Advances in Computer Science and Information Technology (ACSIT)
Print ISSN: 2393-9907; Online ISSN: 2393-9915; Volume 2, Number 4; April-June, 2015 pp. 366-369
© Krishi Sanskriti Publications
http://www.krishisanskriti.org/acsit.html

Comparison of Feed Forward Neural Network Training Methods for Visual Character Recognition

Sukhleen Bindra Narang[1], Manjeet Singh[2] and Kunal Pubby[3]

[1]Department of Electronics Technology, Guru Nanak Dev University, Amritsar, Punjab
[2]Ex-Student Department of Electronics Technology, Guru Nanak Dev University, Amritsar, Punjab
[3]M.tech. Student, Department of Electronics Technology, Guru Nanak Dev University, Amritsar, Punjab
E-mail: [1]sukhleen2@yahoo.com, [3]kunalpubby02@gmail.com

Abstract—*Feed-Forward Neural Network (FFNN) can be used to recognize the characters from images. This paper compares the seven different training algorithms belonging to two classes: Gradient descent (with variable learning rate, with variable learning rate and momentum, resilient back-propagation), and conjugate gradient (Fletcher-Reeves update, Polak-Ribiére update, Powell-Beale restart, scaled conjugate gradient), which are used to train the BP network. The different training algorithms are compared in terms of training performance, training time and number of training iterations to reach the optimal weights.*

Keywords: *Gradient descent; conjugate gradient.*

1. INTRODUCTION

The simultaneous availability of inexpensive powerful computers, powerful learning & training algorithms, and large databases, have caused rapid progress in character recognition in the last few years. While recognizing individual character is only one of many problems involved in designing a practical recognition system, it is an excellent benchmark for comparing shape recognition methods. This study concentrates on different training algorithms that operate directly on size-normalized images.

Seven different training algorithms belonging to two classes: gradient descent (with variable learning rate, with variable learning rate and momentum, resilient back-propagation), and conjugate gradient (Fletcher-Reeves update, Polak-Ribiére update, Powell-Beale restart, scaled conjugate gradient) are evaluated using a dataset of 36 binary images. Weights were initialized to random values and Training stops when any of these conditions was satisfied:

1. Maximum Epoch
2. Minimum Gradient
3. Performance Goal

2. CHARACTER RECOGNITION

2.1 Feed-Forward Neural Network (FFNN)

Neural network (NN) can be considered as non-linear statistical data modeling tool that can model almost any nonlinear relationship that may exist between inputs and outputs or find patterns in data. These computational models are characterized by their architecture, learning algorithm and activation function [1]. The feed-forward NN (FFNN) architecture is selected in this study. The FFNN consists of one or more nonlinear hidden layers. The hidden layers' activation functions are sigmoidal functions that empower the network to learn the complex and nonlinear relationship between the inputs and the targets. In this architecture, a unidirectional weight connection exists between each two successive layers. A two-layer FFNN with sigmoidal functions in the hidden layer and output layer can potentially approximate any function with finite number of discontinuities, provided a sufficient number of neurons exists in the hidden layer [2]. The Log-sigmoid transfer function was used in hidden layer and the output layer.

3. THE ALGORITHM

In MATLAB, a feed-forward back-propagation network is created using *newff* function. User needs to provide input argument such as input and output data, hidden layer and node size, node activation function, networks training algorithm and etc. The GDA, Rprop and SCG training are specified using *traingda, trainrp* and *trainscg* respectively. The initial weight of the networks is randomly created by default, every time the *newff* is called. User also has a choice to initialize the random initial weight using *rands* command. The BP weight training can be directly executed using *train* function once the networks and the parameters are properly set. Here ms is column normalized array of each character and of size <1008 X 36>.

```
net =newff(minmax(ms), [25,36], {'logsig', logsig'},'traingda');
```

4. TRAINING ALGORITHMS

Training is the process of determining the optimal weights of the NN. This is done by defining a performance function (which is usually the mean square error between the network's output and the desired target) and then minimizing it with respect to weights. The minimization is performed by calculating the gradient using a technique called back-propagation which can be done in batch or incremental styles [1]. In this paper, we used the batch training style. The NNs were trained using different training algorithms belonging to two back-propagation classes described as follows:

4.1 Gradient Descent (GD)

Gradient descent is also known as steepest descent, or the method of steepest descent. This method updates the network weights and biases in the direction of the performance function that decreases most rapidly, i.e. the negative of the gradient. One iteration of this algorithm can be written as follows [1]:

$$\Delta w_i = -\mu_i g_i \qquad (1)$$

Where, Δw_i is the vector of weights changes, g_i is the vector of gradients and μ_i is the learning rate that determines the length of the weight update. Although the GD algorithm is easy to implement, it has several disadvantages such as slow learning, requiring a good training dataset, getting stuck in local minima and having little or no robustness to noise. Other modifications can be applied to improve the performance of GD algorithm.

1) Gradient Descent with Variable Learning Rate (GDA):

The learning rate parameter is used to determine how fast the BP method converges to the minimum solution. The larger the learning rate, the bigger the step and the faster the convergence. However, if the learning rate is made too large the algorithm will become unstable. In the adaptive learning technique, the step size is chosen as large as possible while keeping learning stable. In each iteration, if the new error is greater than the old one by a predefined ratio (here set to 1.04), the new parameters (weights) are discarded and the learning rate is decreased (here by multiplying by 0.7). Otherwise, the new parameters are kept. If the new error is less than the old error, the learning rate is increased (here by multiplying by 1.05). The initial value of learning rate was set to 0.01. In Matlab, Backpropagation training with an adaptive learning rate is implemented with the function 'traingda'.

2) Gradient Descent with Variable Learning Rate and Momentum (GDX):

It combines adaptive learning rate with momentum training. In order to reduce the sensitivity of the network to fast changes of the error surface, a fraction of the previous weight change (called momentum term) can be added to the gradient decreasing term, as follows [1]:

$$\Delta w_i = -\mu_i g_i + p\,\Delta w_{i-1} \qquad (2)$$

where p(lies between 0 to 1) is the momentum parameter. The momentum parameter p was set to 0.9 in our application. It is invoked in the same way as traingda, except that it has the momentum coefficient (mc) as an additional training parameter.

3) Resilient Backpropagation (RP)

It is a first-order optimization algorithm. For the FFNNs with sigmoidal activation functions, the gradient can be of very small magnitude even though the weights are far from their optimum values. This is due to the slope of sigmoidal functions that approaches zero as the input magnitude increases [3]. A solution to this problem is to use only the direction of the gradient to update the weights, while the amount of the update is determined by another update factor (here initially set to 0.07). When the gradient has the same sign for two successive iterations, the update factor is increased by a ratio (here set to 1.05). The update factor is decreased (here by multiplying by 0.8) when the gradient changes sign from the previous iteration. When the derivative is zero, the update value is not changed [3] It also has the nice property that it requires only a modest increase in memory requirements. You do need to store the update values for each weight and bias, which is equivalent to storage of the gradient.

4.2 Conjugate Gradient (CG)

All the conjugate gradient algorithms start out by searching in the steepest descent direction (negative of the gradient) on the first iteration.

$$p_0= -g_0 \qquad (3)$$

A line search is then performed to determine the optimal distance to move along the current search direction:

$$x_{k+1} = x_k \alpha_k p_k \qquad (4)$$

Then the next search direction is determined so that it is conjugate to previous search directions. The general procedure for determining the new search direction is to combine the new steepest descent direction with the previous search direction [7]:

$$p_k = -g_k + \beta_k p_{k-1} \qquad (5)$$

where $\boldsymbol{\beta_k}$ is a constant which is computed to force the consecutive directions to be conjugate.

The various versions of the conjugate gradient algorithm are distinguished by the manner in which the constant $\boldsymbol{\beta_k}$ is computed.

1) Fletcher-Reeves Update (CGF):
This method updates $\boldsymbol{\beta_k}$ as the norm squared of the current gradient vector divided by the norm squared of the previous gradient vector [4]:

$$\beta_k = g_k^T g_k / g^T_{k-1} g_{k-1} \qquad (6)$$

2) Polak-Ribiére Update (CGP):
This method updates $\boldsymbol{\beta_k}$ as the dot product of the previous change in the gradient vector with the current gradient vector

divided by the norm squared of the previous gradient vector [4]:

$$\beta_k = \Delta g_{k-1}^T g_k / g^T_{k-1} g_{k-1} \quad (7)$$

3) Powell-Beale Restarts (CGB):
In all CG algorithms the search direction is periodically reset to the negative direction of the gradient. The standard reset point occurs when the number of iterations is equal to the number of network parameters (weights and biases), but there are other reset methods that can improve the efficiency of training. One such reset method was proposed by Powell and Beale. This technique restarts if there is very little orthogonality left between the current gradient and the previous gradient. This is tested with the following inequality: [5]:

$$|g^T_{k-1} g_{k-1}| \geq 0.2 \, \|g_k\|^2 \quad (8)$$

If this condition is satisfied, the search direction is reset to the negative of the gradient.

4) Scaled Conjugate Gradient (SCG):
The introduced CG algorithms so far are based on a line search in each iteration which is computationally expensive. To overcome this shortcoming, the SCG method combines the trust region approach with CG algorithm. SCG does not contain any user dependent parameters whose values are crucial for the success of SCG. By using a step size scaling mechanism, SCG avoids time consuming line search operations which makes the algorithm faster than any other second order algorithm. For more details, the reader is referred to [6].

Same as the GD algorithms, in most of the CG algorithms, the step size is adjusted at each iteration. Here, the Charalambous' search method [7] was utilized in all the CG algorithms except the SCG to adjust the step size.

5. EXPERIMENTAL RESULTS

5.1 Database

The database used to train and test the systems described in this paper contains the binary images of characters as shown in Fig. 1.

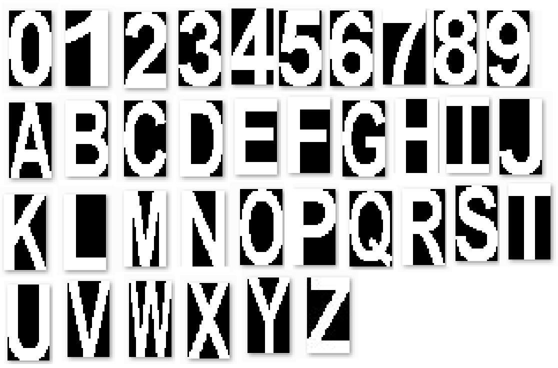

Fig. 1: Numeric and alphabetic Symbols

All the images were size normalized to fit in a 42x24 pixel box (while preserving the aspect ratio). Each character was normalized using column normalization in MATLAB. The normalized data was used as the inputs to the network. There were 36 target values which correspond to one for each training sample and size of the network was 1008 x 36.

5.2 Evaluation Criteria

The BP estimation results were compared in terms of training performance. The training performance of the NNs was compared in terms of *1)* training time, *2)* number of training iterations to reach the optimal weights, 3) mean square error and 4) Regression rate.

5.3 Parameter setup

The default value of Matlab ANNs training parameters were used for all the algorithms except for the learning rate, goal and the maximum number of epochs.

net.trainParam.goal=0;

net.trainParam.epoches = 2000;

net.trainParam.lr = 0.01;

5.4 Train and Test Strategy

Same dataset was used for all training algorithms. The network was trained with 100 % training dataset. Training was done using feed-forward net with 1 hidden layer and at different number of hidden units. Input and output layers were having log sigmoid transfer function. The mean square error (MSE) is the condition to terminate training of all the BP methods. MSE is originally set at 0, however, these BP methods can also stop once the training exceeds the number of epoch set. After the Training, network is simulated with same input data to calculate the accuracy.

5.5. Results

The performance of different training algorithms is shown in Table1, and Table2. Table 1 shows the performance of training algorithms at 25 hidden units. Table 2 shows the performance of

Table 1: Training performance at 25 hidden units

		Avg. No: of epochs	**Avg. Training Time(s)**	**MSE**	**Regression**	**Accuracy(%)**
Gradient Descent	GDA	1568	13	2.89e-05	0.9998	97.2
	GDX	583	5	0.00157	0.97069	94.4
	RP	59	0.7	2.08e-05	0.99962	100
	CGP	2000	54	0.00674	0.87163	77.8
Conjugat	CGB	2000	57	0.01014	0.7922	41.6

e Gradient	SCG	345	5	0.000772	0.98561	100
	CGF	2000	53	0.0229	0.45771	16

Table 2: Training Performance at 50 hidden units

		Avg. No: of epochs	Avg. Training Time(s)	MSE	Regression	Accuracy (%)
Gradient Descent	GDA	614	8	0.00156	0.9708	97.2
	GDX	479	7	1.78e-05	0.9986	100
	RP	34	1	7.27e-07	0.9999	100
Conjugate Gradient	CGP	327	13	0.00307	0.94152	86.1
	CGB	438	16	0.00224	0.95773	88.8
	SCG	279	5	9.63e-07	0.9999	100
	CGF	1288	47	0.00771	0.84641	77.8

training algorithms at 50 hidden units. These values were averaged over the 10 runs of the algorithms. The results were obtained on Intel i5, 2.67 GHz processor with 3.0 GB of RAM. Among the GD algorithms with 25 hidden units,

The best results in terms of both speed and accuracy are obtained using RP. Among the GD algorithms with 50 hidden units, GDX has improved training performance than GDA, but RP is still leading in overall performance.

In case of CG, with 25 hidden units, CGF and CGB have worst training performance and accuracy. CGP has improved training performance, but training time is much longer.

Best results are obtained using SCG in terms of time, speed and Accuracy.

Among the CG algorithms with 50 hidden units,

CGF has worst training performance and accuracy while CGP and CGB have almost same training parameters. Again, best results are obtained using SCG with much improved training performance.

Comparing all the seven different training algorithms together, it is found that the training performances significantly differ. However, the best training performance and fastest training are obtained using the RP algorithm.

6. CONCLUSION AND FUTURE WORK

Comparison of the results of the study indicated that the original research hypothesis that RP would perform faster in all cases was proven correct. In CG, SCG was superior in training performance and training time. To eliminate the possible confounding variables in this study, the number of trials could be increased and the size of the data sets also enlarged. Other back-propagation based algorithms could be comparatively tested fairly easily, utilizing the same data sets and similar network structures.

Besides the BP methods, there also are other neural networks methods like the radial basis function (RBF) that we can implement in character recognition. These methods will be studied next.

REFERENCES

[1] Jang, J.S.R., Sun, C.T. and Mizutani, E., Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence, NJ: Prentice-Hall, 1997.

[2] Cybenko, G., “Approximation by superpositions of a sigmoidal function”, Math Control Signals Systems, Vol. 2, pp. 303-314, 1989.

[3] Riedmiller, M. and Braun, H., “A direct adaptive method for faster backpropagation learning: The RPROP algorithm”, in proc. IEEE International Conference on Neural Networks, San Fransisco, USA, Mar. 1993, pp. 586 – 591.

[4] Fletcher, R. and C.M. Reeves, C.M., “Function minimization by conjugate gradients,” Computer Journal, Vol. 7, pp. 149–154, 1964.

[5] Powell, M.J.D., “Restart procedures for the conjugate gradient method,” Mathematical Programming, Vol. 12, pp. 241–254, Dec. 1977.

[6] Moller, M.F., “A scaled conjugate gradient algorithm for fast supervised learning”, Neural Networks, Vol. 6, pp. 525–533, 1993.

[7] Charalambous, C., “Conjugate gradient algorithm for efficient training of artificial neural networks,” IEEE Proceedings, Vol. 139, pp. 301–310, June 1992.

[8] Moriera, M. and Fiesler, E., “Neural Networks with Adaptive learning rate and Momentum terms,” IDIAP Technical Report, October 1995.