# Improvement the Back-propagation Technique

**Nidhal H. AL-Assady  Baydaa I. Khaleel  Shahbaa I.Khaleel**
shahbaaibrkh@uomosul.edu.iq
*Dept. of Computer Science*
*College of Computer Sciences and Mathematics*
*University of Mosul, Iraq*

**Received on: 28/09/2002  Accepted on: 15/03/2003**

**Abstract**

Error backpropagation neural network (EBP) used training algorithm for feedforward artificial neural networks (FFANNs). The main problem with the EBP algorithm that it is very slow and the converge to the optimal solution is not guaranteed. This problem leads to search for improvements to speed up this algorithm. In this research we use several methods to speed up the EBP algorithm. A many layer neural network was designed for building pattern compression system, encoding and recognition. We also used many methods to speed up this algorithm (EBP) and comparison between them.

**Keywords:** Artificial neural networks, backpropagation, image recognition, pattern compression.

# تحسين تقنية انتشار الخطأ خلفاً

**نضال حسين الاسدي  بيداء ابراهيم خليل  شهباء ابراهيم خليل**
*كلية علوم الحاسوب والرياضيات*
*جامعة الموصل*

**تاريخ استلام البحث: 2002/09/28  تاريخ قبول البحث: 2003/03/15**

**الملخص**

تستخدم خوارزمية انتشار الخطأ خلفاً في الشبكات العصبية الاصطناعية ذات التغذية الأمامية. والمشكلة الموجودة في خوارزمية الشبكة العصبية ذات الانتشار العكسي هي أن بطء الاقتراب والوصول إلى الحل الأمثل ليس أكيدا،وقد أدت هذه المشكلة إلى البحث عن تحسينات لتسريع هذه الخوارزمية. استخدم في هذا البحث العديد من الطرائق لتسريع خوارزمية الشبكة العصبية ذات الانتشار العكسي للتوصل إلى احسن خوارزمية لتسريع عمل الشبكة وتم تصميم شبكات عصبية ذات انتشار عكسي ومحتوية على عدد من الطبقات لتمييز النماذج وتنفيذ جميع الطرق المحسنة للشبكة على هذه التطبيقات وإجراء المقارنة بينها.

**الكلمات المفتاحية:** الشبكات العصبية الاصطناعية ،شبكة انتشار الخطأ خلفا ، تمييز الصور، كبس النماذج.

## INTRODUCTION

The EBP learning rule for multilayer FFANNs known as the backpropagation algorithm, is generalization of the delta learning rule for single layer artificial neural networks. EBP is now the most popular learning algorithm for multilayer FFANNs because of its simplicity, its power to extract useful information from examples, and its capability of storing information implicitly in the connecting links in the form of weights.

The original version of the EBP learning algorithm has been of great concern to practical users for many reasons: it is extremely slow if it does converge ,it may get stuck in local minima bsefore learning all the examples, it is sensitive to initial conditions , it may start oscillating, and so on. Several methods have been proposed to improve the performance of the EBP algorithm. [9]

The most important suggested modifications to the original EBP algorithm are presented in this research.

## 1. An Overview of Training

The objective of training network is to adjust the weights. The input–output sets can be referred to as vectors. Training assumes that each input vector is paired with a target vector representing the desired output; together these are called a training pair. Usually, a network is trained over a number of training pairs. This group of training pairs is called a training set. Before starting the training process, all of the weights must be initialized to small random numbers. This ensures that the network is not saturated by large values of the weights, and prevents certain other pattologies. For example, if the weights all start at equal values and the desired performance requires unequal values, the network will not learn.

Each training pattern is propagated forward layer by layer until an output pattern is computed. This output is then compared to a desired or target output and an error value is determined. The errors are used as inputs to feedback connections from which adjustments are made to the synaptic weights layer by layer in a backward direction. The backward linkages are used only for the learning phases, as where the forward connections are used for both the learning and operational phases. After training is stopped, the performance of the network is tested [10]

**Error Backpropagation Training Algorithm**

The training process of a FFANN is an iterative process. Each iteration consists of the following steps: -

Step 1: Initialize all the network weights W to small random values.

Step 2: Select the next training pair from the training set [$X_p$ , $T_p$] (input, target), and compute in a forward direction the output values for each unit j of each layer L, thus

$$\boldsymbol{net}_{pj}^{L+1} = \sum_{i=1}^{n^L} \mathbf{w}_{ij}^{L}\, \mathbf{out}_{i}^{L} + \mathbf{bias}_{j}^{L+1} \qquad (1)$$

$$\boldsymbol{out}_{pj}^{L+1} = \mathbf{f}(\mathbf{net}_{pj}^{L+1}) = \frac{\mathbf{1}}{\mathbf{1}+\mathbf{e}^{-\beta\, \mathbf{net}_{pj}^{L+1}}} \qquad (2)$$

Where $\boldsymbol{net}_{pj}^{L+1}$ is the weighted sum of inputs to unit j , and $\boldsymbol{out}_{pj}^{L+1}$ is the output of jth unit in layer L+1, p is the pattern ,bias is bias unit.

Note that the inputs to layer one (input layer) are indexed with subscript 0, and hence, $\boldsymbol{out}_{pj}^{0} = x_j$ (i .e $x_j$ represent the input of the pattern).

Step 3: Calculate the error between the actual output of the network ($out_{pj}$) and the desired output (t) the target vector from the training pair, and then, use the values $\boldsymbol{out}_{pj}^{o}$ computed by the

final layer units and the corresponding values $t_{pj}$ to compute the delta quantities (δ).

$$\delta^{o}_{pj} = (\mathbf{t_{pj}} - \mathbf{out^{o}_{pj}})\, \mathbf{f'(net^{o}_{pj})} \qquad (3)$$

For all j using pattern p.

Where $t_{pj}$ is the target value for unit j , $out_{pj}$ is the output value for unit j , $\boldsymbol{f'(net_{pj})}$ is the derivative of the sigmoid function (f(net)).

Step 4: Compute the deltas (δ) for each of the proceeding layers (hidden layers) by backpropagating the errors using:

$$\delta^{L+1}_{pi} = \mathbf{f'(net^{L+1}_{pi})} \left[ \sum_{j=1}^{m^{L+2}} \delta^{L+2}_{pj}\ \mathbf{w^{L+1}_{ij}} \right] \qquad (4)$$

Where m is the number of units in layer L .

Step 5: Adjust all weights $w_{ij}$ using

$$\boldsymbol{w^{new}_{ij}} = \mathbf{w^{old}_{ij}} + \Delta\, \mathbf{w^{L}_{ij}} \qquad (5)$$

$$\Delta\, \mathbf{w^{L}_{ij}} = \eta\, \delta^{L+1}_{pj}\ \mathbf{out^{L}_{pi}} \qquad (6)$$

Step 6: Return to step 2 and repeat for each pattern p in the training set until the total error has reached an acceptable value.

From the training steps, we can see that the backpropagation learning algorithm involves a forward propagation pass followed by a backward propagation pass. Both two passes are done for each pattern presentation during the training. [9]

**Forward Pass**

Step 2 can be expressed in vector form as follows an input vector X is applied and an output vector Y is produced. The input-target vector pair X and T comes from the training set. The calculation is performed on X to produce the output vector Y.

As we have seen, calculation in multilayer networks is done layer by layer, starting at the layer nearest to the inputs. The net value of each neuron in the first layer is calculated as the weighted sum of its neuron's inputs. The activation function *f* then "squashes"

net to produce the out value for each neuron in that layer. Once the set of outputs for a layer is found, it serves as an input to the next layer. The process is repeated, layer by layer, until the final set of network outputs is produced. [3] [9]

**2.2 Backward Pass (Reverse Pass)**

**Adjusting the Weights of the Output Layer**

Because a target value is available for each neuron in the output layer, adjusting the associated weights is easily accomplished using a modification of the delta rule. Interior layers are referred to as "hidden layers", whose outputs have no target values for comparison; hence, training is more complicated.

The EBP algorithm for FFANNs proceeds by representing an input pattern to the input (or the 0th) layer, after which the network produces an output. This output is compared to a desired or target output. The difference between the target output and actual network is called error.

Formally, the error $E_{pj}$ for the jth unit of the output layer O for the input training pair ($X_p$ , $T_p$) is computed as :

$$\boldsymbol{E_{pj} = t_{pj} - out^{o}_{pj}} \qquad (7)$$

An objective of a learning algorithm is to use this error to adjust the weights in such a way that the error gradually reduces. The training process stops when the error of every neuron for every training pair is reduced to an acceptable level, or when no further improvement is obtained. In the latter case, the network is a gain initializes with small weights and the training process starts a fresh.

Figure (1) shows the training process for a single weight from unit i in the hidden layer L+1 to unit j in the output layer L. The output of a unit in layer L+2 is subtracted from its target value to produce an error signal as shown above. This is multiplied by the derivative of the squashing $\boldsymbol{[out^{o}_{pj} (1 - out^{o}_{pj})]}$

calculated for that layer's neuron L, there by producing the δ value: $\delta = \mathbf{out}^{o}_{pj}(1 - \mathbf{out}^{o}_{pj})(\mathbf{t}_{pj} - \mathbf{out}^{o}_{pj})$

Then δ is multiplied by out from a neuron I, the source neuron for the weight in question. This product is in turn multiplied by a training rate coefficient η (typically 0.01 to 1.0) and the result is added to the weight, an identical process is performed for each weight proceeding from a neuron in the hidden layer to a neuron in the output layer.

The following equations illustrate this calculation:

$$w_{ij}^{new} = \mathbf{w}_{ij}^{old} + \Delta \mathbf{w}_{ij}^{L+1}$$

$$\Delta \mathbf{w}_{ij}^{L+1} = \eta \, \delta_{pj}^{L+2} \, \mathbf{out}_{pi}^{L+1}$$

Where:

$w_{ij}^{old}$ is the value of a weight from neuron i in the hidden layer to a neuron j in the output layer (weight before adjustment).

$w_{ij}^{new}$ is the value of the weight after adjustment.

$\delta_{pj}^{L+2}$ is the value of δ for neuron j in the output layer L+2.

$out_{pi}^{L+1}$ is the value of out for neuron i in the hidden layer L+1.

Note that subscripts i and j refer to a specific neuron, as where subscripts L+2 and L+1 refer to a layer.[5][9]

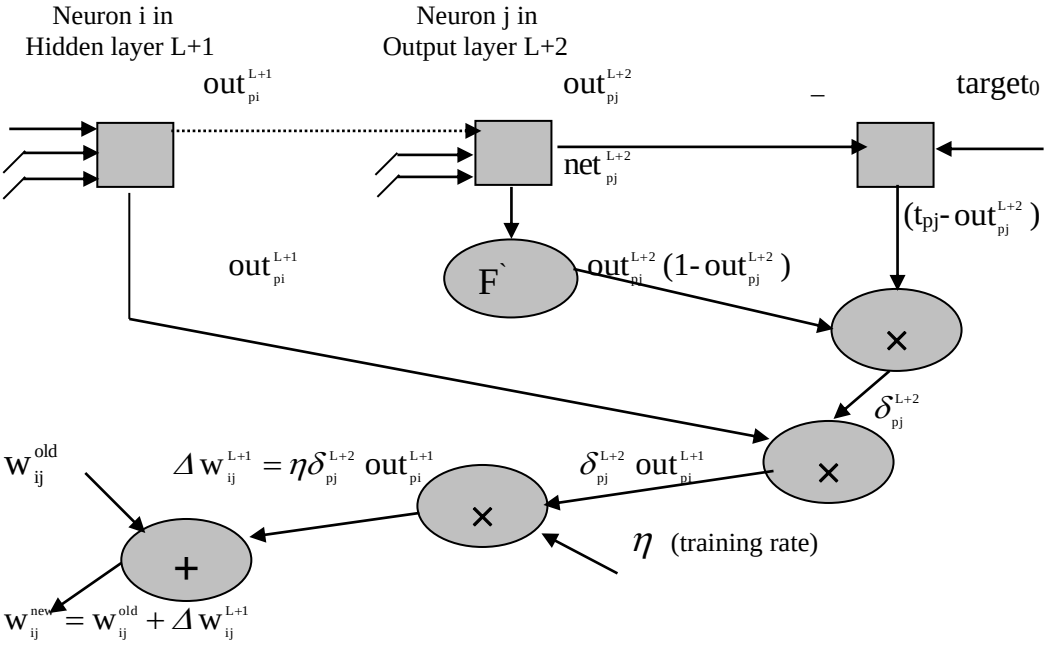



**Figure (1) Training the weight in the output layer**

### 2.2.2 Adjusting The Weights Of The Hidden Layers

Hidden layers have no target vector, so the training process described above cannot be used. This lack of a training target Stymied efforts to train multilayer networks until backpropagation provided a workable algorithm.

Backpropagation trains the hidden layers by propagating the output error back through the network layer by layer, adjusting weights at each layer. Equations 3 and 4 are used for all layers, both output and hidden; however, for hidden layers δ must be generated without benefit of a target vector. Figure (1) shows how this is accomplished. First, δ is calculated for each neuron in the output layers, as in equation (3). It is used to adjust the weights feeding into the output layers, then it is propagated back through the same weights to generate a value for δ for each neuron in the first hidden layer. These values of δ are used, in turn, to adjust the weights of this hidden layer and, in a similar way, they are propagated back to all preceding layers.

Consider a single neuron in the hidden layer just before the output layer. In the forward pass, this neuron propagates its output value to neurons, in the output layer through the inter connecting weights. During training these weights operate in reverse, passing the value of δ from the output layer back to the hidden layer. Each of these weights is multiplied by the δ value of the neuron to which it connects in the output layer. The value of δ needed for the hidden layer neuron is produced by summing all such products and multiplying by the derivative of the squashing function by using eq. (4) as shown here:

$$\delta_{pi}^{L+1} = \mathbf{f}\,(\mathbf{net}_{\mathbf{pj}}^{\mathbf{L+1}}) \left[ \sum_{j=1}^{m^{L+2}} \delta_{pj}^{L+2}\, \mathbf{w}_{\mathbf{ij}}^{\mathbf{L+1}} \right]$$

See figure (2) with δ in hand, shows that the weights feeding the first hidden layer can be adjusted using equations 3 and 4, modifying indices to indicate the correct layers.

For each neuron in a given hidden layer, δs must be calculated and all weights associated with that layer must be adjusted. This is repeated, moving back toward the input layer by layer, until all weights are adjusted.

For measuring the performance of the learning algorithm, an objective function is defined in such a way that as the error reduces so does the value of the objective. Thus a training algorithm decides the change of weights using some procedure that guarantees no increase in the objective functions value. The objective functions are known as the energy functions (named after similar functions in physics).

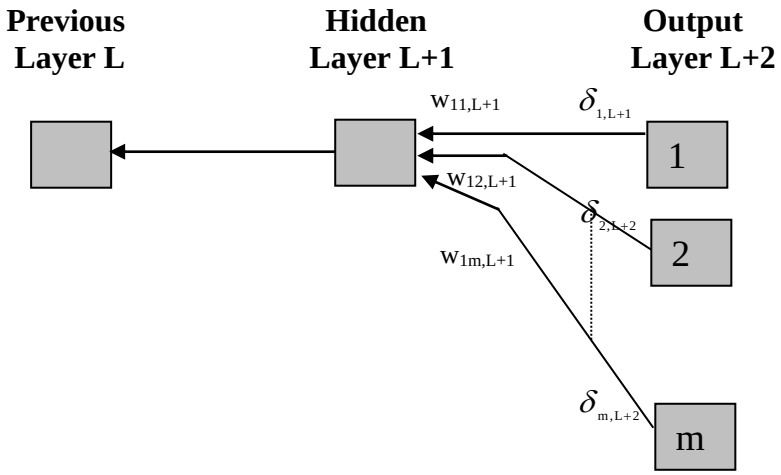



***Figure (2) Training a weight in a hidden layer***

In their original EBP algorithm used the sum of the squared error as the energy function.

$$E = \frac{1}{2}\sum_{p=1}^{p}\sum_{j=1}^{m^o}(E_{pj})^2 \qquad (8)$$

The energy function can be defined for only one training pattern pair ($X_p$,$T_p$) as: $E_p = \frac{1}{2}\sum_{j=1}^{m^o}(E_{pj})^2$ (9)

There are two versions of the EBP algorithm, online and batch. In the online EBP algorithm, the weights are updated using the error corresponding to every training pattern. This method uses energy function defined by equation (9). However, in the batch EBP algorithm, the weights are updated after accumulating errors corresponding to all input patterns, and thus making use of the energy function defined by equation (8). [4][9]

## 3. Using β Variable

The variable β can be used in the sigmoidal function during the backpropagation-training algorithm in order to determine the steepness of the shape of sigmoid function. When β is used in backpropagation training it lies, in the range [0.1 – 1.0], a lower value of β gives a smoother transition from a lower to a higher activation level as the $net_{pj}^{L}$ changes its value; a higher value for β will cause a step-like transition in the activation level. (i. e when β has a value of 0.1 learning has slow converge, but when the value of β is 1 the learning is faster. Here $f'(.)$ is the sigmoidal derivative which is calculated as :

$$f'(net_{pj}^{L}) = \beta\,(1 - out_{pj}^{L})\,out_{pj}^{L} \qquad (10)$$

The β variable is used in the standard backpropagation training algorithm.

## 4. Modification to Original EBP Algorithm

It has been realized that the original EBP algorithm is too slow for most practical applications, and many modifications have been suggested to speed it up. These Modifications on the original EBP algorithm that have been suggested to speed up the training of FFANNs are:

### Momentum Method (MOM)

The learning strategy used in the original EBP algorithm is actually a gradient descent on the multidimensional energy surface in the weight space. A closer look on the effect of the

value of the learning rate coefficient η reveals that in an area of the energy surface where the gradient is not changing sign, a large value of η reduces the energy function faster; on the other hand, near an area where the sign of the gradient is changing quickly, a smaller value of η keeps the descent a long the energy surface. This disparate need for the value of η suggests a method that a dapts the value of η dynamically depending on the characteristic of the energy surface. The momentum strategy implements such a variable learning rate coefficient implicitly by adding a fraction of the last weight change to the current direction of movement in the weight space, and it is a slight change to the weight updating rule of the original EBP .

When derived the backpropagation algorithm, the mode statement the learning implements a gradient descent on the error surface if the weight change is determined after complete presentation of all training patterns. In practice, we can follow this statement to update the weights for each cycle rather than for each pattern presentation.

Momentum method is enhancing the stability of the training process, and involves adding a term to the weight adjustment that is proportional to the amount of the previous weight change. Once an adjustment is made, it is remembered and serves to modify all subsequent weight adjustments. The adjustment equations are modified to the following:

$$\Delta \mathbf{w}_{\mathbf{ij}}^{\mathbf{L}}(\mathbf{s}+\mathbf{1}) = \eta\,(\delta_{pj}^{L+1}\ \mathbf{out}_{\mathbf{pi}}^{\mathbf{L}}) + \alpha\left[\Delta \mathbf{w}_{\mathbf{ij}}^{\mathbf{L}}(\mathbf{s})\right] \qquad (11)$$

$$w_{ij}^{L}(\mathbf{s}+\mathbf{1}) = \mathbf{w}_{\mathbf{ij}}^{\mathbf{L}}(\mathbf{s}) + \Delta \mathbf{w}_{\mathbf{ij}}^{\mathbf{L}}(\mathbf{s}+\mathbf{1}) \qquad (12)$$

Where α is a momentum coefficient that can have a value between zero and one, s is the cycle number.

A similar method based on exponential smoothing that may prove superior in some applications is .

$$\Delta \mathbf{w}_{\mathbf{ij}}^{\mathbf{L}}(\mathbf{s}+\mathbf{1}) = \alpha\,\Delta \mathbf{w}_{\mathbf{ij}}^{\mathbf{L}}(\mathbf{s}) + (\mathbf{1}-\alpha)\,\delta_{pj}^{L+1}\ \mathbf{out}_{\mathbf{pi}}^{\mathbf{L}} \qquad (13)$$

Then the weight change is computed:

$$w_{ij}^{L}(\mathbf{s}+\mathbf{1}) = \mathbf{w}_{\mathbf{ij}}^{\mathbf{L}}(\mathbf{s}) + \eta\,\Delta \mathbf{w}_{\mathbf{ij}}^{\mathbf{L}}(\mathbf{s}+\mathbf{1}) \qquad (14)$$

Where α is a smoothing coefficient in the range of 0.0 to 1.0. If α is 0.0 then the smoothing is minimum; the entire weight adjustment comes from the newly calculated change. If α is 1.0, the new adjustment is ignored and the previous one is repeated.

Between 0 and 1 is a region where the weight adjustment is smoothed by an a mount proportional to α. again η is the training rate coefficient, serving to adjust the size of the average weight change .[7][9]

**Bp with Expected Source Values Method (ESV)**

Bp is a gradient descent learning rule that minimizes the sum-squared error over the output layer of a feed forward neural network. Weights $w_{ij}$ from a unit (i) to a unit (j) are updated according to eq. (6) and $\delta_{pj}^{L}$ is an associated error term for unit (j), defined as eq. (3) for output units and according to eq. (4) for hidden units.

The backpropagation learning rule converges significantly faster if the expected values of source units are used for updating weights. The expected value of a unit can be approximated as the sum of the output of the unit and its error term .

The decrease in $\delta_j$ applied only for the current training input, say x, and it is in effect only as long as other network parameters remain unchanged. In particular, the effectiveness of the weight update will be reduced if, as learning proceeds, the output of unit i to x deviates from the $out_i$ value used for the weight update. If i is an input unit, x will always lead to this $out_i$ value for unit i, If i is a hidden unit, however, there are weights $w_{hi}$ that are also subject to modification. These modifications will affect the output of unit i to the input x. Thus the weight changes are dictated by eq. (6).

We can exploit the analogy between $\delta_i$ and the error attributed to unit i to compensate for future changes to weights in earlier layers. Instead of using the current value of the source unit i for update, we can use the expected value for unit i. The

expected value can be formulated simply as the sum of the current value and the error term. This leads to the following modified backpropagation learning rule:

$$\Delta w_{ij}^{L} = \eta ( out_{pi}^{L} + Beta\ \delta_{pi}^{L} )\ \delta_{pj}^{L+1} \qquad (15)$$

Where Beta is a constant. This eq. reverts to the original rule when Beta equals 0.0. Beta can usually be set to 1.0, and superior results over eq. (3) are consistently obtained. In many cases, however, higher values of Beta can further accelerate learning. As with $\eta$, further increases in Beta beyond some problem of specific value result in oscillations and non-convergence. [8]

**The Bold Driver Method (B_D)**

In the previous method we found that the momentum coefficient implicitly adjusts the effective learning rate coefficient dynamically, depending on the nature of the energy surface. The improvement in the learning time comes from this dynamic behavior of the effective learning coefficient. There are methods where the learning coefficient is explicitly adjusted to obtain an improved convergence speed, one of them is the bold driver method.

This method makes two trivial changes to the original EBP algorithm, it monitors the value of the energy function E given by equation (8), and it dynamically adjusts the value of the learning rate coefficient η.

The training starts with some arbitrary value of the learning rate coefficient η. If the value of E decreases, the learning rate is increased by a factor $\rho$ ($\rho$>1). This helps to take a longer step in the next iteration. The value of the learning rate also grows exponentially in a constant gradient region of the energy function. On the other hand, if the value of the energy function E increases, it is assumed that the last stepsize was too large and firstly the last weight correction to every weight is canceled. Secondly, the value of the learning rate coefficient is

decreased by a factor $\sigma$ ($\sigma$<1), and thirdly, a new trial is performed. If the new trial shows a reduction in the value of the energy function, the deceased learning rate coefficient is accepted as the next learning rate; otherwise, the learning rate coefficient is repeatedly reduced until it gives a step size that reduces the value of the energy function.

The bold driver method is non-local because it keeps only one learning rate coefficient for all the weights, whereas the momentum strategy adjusts the effective learning coefficient locally for each weight through the momentum term. This method controls the oscillation of the value of the energy function. [10]

**Self Adaptive Backpropagation Method (SAB)**

This method is a local acceleration strategy, and it is supported by the following observations: firstly every weight should have its own learning rate coefficient, because the partial derivative of the energy function E with respect to each weight gives the gradient for that weight only. Also, the learning rate coefficient for one weight need not be appropriate for other weights. Secondly, the learning rate coefficient should be allowed to vary depending on the nature of the surface of the energy function a long the dimension under consideration.

Thirdly, the learning rate coefficient for a weight should be increased if consecutive steps have the same sign. Fourthly, the learning rate coefficient should be small when the derivative of the energy function with respect to a weight changes sign.

Let the learning rate coefficient for the weight $w_{ij}^{L}$ at time step s be $\eta_{ij}^{L}$ **[s]**. The algorithm is as follows:

Step 1: Choose an initial learning rate coefficient η.

Step 2: Set the learning rate coefficient $\eta_{ij}^{L}$ **[0]** $=\eta$ for all weights in the neural network.

Step 3: Do a backpropagation step without momentum term.

Step 4: If $G_{ij}^{L}[s]$, the negative of the partial derivative of the energy function E, has the same sign,

set $\eta_{ij}^{L}[s+1]=\rho\,\eta_{ij}^{L}[s]$ (16)

For weight $w_{ij}^{L}[s]$; we used the increment factor $\rho$ >1 .

Step 5 : If $G_{ij}^{L}[s]$, changes sign then

Set $\eta_{ij}^{L}[s+1]=\eta$ (17)

weight $w_{ij}^{L}[s]$.

Estimate a good weight $w_{ij}^{L}[s+1]$ by interpolation(based on previous $\Delta w_{ij}^{L}$ values.

Do a number of backpropagation steps with momentum term.

Step 6: Restart the algorithm from step (3).

The SAB method functions mentally different from all other methods. In this method, one learning rate coefficient is kept for each weight.

This algorithm performs better than the original EBP, because it can adjust the learning rate coefficient over a wide range if the initial learning rate coefficient is small. One problem inherent in the original EBP algorithm, the selection of a good learning rate coefficient by the user, remains unchanged in this algorithm. In addition, this algorithm starts with the initial learning rate coefficient if a change in the sign of the gradient occurs. [7]

**Super Self Adaptive Backpropagation Method (S_SAB)**

This method is new and different from all other methods. Let $\rho$ be the factor by which the learning rate coefficient is increased, and let $\sigma$ be the factor by which the learning rate is decreased, if necessary. Experimental studies suggest that the learning rate coefficient decrease should be faster than the

increase, Recall that the learning rate coefficient at time steps for weight $w_{ij}^{L}$ is denoted by $\eta_{ij}^{L}[s]$. The algorithm is as follows:

Step 1: Choose an initial learning rate coefficient η.

Step 2: Set the learning rate coefficient $\eta_{ij}^{L}[0]=\eta$; for all weights in the neural network.

Step 3: Do a number of backpropagation steps with momentum term.

Step 4: If $G_{ij}^{L}[s]$, the negative of the partial derivative of the energy function E has the same sign, set $\eta_{ij}^{L}[s+1]=\rho\,\eta_{ij}^{L}[s]$ for weight $w_{ij}^{L}[s]$ in neural network; we assume the value of increment factor $\rho>1$.

Step 5: If $G_{ij}^{L}[s]$, changes sign then

Set $\eta_{ij}^{L}[s+1]=\sigma\,\eta_{ij}^{L}[s]$ (18)

For weight $w_{ij}^{L}[s]$ in the neural network; we assume the value of the decrement factor $\sigma<1$.

Undo the previous weight update; and

Set $\Delta w_{ij}^{L}[s+1]=0$;

Step 6: Restart the algorithm from step (3).

In the SAB learning method, the learning rate coefficient is set to the initial value, but in the S_SAB learning method the value of the learning rate coefficient is reduced by a constant factor.

This method speeds up learning considerably. Also, because the learning rate coefficient is never set to its initial value (as in SAB), the choice of initial learning rate coefficient would not have much influence on learning time and hence can be made arbitrarily.

## 5. The Applications of the BP Learning Algorithm

Backpropagation neural network has been designed for pattern recognition, encoding, compression, and also improvements of the standard backpropagation neural networks are applied in all the applications we said above. We described here the practical implementation of the BPNN in all application we used here, figure (3) shows an outline of this implementation.

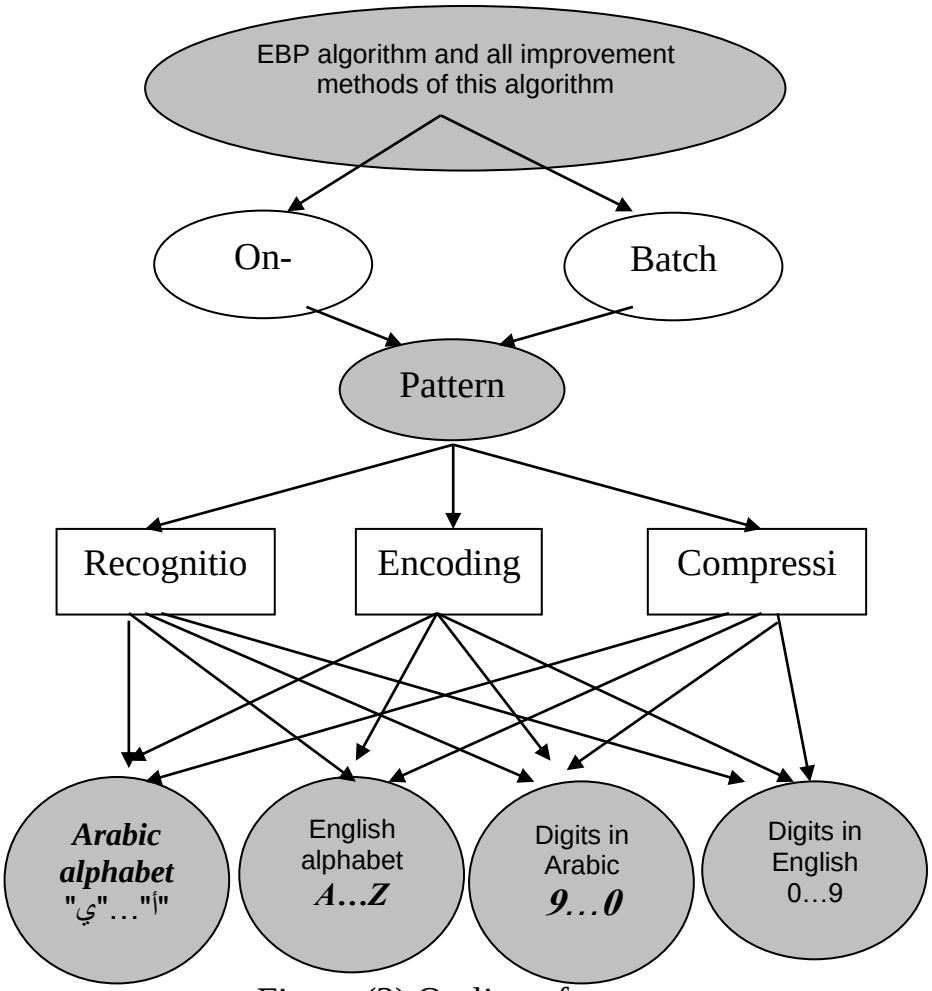



Figure (3) Outline of system

## 5.1 Multilayer Feed Forward Network for Pattern Recognition

In this research, a backpropagation neural network algorithm and all the methods that improved this algorithm are used to recognition uppercase letters in English "A"…"Z", letters in Arabic"ي"…"أ" , digits "0" … " 9" in English and digits "9"…"0" in Arabic, by using online and batch backpropagation algorithm. The representation of characters and digits in both languages, English and Arabic is shown in black and white pixels.

To recognize the uppercase letters in English, the letters "A"…"Z" are encoded as 35 binary-valued input vectors representing black and white pixels. and the output pattern has been a single output bit that has been set to 1 for each character pattern to be recognized (local representation). The total output neurons that has been required is 26 neurons. Figure (4) shows the process of character recognition (alphabetic in English).

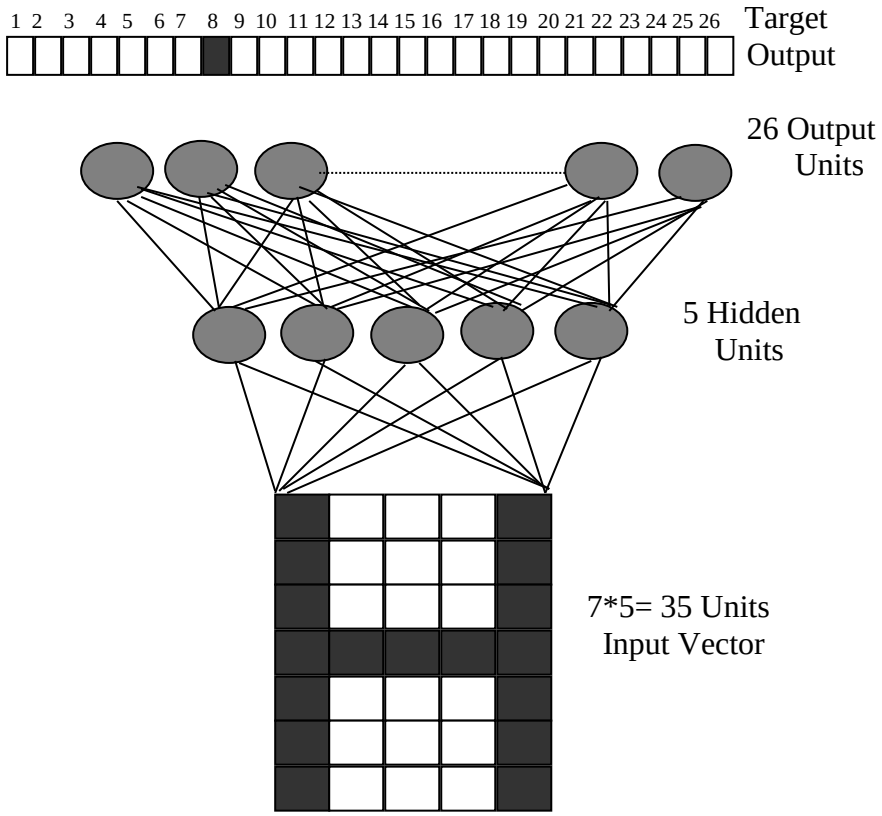



Figure (4) The Process of character recognition

When we recognize the digits "0" through "9" the input vector is represented by 35 binary value and 10 units in the output, and finally to recognize Arabic digits "0" … "9" we use the same architecture of English digits.

## 5.2 Multi-layer Feedforward Network for Encoding of the Pattern

Here, a backpropagation neural network algorithm and all the methods that improved this algorithm are used to encode English and Arabic alphabets and digits.

The representation of these patterns is the same as the representation of pattern recognition. Now, we need to name each of the output categories and we can assign a simple 4 or 5 bit binary code. Figure (5) shows the architecture of the Arabic alphabet encoding.

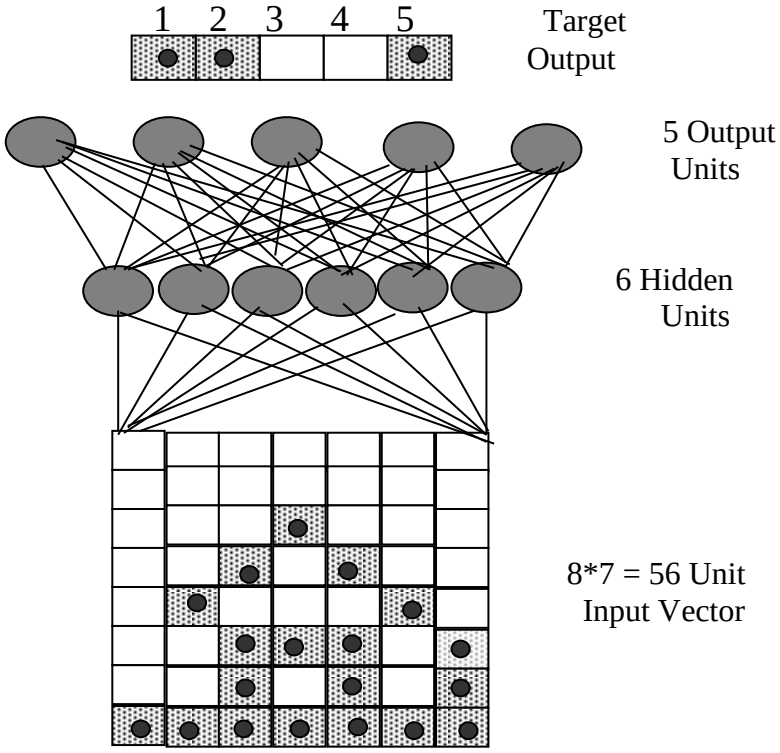
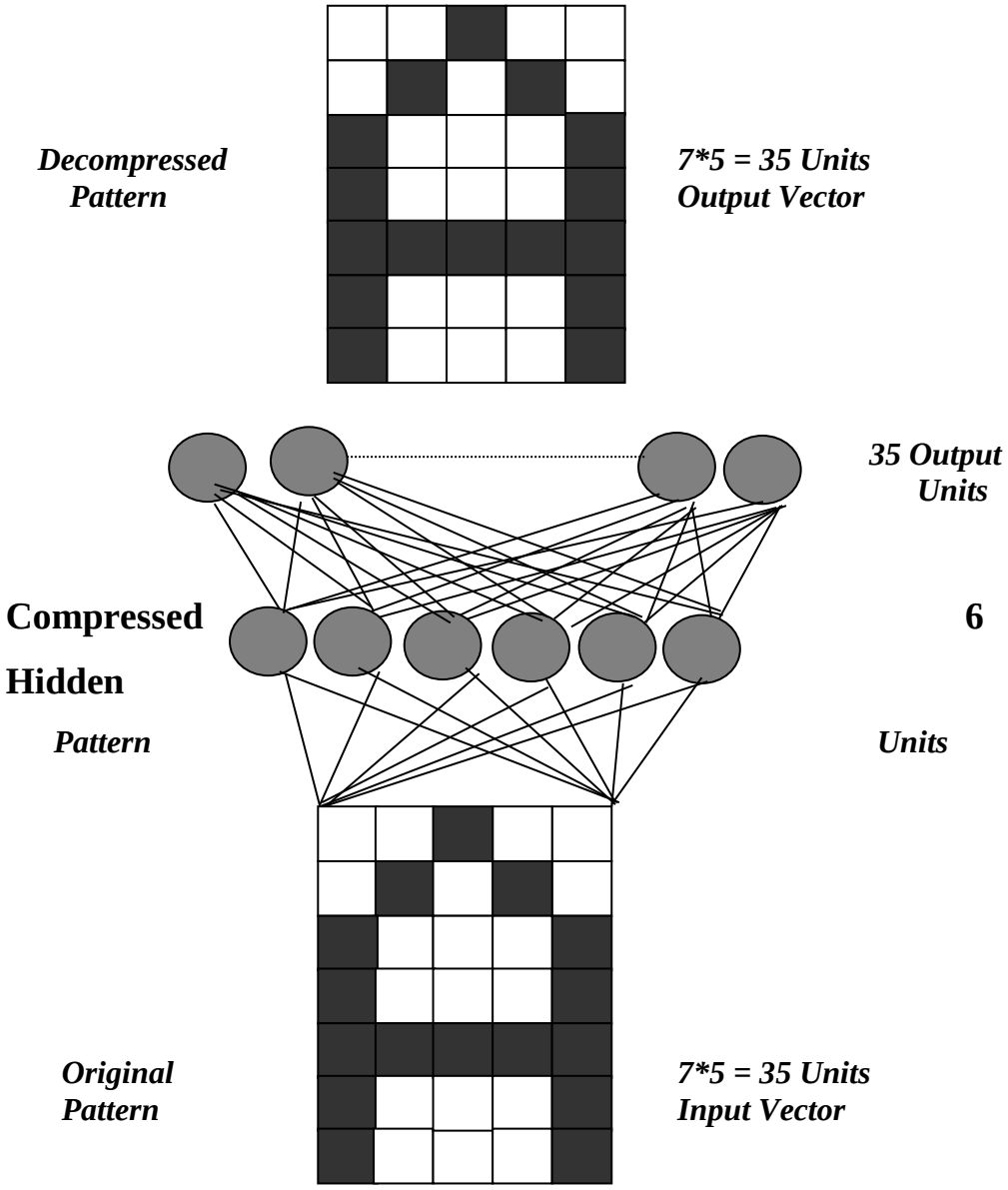



Figure (5) Architecture of Arabic alphabet encoding

### 5.3 Multi-layer Feedforward Network for Pattern Compression

A backpropagation neural network algorithm and all the methods of speeding up this algorithm are used to compress the English, Arabic alphabet and also English, Arabic digits.The representation of the pattern is the same as the representation in pattern recognition and encoding, but the output pattern has the same dimension of input.

By converting the original data that are represented by conventional code into a different code, compression algorithm may provide a level of security illicit monitoring. Figure (6) illustrates the architecture of the English alphabet compression.




Figure(6) Architecture of English alphabet compression

## 6. Results of methods to speed up EBP algorithm

In this section, the methods to speed up EBP algorithm described in section 4 are compared to the EBP algorithm described in section 1 on the basis of the number of adaptation cycles required by each algorithm to achieve the same small value of the total error.

The five algorithms are used individually to train the same BPNN architecture, and with the same initial set of synaptic weights that were provided by a random number generator producing numbers between –0.01 and 0.01, for each application individually [i.e. when we take Arabic alphabetic character recognition the same architecture is used for EBP and five modification methods, and so on ).

This research, applied the EBP algorithm and all five modification methods of this algorithm for English, Arabic character and English, Arabic digits to recognition, compression and encoding of these. Here we take a sample from each type of this application, and show the difference between these methods and original BP. The SBP is aslow method where MOM method is faster than the original EBP, ESV is very fast (this method train the pattern with few iteration), B_D is faster than ESV and converge to optimal solution with very few iteration and with minimum error , SAB is faster than other methods and finally S_SAB is faster than all previous methods, this can be clearly noticed in the following tables.

Table (1) The Difference between methods for pattern compression of English digit

| Method | Input Units | Hidden Units | Out put Units | $\eta$ | $\alpha$ | Beta | $\rho$ | $\sigma$ | Iteration no. | Error |
|---|---|---|---|---|---|---|---|---|---|---|
| SBP | 35 | 10 | 35 | 0.9 | 0.0 | 0.0 | 0.0 | 0.0 | 2271 | 0.099987 |
| MOM | 35 | 10 | 35 | 0.9 | 0.5 | 0.0 | 0.0 | 0.0 | 1121 | 0.099987 |
| ESV | 35 | 10 | 35 | 0.9 | 0.0 | 4.0 | 0.0 | 0.0 | 545 | 0.099992 |
| B_D | 35 | 10 | 35 | 0.7 | 0.0 | 0.0 | 1.1 | 0.5 | 110 | 0.009619 |
| SAB | 35 | 10 | 35 | 0.9 | 0.5 | 0.0 | 1.1 | 0.0 | 43 | 0.009515 |
| S_SAB | 35 | 10 | 35 | 0.9 | 0.3 | 0.0 | 1.1 | 0.5 | 32 | 0.035117 |

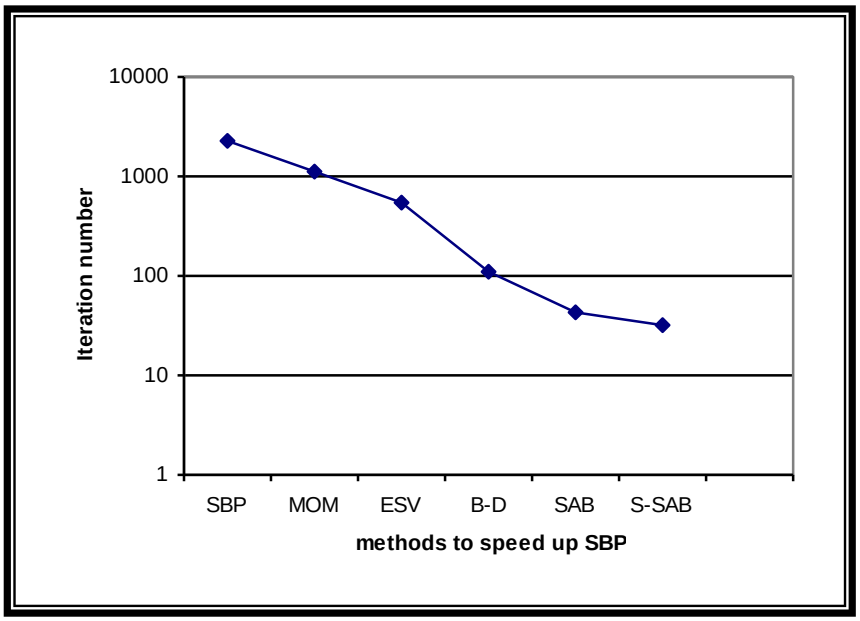



Figure (7) The Difference between methods for pattern compression of English digit

Table (2) The Difference between methods for pattern encoding of Arabic digits

| Method | Input Units | Hidden Units | Output Units | $\eta$ | $\alpha$ | Beta | $\rho$ | $\sigma$ | Iteration no. | Error |
|---|---|---|---|---|---|---|---|---|---|---|
| SBP | 35 | 5 | 4 | 0.9 | 0.0 | 0.0 | 0.0 | 0.0 | 3796 | 0.009999 |
| MOM | 35 | 5 | 4 | 0.7 | 0.7 | 0.0 | 0.0 | 0.0 | 1389 | 0.009997 |
| ESV | 35 | 5 | 4 | 0.9 | 0.0 | 6.0 | 0.0 | 0.0 | 421 | 0.009990 |
| B_D | 35 | 5 | 4 | 0.9 | 0.0 | 0.0 | 1.1 | 0.5 | 64 | 0.009916 |
| SAB | 35 | 5 | 4 | 0.9 | 0.5 | 0.0 | 1.1 | 0.0 | 38 | 0.009914 |
| S_SAB | 35 | 5 | 4 | 0.9 | 0.3 | 0.0 | 1.1 | 0.5 | 20 | 0.033850 |

Table (3) The Difference between methods for pattern recognition of Arabic alphabet

| Method | Input Units | Hidden Units | Output Units | $\eta$ | $\alpha$ | Beta | $\rho$ | $\sigma$ | Iteration no. | Error |
|---|---|---|---|---|---|---|---|---|---|---|
| SBP | 56 | 6 | 28 | 0.9 | 0.0 | 0.0 | 0.0 | 0.0 | 357 | 0.399824 |
| MOM | 56 | 6 | 28 | 0.9 | 0.5 | 0.0 | 0.0 | 0.0 | 174 | 0.399503 |
| ESV | 56 | 6 | 28 | 0.9 | 0.0 | 6.0 | 0.0 | 0.0 | 23 | 0.398148 |
| B_D | 56 | 6 | 28 | 0.9 | 0.0 | 0.0 | 1.1 | 0.5 | 14 | 0.392830 |
| SAB | 56 | 6 | 28 | 0.9 | 0.5 | 0.0 | 1.1 | 0.0 | 12 | 0.278834 |
| S_SAB | 56 | 6 | 28 | 0.9 | 0.1 | 0.0 | 1.1 | 0.5 | 8 | 0.023967 |

## ***REFERENCES***

[1] Abod L.K., (1998) classification of satellite image using Neural Network, Ph.D. Thesis, Department of physics, College of Sciences, University of Baghdad .

[2] Baluja S., (June 1996) Evolution of an Artificial Neural Network Based Autonomous Land Vehicle controller, IEEE transaction or system, MAN, And Cybernetics-part Bi Cybernetics, Vol. 26, No. 3, PP.450-463.

[3] Chitra, S.P.(April 1993) use of Neural Networks for problem Solving, Chemical Engineering progress, PP. 44-52.

[4] Fukuoka Y. Matsuki H., Minamitani H., Ishida A., (June 1998) A modified back-propagation method to avoid false local minma, Neural Networks, Japan, PP. 1059-1072.

[5] Jones W.P. and Hoskins J., (october 1987) Back-propagation Ageneralized delta learning rule, BYTE, PP. 155-162.

[6] Lippmann R.P., (April 1987) Introduction to Computing with Neural Nets, IEEE Assp Magazine, Vol. 4, No. 2, PP. 4-22.

[7] Rzempoluck E.J., (1998) Neural Network Data Analysis Using Simulnet, Springer – Verlag NewYork.

[8] Samad T.,(April 1991) Backpropagation with Expected Source Values , Neural Network,Vol. 4,pp. 615-618.

[9] Wasserman, P. D., (1989) Neural Computing Theory and Practice, Van Nostrand Reinhold, NewYork.

[10] Zurada J. M., (1994) Introduction to Artificial Neural Systems, Jaico publishing House, Mumbai.