEEE international conference on Neural Networks (ICNN), Vol. I, San Francisco, CA, ,1993, 586-591.

[21] Robert Callan, The Essence of Neural Networks, Southmpton Institute, 1999, 33-52.

[22] Rumelhart, D. E., Hinton, G. E., and Williams, R. J., Learning internal representations by error propagation, In D. E. Rumelhart and J. L. McClelland (eds) *Parallel Distributed Processing*, 1986, 318-362.

[23] Rumelhart, D. E., Richard Durbin, Richard Golden, and Yves Chauvin, Backpropagation: theoretical foundations. In Y.Chauvin and D. E Rumelhart (eds). *Backpropagation and Connectionist Theory*. Lawrence Erlbaum, 1992.

[24] Simon Hakin , Neural Networks A Comprehensive Foundation ,2nd Edition , 1999 ,161-184

[25] T. M. Mitchell, "Machine Learning (McGraw Hill, Boston, MA), 1997,p.119

[26] T. Tollenaere, "SuperSAB: Fast adaptive backpropagation with good scaling properties", Neural Networks, 1990,3,561-573.

[27] W. Schiffmann, M. Joost and R Werner, " Comparison of optimized backprop algorithms", Artificial Neural Networks European Symposium, 1993.

[28] X. H. Yu and G.A. Chen, "Efficient estimation of dynamically optimal learning rate and momentum for backpropagation learning", in Proc. of IEEE ICNN-95, 1995,385-88.

[29] Y Dali and L Zemin,"A LMS algorithm with stochastic momentum factor", in Proc. of ISCAS-93, 1993, 1250-1253.

[30] Yve Chuvin. *A backpropagation algorithm with optimal use of hidden units*. In Touretzky, 1989 , 519-526.