NEURAL NETWORKS BASED NOISE CANCELLER

Dr. Haider Hadi Abbas

Mansour University College

Abstract:

This paper deals with the implementation of noise canceller using neural networks. Two types of noise canceller has been software implemented. Then the performance of them has been checked for different training algorithms.

At first, batch gradient descent algorithm is used with different noise levels. The effect of increasing noise levels is to degrade the performance (increasing the mean square error) for the two systems under consideration. Then for the same training algorithm, the effect of hidden layer neurons is investigated. It is clear that increasing hidden layer neurons degrades the performance of the two above systems.

The second training algorithm which is used is gradient descent with momentum. The effect of increasing momentum is to improve the performance of the two noise canceller systems.

The third training algorithm which is used is gradient descent with variable learning rate. As in the case of momentum, increasing learning rate improves the performance of the two above systems.

Then the effect of momentum with learning rate increase is investigated. It is clear that the effect of momentum with learning rate increase will improve the performance.

The last training algorithm which is used is resilient backpropagation and it gives the best performance at all.

1. INTRODUCTION

Neural networks can be trained to recognize and produce both spatial and temporal patterns.

An example of a problem where temporal patterns are recognized and classified with a spatial pattern is noise cancellation.

Noise cancellation requires that a noisy waveform be presented to the network through time, and the network output the waveform without noise [1, 2, 3].

2. BACKPROPAGATION ALGORITHM

The simplest implementation of backpropagation learning updates the network weights and biases in the direction in which the performance function decreases most rapidly (the negative of the gradient). One iteration of this algorithm can be written

$$X_{k+1} = X_k - \alpha_k g_k \quad (1)$$

Where X_k is a vector of current weights and biases, g_k is the current gradient, and α_k is the learning rate.

There are two different ways in which this algorithm can be implemented: incremental mode and batch mode. In the incremental mode, the gradient is computed and the weights are updated after each input is applied to the network. In the batch mode all of the inputs are applied to the network before the weights are updated. The next section describes the batch mode of training [1, 2, 3, 4, 5, 6].

3. BATCH TRAINING

In batch mode the weights and biases of the network are updated only after the entire training set has been applied to the network. The gradients calculated at each training example are added together to determine the change in the weights and biases. The original algorithm to accomplish the above task is batch gradient descent algorithm [7, 8, 9, 10].

4. BATCH GRADIENT DESCENT WITH MOMENTUM

In addition to batch gradient descent algorithm, there is another batch algorithm for feedforward networks that often provides faster convergence which is steepest descent with momentum. Momentum allows a network to respond not only to the local gradient, but also to recent trends in the error surface. Acting like a low-pass filter, momentum allows the network to ignore small features in the error surface. Without momentum a network may get stuck in a shallow local minimum. With momentum a network can slide through such a minimum. Momentum can be added to backpropagation learning by making weight changes equal to the sum of a fraction of the last weight change and the new change suggested by the backpropagation rule. The magnitude of the effect that the last weight change is allowed to have is mediated by a certain momentum constant which can be any number between 0 and 1. When the momentum constant is 0, a weight change is based solely on the gradient. When the momentum constant is 1, the new weight change is set to equal he last weight change and the gradient is simply ignored. The gradient is computed by summing the gradients calculated at each training example, and the weights and biases are only updated after all training examples have been presented [7, 8, 9, 10].

5. FASTER TRAINING

The previous section presented two backpropagation training algorithms: gradient descent, and gradient descent with momentum. These two methods are often too slow for practical problems. In this section we discuss several high performance algorithms that can converge from ten to one hundred times faster than the algorithms discussed previously. All of the algorithms in this section operate in the batch mode [11, 12, 13].

5.1 Variable Learning Rate

With standard steepest descent, the learning rate is held constant throughout training. The performance of the algorithm is very sensitive to the proper setting of the learning rate. It is not practical to determine the optimal setting for the learning rate before training, and, in fact, the optimal learning rate changes during the training process, as the algorithm moves across the performance surface. The performance of the steepest descent algorithm can be improved if we allow the learning rate to change during the training process. An adaptive learning rate will attempt to keep the learning step size as large as possible while keeping learning stable. The learning rate is made responsive to the complexity of the local error surface. An adaptive learning rate requires some changes in the original training procedure. First, the initial network output and error are calculated. At each iteration new weights and biases are

calculated using the current learning rate. New outputs and errors are then calculated [11, 12, 13].

5.2 Resilient Backpropagation

Multilayer networks typically use sigmoid transfer functions in the hidden layers. These functions are often called "squashing" functions, since they compress an infinite input range into a finite output range. Sigmoid functions are characterized by the fact that their slope must approach zero as the input gets large. This causes a problem when using steepest descent to train a multilayer network with sigmoid functions, since the gradient can have a very small magnitude; and therefore, cause small changes in the weights and biases, even though the weights and biases are far from their optimal values.

The purpose of resilient backpropagation training algorithm is to eliminate these harmful effects of the magnitudes of the partial derivatives. Only the sign of the derivative is used to determine the direction of the weight update; the magnitude of the derivative has no effect on the weight update. The size of the weight change is determined by a separate update value. The update value for each weight and bias is increased by a certain factor whenever the derivative of the performance function with respect to that weight has the same sign for two successive iterations. The update value is decreased by the same factor whenever the derivative with respect that weight changes sign from the previous iteration. If the derivative is zero, then the update value remains the same. Whenever the weights are oscillating the weight change will be reduced. If the weight continues to change in the same direction for several iterations, then the magnitude of the weight change will be increased [11, 12, 13].

6. PROBLEM DEFINITION

The problem to be solved in this paper is as follows:

A noisy waveform (sin or square) is introduced to the network with different noise levels are considered. Four patterns are used as an input to the network. The target will be the waveform without noise (sin or square).

7. SOFTWARE IMPLEMENTATION

7.1 Sine Wave Noise Canceller

The software implementation of this part is shown in the following flowchart:

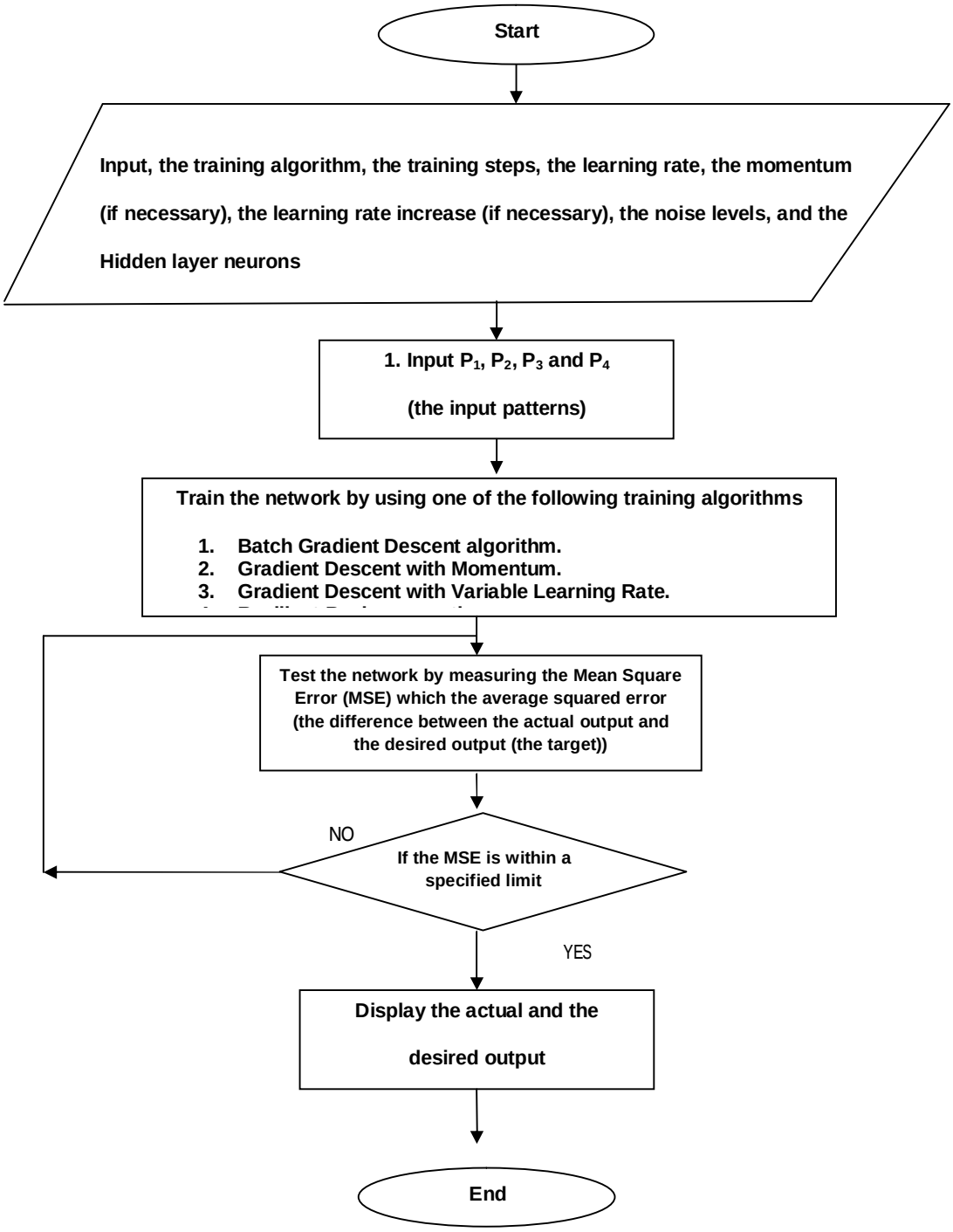

Fig. (1) sine and cosine wave noise canceller Flowchart

For this case the input patterns are:

p_1= sin (2p 50 t) + Noise 1

p_2= sin (2p 50 t) + Noise 2

p_3= sin (2p 50 t) + Noise 3

p_4= sin (2p 50 t) + Noise 4

Noise 1, Noise 2, Noise 3 and Noise 4 are chosen by the program.

And the output is:

$$t_1 = \sin(2p\ 50\ t)$$

The program for the above flowchart is written using Matlab programming language.

7.2 Square Wave Noise Canceller

The software implementation of this part uses the same flowchart shown in Fig. (1).

The difference between two models is that for this case the input patterns are:

p_1 = square (2p 100 t) + Noise 1

p_2 = square (2p 100 t) + Noise 2

p_3 = square (2p 100 t) + Noise 3

p_4 = square (2p 100 t) + Noise 4

And the output is:

$$t_1 = \text{square}(2p\ 100\ t)$$

The program for this part is written by the same programming language used in part one

8. SOFTWARE TESTING

The implemented noise canceller performance is tested for different cases. The performance measure is the Mean Square Error (MSE) which the average

squared error (the difference between the actual output and the desired output (the target)). The implemented software is tested for the following cases:

8.1 Effect of Noise Levels

In this case three sets of noise levels are considered which are:

1- 0.01 0.015 0.02 0.025

2- 0.015 0.02 0.025 0.05

3- 0.02 0.025 0.05 0.1

With the training algorithm is gradient descent algorithm.

- **a- The training steps are 50.**
- **b- The learning rate is 0.05.**
- **c- The number of iteration is 1000.**
- **d- The hidden layer neurons are 20.**

The output for the two cases (sin and square) is shown in Table (1).

Table (1)

	(Sine wave / mean square error)	(Square wave / mean square error)
1	0.0015	0.00086
2	0.0022	0.00123
3	0.0041	0.00134

From the above table, it is clear that increasing noise levels degrades the performance (increases the mean square error) for the two cases.

8.2 Effect of Hidden Layer Neurons

In this case five values for hidden neurons are considered which are:

10, 20, 30, 40, and 50.

With

- **a- The training algorithm is gradient descent algorithm.**
- **b- The training steps are 50.**
- **c- The learning rate is 0.05.**
- **d- The number of iteration is 1000.**
- **e- The noise levels are 0.015, 0.02, 0.025, and 0.05.**

The output for the two cases is shown in Table (2).

Table (2)

	(Sine wave / mean square error)	(Square wave / mean square error)
1	0.00213	0.00045
2	0.0022	0.0014
3	0.0034	0.2475
4	0.0074	Does not converge
5	Does not converge	Does not converge

From the above table, it is clear that increasing hidden layer neurons degrades the performance (increases the mean square error). The reason for this behavior is that the problem under consideration does not need a big number of hidden layer neurons.

8.3 Effect of Momentum

In this case three values for momentum are considered which are:

1- 0.2
2- 0.5
3- 0.9

With

- **a- The training algorithm is gradient descent with momentum.**
- **b- The training steps are 50.**
- **c- The number of iterations is 1000.**
- **d- The noise levels are 0.015, 0.02, 0.025 and 0.05.**
- **e- The hidden layer neurons are 50.**

The output for the two cases is shown in Table (3).

Table (3)

	(Sine wave / mean square error)	(Square wave / mean square error)
1	0.0133	Doesn't converge
2	0.0094	0.0014
3	0.0085	0.00136

From the above table, it is clear that increasing the momentum improves the performance (decreases the mean square error).

8.4 Effect of learning Rate Increase

In this case three values for learning rate increase are considered which are:

1- 1.01
2- 1.05
3- 1.09

With

a- The training algorithm is gradient descent algorithm with variable learning rate.
b- The training steps are 50.
c- The learning rate is 0.05.
d- The number of iterations is 1000.
e- The noise level is 0.015, 0.02, 0.025, 0.05
f- The hidden layer neurons is 50.

The output for the two cases is shown in Table (4).

Table (4)

	(Sine wave / mean square error)	(Square wave / mean square error)
1	0.0214	0.01
2	0.0199	0.0076
3	0.0188	0.0033

From the above table, it is clear that increasing learning rate improves the performance (decreases the mean square error). The reason for this behavior is the capability of increasing learning rate if necessary.

8.5 Effect of Momentum (With Learning Rate Increasing)

In this case the same values given in section 8.3 are given but with:

1. The learning rate increase is 1.05.

2. The training algorithm is gradient descent with momentum and variable learning rate.

The output for the two cases is shown in Table (5).

Table (5)

	(Sine wave / mean square error)	(Square wave / mean square error)
1	0.008	0.0032
2	0.0048	0.0028
3	0.0018	0.0015

From the above table, it is clear that the momentum will improve the performance (decreases the mean square error) if it is included in addition to learning rate increase.

8.6 Using Faster Training algorithms

In this subsection resilient backpropagation algorithm is used with the following parameters:

a. The noise levels are 0.015, 0.02, 0.025 and 0.05.

b. The training step is 50.

c. The number of hidden layer neurons is 50.

d. The number of iterations is 1000.

And with three values for learning rate which are:

1- 0.01
2- 0.05
3- 0.09

The output for the two cases is shown in Table (6).

Table (6)

	(Sine wave /mean square error)	(Square wave / mean square error)
1	0.00138	$3.47 * 10^{-6}$
2	0.00105	$2.89 * 10^{-6}$
3	0.00104	$4.97 * 10^{-6}$

From the above table, two points are concluded which are:

1. **Resilient backpropagation algorithm gives the best performance at all.**
2. **Increasing the learning rate improves the performance.**

The following table summarizes the testing results:

Table (7)

Training algorithm	Testing parameter	Testing result
Batch Gradient Descent algorithm	Increasing noise levels	Degrading the performance (increasing the mean square error)
Batch Gradient Descent algorithm	Increasing Hidden Layer Neurons	Degrading the performance (increasing the mean square error)
Gradient Descent with Momentum	Increasing the Momentum	Improving the performance (decreasing the mean square error)
Gradient Descent with Variable Learning Rate	Increasing the learning rate	Improving the performance (decreasing the mean square error)
Resilient Backpropagation		Improving the performance (decreasing the mean square error). This algorithm gives the best results at all.

9. CONCLUSIONS

The following points are concluded from this work:

1. Increasing noise Levels degrades the performance of the system.

2. Increasing hidden layer neurons degrades the performance of the system .

3. The effect of momentum will improve system performance .

4. Increasing Learning rate improves the performance of the system .

5. Resilient backpropagation algorithm gives the best performance at all .

9. REFERENCES

1. Bosque, M. (2002), Understanding 99% of Artificial Neural Networks: Introduction and Tricks, Writers Club Press.

2. Fansett (1994), Fundamentals of neural networks, prentice Hall.

3. Gurney (1997), An Introduction to neural networks, VCL press.

4. Haykin, S. (1999), Neural Networks, second edition, prentice Hall.

5. Haykin, S. (2008), Neural Networks and Learning Machines, Prentice Hall.

6. Aleksander, I., and H. Morton (1990), An Introduction to Neural Computing, Chapman and Hall, London.

7. Fausett, L. (1994), Fundamentals of Neural Networks – Architectures, Algorithms and Applications, Prentice Hall, Englewood Cliffs, NJ.

8. Dreyfus, S. E. (1990), "Artificial Neural Networks, Backpropagation and the Kelley-Bryson Gradient Procedure", Journal of Guidance, Control and Dynamics, Vol. 13, No. 5, pp. 926–928.

9. Gallinari, P. (1995), "Training of Modular Neural Net Systems", in: [Arbib 1995], pp. 582–585.

10. Rojas, R., and M. Pfister (1993), "Backpropagation Algorithms", Technical Report B 93, Department of Mathematics, Free University Berlin.

11. Mandic, D., and Chambers, J. (2001), Recurrent Neural Networks for Prediction: Architectures, Learning algorithms and Stability, Wiley.

12. Moller, M. (1993), Efficient Training of Feed-Forward Neural Networks, PhD Thesis, Aarhus University, Denmark.

13. Fahlman, S. (1989), "Faster Learning Variations on Back-Propagation: An Empirical Study", in: [Touretzky et al. 1989], pp. 38–51.

مزيل الضوضاء بأستخدام الشبكات العصبية

د. حيدر هادي عباس

كلية المنصور الجامعة

المستخلص :

يتناول هذا البحث موضوع تصميم مزيل للضوضاء باستخدام الشبكات العصبية. تم بناء برنامجين مختلفين لنوعين مختلفين من مزيلات الضوضاء و من ثم تم فحص خصائص كلا النوعين و ذلك باستخدام خوارزميات تعلم مختلفة.

في البداية تم استخدام الخوارزمية (batch gradient descent) ولمستويات ضوضاء مختلفة. إن تأثير زيادة مستوى الضوضاء هو تدني المواصفات (أو زيادة mean square error) لكلا المنظومتين المأخوذتين بنظر الاعتبار. و باستخدام نفس خوارزمية التعلم تم بحث تأثير الخلايا العصبية للطبقة الخفية. و تبين انه من الواضح إن زيادة عدد الخلايا العصبية للطبقة الخفية سوف يؤدي إلى تدني المواصفات لكلا المنظومتين أعلاه.

إن خوارزمية التعلم الثانية التي تم استخدامها هي (gradient descent with momentum). أن تأثير زيادة العزم هو تحسين المواصفات لكلا النوعين من مزيلات الضوضاء المقترحة.

إن خوارزمية التعلم الثالثة التي تم استخدامها هي (gradient descent with variable learning rate) و كما في حالة زيادة العزم نجد إن زيادة معدل التعلم سوف يحسن المواصفات لكلا المنظومتين أعلاه.

تم فحص تأثير العزم مع زيادة معدل التعلم و من الواضح أن تأثير العزم مع زيادة معدل التعلم سوف يحسن المواصفات.

أن خوارزمية التعلم الأخيرة التي تم استخدامها هي (resilient backpropagation) و تبين إن هذه الخوارزمية أعطت أفضل المواصفات عل الإطلاق.