IMPROVEMENTS TO THE BACKPROPAGATION ALGORITHM

MIRCEA PETRINI [*]

ABSTACT: *This paper presents some simple techniques to improve the backpropagation algorithm. Since learning in neural networks is an NP-complete problem and since traditional gradient descent methods are rather slow, many alternatives have been tried in order to accelerate convergence. Some of the proposed methods are mutually compatible and a combination of them normally works better than each method alone.*

KEYWORDS: *Artificial Neural Network (ANN), backpropagation, extended network*

JEL CLASSIFICATION: *C*

1. INTRODUCTION

Artificial Neural Networks (ANNs) are logical methods modeled on the learning processes of human brain. Artificial Neural Networks (ANNs) works by processing information like biological neurons in the brain and consists of small processing units known as Artificial Neurons, which can be trained to perform complex calculations. As human being, we learn how to write, read, understand speech, recognize and distinguish pattern – all by learning from examples. In the same way, ANNs are trained rather than programmed.

The traditional Backpropagation Neural Network (BPNN) Algorithm is widely used in solving many practical problems. The BPNN learns by calculating the errors of the output layer to find the errors in the hidden layers. Due to this ability of Back-Propagating, it is highly suitable for problems in which no relationship is found between the output and inputs. Due to its flexibility and learning capabilities, it has been successfully implemented in wide range of applications.

A Backpropagation network consists of at least three layers of units: an input layer, at least one intermediate hidden layer, and an output layer. Typically, units are

[*] *Lecturer, University of Petroşani, Romania, petrini_mircea@yahoo.com*

connected in a feed-forward fashion with input units fully connected to units in the hidden layer and hidden units fully connected to units in the output layer.

Backpropagation Neural Network algorithm is the most popular and the oldest supervised learning multilayer feed-forward neural network algorithm proposed by Rumelhart, Hinton and Williams (Rumelhart, et al., 1986).

2. INITIAL WEIGHT SELECTION

A well-known initialization heuristic for a feed-forward network with sigmoidal units is to select its weights with uniform probability from an interval $[-\alpha,\alpha]$. The zero mean of the weights leads to an expected zero value of the total input to each node in the network. Since the derivative of the sigmoid at each node reaches its maximum value of 1/4 with exactly this input, this should lead in principle to a larger backpropagated error and to more significant weight updates when training starts. However, if the weights in the networks are very small (or all zero) the backpropagated error from the output to the hidden layer is also very small (or zero) and practically no weight adjustment takes place between input and hidden layer. Very small values of α paralyze learning, whereas very large values can lead to saturation of the nodes in the network and to flat zones of the error function in which, again, learning is very slow. Learning then stops at a suboptimal local minimum (Lisboa & Perantonis, 1991). Therefore it is natural to ask what is the best range of values for α in terms of the learning speed of the network.

Some authors have conducted empirical comparisons of the values for α and have found a range in which convergence is best (Wessels & Barnard, 1992). The main problem with these results is that they were obtained from a limited set of examples and the relation between learning step and weight initialization was not considered. Others have studied the percentage of nodes in a network which become paralyzed during training and have sought to minimize it with the "best" α (Drago & Ridella, 1992). Their empirical results show, nevertheless, that there is not a single α which works best, but a very broad range of values with basically the same convergence efficiency.

2.1. Maximizing the derivatives at the nodes

Let us first consider the case of an output node. If n different edges with associated weights $w_1, w_2, ..., w_n$ point to this node, then after selecting weights with uniform probability from the interval $[-\alpha,\alpha]$ the expected total input to the node is

$$\left\langle \sum_{i=1}^{n} w_i x_i \right\rangle = 0$$

where $x_1, x_2, ..., x_n$ is the input values transported through each edge. We have assumed that these inputs and the initial weights are not correlated. By the law of large numbers we can also assume that the total input to the node has a Gaussian distribution Numerical integration shows that the expected value of the derivative is a decreasing

function of the standard deviation σ. The expected value falls slowly with an increase of the variance. For σ = 0 the expected value is 0.25 and for σ = 4 it is still 0.12, that is, almost half as big.

The variance of the total input to a node is

$$\sigma^2 = E((\sum_{i=1}^{n} w_i x_i)^2) - E((\sum_{i=1}^{n} w_i x_i))^2 = \sum_{i=1}^{n} E(w_i^2)E(x_i^2),$$

since inputs and weights are uncorrelated.

If n = 100, selecting weights randomly from the interval [−1.2, 1.2] leads to a variance of 4 at the input of a node with 100 connections and to an expected value of the derivative equal to 0.12.

Therefore in small networks, in which the maximum input to each node comes from fewer than 100 edges, the expected value of the derivative is not very sensitive to the width α of the random interval, when α is small enough.

2.2. Maximizing the backpropagated error

In order to make corrections to the weights in the first block of weights (those between input and hidden layer) easier, the backpropagated error should be as large as possible. Very small weights between hidden and output nodes lead to a very small backpropagated error, and this in turn to insufficient corrections to the weights. In a network with *m* nodes in the hidden layer and *k* nodes in the output layer, each hidden node h receives a backpropagated input δ_h from the *k* output nodes, equal to

$$\delta_h = \sum_{i=1}^{k} w_{hi} s_i' \delta_i^0,$$

where the weights w_{hi}, are the ones associated with the edges from hidden node *h* to output node *i*, s_i' is the sigmoid derivative at the output node i, and δ_i^0 is the difference between output and target also at this node.

After initialization of the network's weights the expected value of δ_h is zero. In the first phase of learning we are interested in breaking the symmetry of the hidden nodes. They should specialize in the recognition of different features of the input. By making the variance of the backpropagated error larger, each hidden node gets a greater chance of pulling apart from its neighbors. The above equation shows that by making the initialization interval [−α, α] wider, two contradictory forces come into play. On the one hand, the variance of the weights becomes larger, but on the other hand, the expected value of the derivative s_0^k becomes lower. We would like to make δ_h as large as possible, but without making s_i' too low, since weight corrections in the second block of weights are proportional to s_i'. Figure 1 shows the expected values of the derivative at the output nodes, the expected value of the backpropagated error for the hidden nodes as a function of α, and the geometric mean of both values.

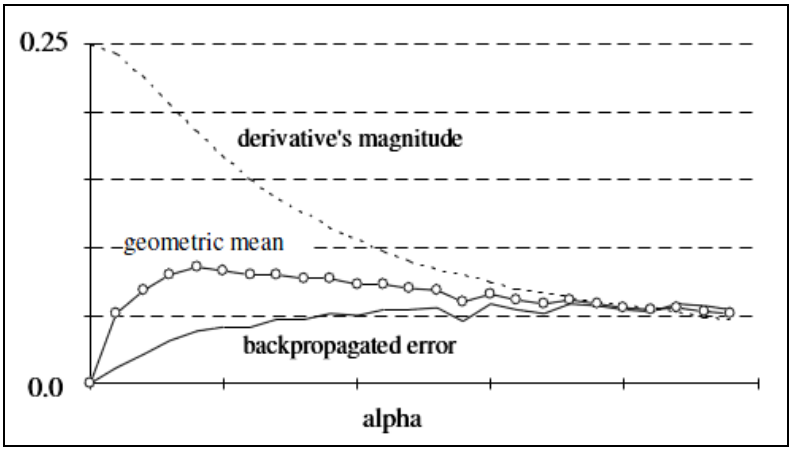

Figure 1. Expected values of the backpropagated error

The figure shows, again, that the expected value of the sigmoid derivative falls slowly with an increasing α, but the value of the backpropagated error is sensitive to small values of α. In the case shown in the figure, any possible choice of α between 0.5 and 1.5 should lead to virtually the same performance. This explains the flat region of possible values for α found in the experiments published in Wessels & Barnard (1992) and Drago & Ridella (1992). Consequently, the best values for α depend on the exact number of input, hidden, and output units, but the learning algorithm should not be very sensitive to the exact α chosen from a certain range of values.

3. CLIPPED DERIVATIVES

One of the factors which lead to slow convergence of gradient descent methods is the small absolute value of the partial derivatives of the error function computed by backpropagation. The derivatives of the sigmoid stored at the computing units can easily approach values near to zero and since several of them can be multiplied in the backpropagation step, the length of the gradient can become too small. A solution to this problem is clipping the derivative of the sigmoid, so that it can never become smaller than a predefined value. We could demand, for example, that $s'(x) \geq 0.01$. In this case the *"derivatives"* stored in the computing units do not correspond exactly to the actual derivative of the sigmoid (except in the regions where the derivative is not too small). However, the partial derivatives have the correct sign and the gradient direction is not significantly affected.

It is also possible to add a small constant " to the derivative and use $s'(x) + \varepsilon$ for the backpropagation step. The net effect of an offset value for the sigmoid derivative is to pull the iteration process out of flat regions of the error function. Once this has happened, backpropagation continues iterating with the exact gradient direction.

It has been shown in many different learning experiments that this kind of heuristic, proposed by Fahlman (1989), among others, can contribute significantly to

accelerate several different variants of the standard backpropagation method (Pfister & Rojas, 1993). Note that the offset term can be implemented very easily when the sigmoid is not computed at the nodes but only read from a table of function values.

The table of derivatives can combine clipping of the sigmoid values with the addition of an offset term, to enhance the values used in the backpropagation step.

4. REDUCING THE NUMBER OF FLOATING-POINT OPERATIONS

Backpropagation is an expensive algorithm because a straightforward implementation is based on floating-point operations. Since all values between 0 and 1 are used, problems of precision and stability arise which are normally avoided by using 32-bit or 64-bit floating-point arithmetic. There are several possibilities to reduce the number of floating-point operations needed.

4.1. Avoiding the computation of the squashing function

If the nonlinear function used at each unit is a sigmoid or the hyperbolic tangent, then an exponential function has to be computed and this requires a sequence of floating-point operations. However, computation of the nonlinearity can be avoided by using tables stored at each unit, in which for an interval $[x_i, x_{i+1}]$ in the real line the corresponding approximation to the sigmoid is stored. A piecewise linear approximation can be used as shown in figure 2, so that the output of the unit is $y = a_i + a_{i+1}(x - x_i)$ when $x_i \leq x < x_{i+1}$ and where $a_1 = s(x_i)$ and $a_{i+1} - s(x_{i+1})$ Computation of the nonlinear function is reduced in this way to a comparison, a table lookup, and an interpolation. Another table holding some values of the sigmoid derivative can be stored at each unit for the backpropagation step. A piecewise linear approximation can also be used in this case. This strategy is used in chips for neural computation in order to avoid using many logic gates.

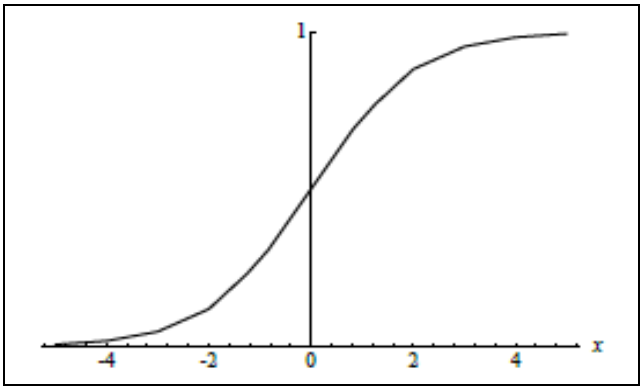

Figure 2. Piecewise linear approximation to the sigmoid

4.2. Avoiding the nonlinear function at the output

In some cases the sigmoid at the output of the network can be eliminated. If the output vectors in the training set are *m*-dimensional vectors of the form $(t_1, \ldots, t_m)$ with $0 < t_i < 1$ for $i = 1, \ldots, m$ then a new training set can be defined with the same input vectors and output vectors of the form $(s^{-1}(y_1), \ldots, s^{-1}(y_m))$, where the function s^{-1} is the inverse of the sigmoid. The sigmoid is eliminated from the output units and the network is trained with standard backpropagation. This strategy will save some operations but its applicability depends on the problem, since the sigmoid is equivalent to a kind of weighting of the output error. The inputs 100 or 1000 produce almost the same sigmoid output, but the two numbers are very different when compared directly.

4.3. Fixed-point arithmetic

Since integer operations can be executed in many processors faster than floating-point operations, and since the outputs of the computing units are values in the interval (0,1) or (−1,1), it is possible to adopt a fixed-point representation and perform all necessary operations with integers. By convention we can define the last 12 bits of a number as its fractional part and the three previous bits as its integer part. Using a sign bit it is possible to represent numbers in the interval (−8,8) with a precision of . Care has to be taken to re-encode the input and output vectors, to define the tables for the sigmoid and its derivatives and to implement the correct arithmetic (which requires a shift after multiplications and tests to avoid overflow). Most of the more popular neural chips implement some form of fixed-point arithmetic.

Some experiments show that in many applications it is enough to reserve 16 bits for the weights and 8 for the coding of the outputs, without affecting the convergence of the learning algorithm (Asanovic & Morgan, 1991). Holt and Baker compared the results produced by networks with floating-point and fixed-point parameters using the Carnegie-Mellon benchmarks (Holt & Baker, 1991). Their results confirmed that a combination of 16-bit and 8-bit coding produces good results. In four of five benchmarks the result of the comparison was *"excellent"* for fixed-point arithmetic and in the other case *"good"*.

Based on these results, groups developing neural computers like the CNAPS of Adaptive Solutions (Hammerstrom, 1990) and SPERT in Berkeley (Asanovic, et al., 1992) decided to stick to 16-bit and 8-bit representations.

Reyneri and Filippi (1991) did more extensive experimentation on this problem and arrived at the conclusion that the necessary word length of the representation depends on the learning constant and the kind of learning algorithm used. This was essentially confirmed by the experiments done by Pfisteron a fixed-point neurocomputer (Pfister, 1995). Standard backpropagation can diverge in some cases when the fixed-point representation includes less than 16 bits. However, modifying the learning algorithm and adapting the learning constant reduced the necessary word length to 14 bits. With the modified algorithm 16 bits were more than enough.

5. DATA DECORRELATION

If the principal axes of the quadratic approximation of the error function are too dissimilar, gradient descent can be slowed down arbitrarily. The solution lies in decorrelating the data set and there is now ample evidence that this preconditioning step is beneficial for the learning algorithm (Holt & Baker, 1991; Silva & Almeida, 1991).

One simple decorrelation strategy consists in using bipolar vectors. We know that the solution regions defined by bipolar data for perceptron are more symmetrically distributed than when binary vectors are used. The same holds for multilayer networks. Pfister showed that convergence of the standard backpropagation algorithm can be improved and that a speedup between 1.91 and 3.53 can be achieved when training networks for several small benchmarks (parity and clustering problems) (Pfister, 1995). The exceptions to this general result are encoded-decode problems in which the data consists of sparse vectors (n-dimensional vectors with a single 1 in a component). In this case binary coding helps to focus on the corrections needed for the relevant weights. But if the data consists of non-sparse vectors, bipolar coding usually works better. If the input data consists of N real vectors $x_1, x_2, \ldots, x_N$ it is usually helpful to center the data around the origin by subtracting the centroid $\hat{x}$ of the distribution, in such a way that the new input data consists of the vectors $x_i - \hat{x}$.

Silva and Almeida have proposed another data decorrelation technique, called Adaptive Data Decorrelation, which they use to precondition data (Silva & Almeida, 1991). A linear layer is used to transform the input vectors, and another to transform the outputs of the hidden units. The linear layer consists of as many output as input units, that is, it only applies a linear transformation to the vectors. Consider an n-dimensional input vector $x = (x_1, x_2, \ldots, x_n)$. Denote the output of the linear layer by $(y_1, y_2, \ldots, y_n)$. The objective is to make the expected value of the correlation coefficient r_{ij} of the i and j output units equal to Kronecker's delta δ_{ij} ,

$$r_{ij} =< y_i y_j >= \delta_{ij}.$$

The expected value is computed over all vectors in the data set.

6. CONCLUSION

The Backpropagation Neural Network (BPNN) is a supervised learning neural network model highly applied in different engineering fields around the globe. Although it is widely implemented in the most practical ANN applications and performed relatively well, it is suffering from a problem of slow convergence and convergence to local minima. This makes Artificial Neural Network's application very challenging when dealing with large problems. From this paper we can conclude that even though several variations to this algorithm have been suggested to improve the performance of BPNN, no one guarantees global optimum solution which is still need to be answered.

REFERENCES:

[1]. Asanovic, K.; Beck, J.; Kingsbury, B. (1992) *SPERT: A VLIW/SIMD Microprocessor for Artificial Neural Network Computations*, International Computer Science Institute, Technical Report, TR–91-072, Berkeley, CA

[2]. Asanovic, K.; Morgan, N. (1991) *Experimental Determination of Precision Requirements for Back-Propagation Training of Artificial Neural Networks*, International Computer Science Institute, Technical Report TR–91-036, Berkeley, CA

[3]. Drago, G.P.; Ridella, S. (1992) *Statistically Controlled Activation Weight Initialization (SCAWI)*, IEEE Transactions on Neural Networks, Vol. 3, No. 4, pp. 627–631

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

[5]. Hammerstrom, D. (1990) *A VLSI Architecture for High-Performance, Low- Cost, On-Chip Learning*, in IEEE 1990, Vol. II, pp. 537–544

[6]. Holt, J.; Baker, T. (1991) *Back Propagation Simulations Using Limited Precision Calculations,* in IEEE 1991, Vol. II, pp. 121–126

[7]. Lisboa, P.G.; Perantonis, S.J. (1991) *Complete Solution of the Local Minima in the XOR Problem*, Network – Computation in Neural Systems, Vol. 2, No. 1, pp. 119–124

[8]. Pfister, M. (1995) *Hybrid Learning Algorithms for Neural Networks*, PhD Thesis, Free University Berlin

[9]. Pfister, M.; Rojas, R. (1993) *Speeding-up Backpropagation – A Comparison of Orthogonal Techniques*, International Joint Conference on Neural Networks, Nagoya, Japan, pp. 517–523

[10]. Reyneri, L.; Filippi, E. (1991) *An Analysis on the Performance of Silicon Implementations of Backpropagation Algorithms for Artificial Neural Networks*, IEEE Transactions on Computers, Vol. 40, No. 12, pp. 1380–1389

[11]. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. (1986) *Learning Internal Representations by error Propagation,* J. Parallel Distributed Processing: Explorations in the Microstructure of Cognition

[12]. Silva, F.; Almeida, L. (1991) *Speeding-up Backpropagation by Data Orthonormalization*, in Kohonen et al. 1991, pp. 1503–1506

[13]. Wessels, L.; Barnard, E. (1992) *Avoiding False Local Minima by Proper Initialization of Connections*, IEEE Transactions on Neural Networks, Vol. 3, No. 6, pp. 899–905