Adaptive Learning Rate Selection for Backpropagation Networks

Jayathi Janakiraman & Vasant Honavar [*]
Department of Computer Science
226 Atanasoff Hall
Iowa State University
Ames Iowa 50011-1040

September 11, 1993

Abstract

Backpropagation is a supervised learning algorithm for training multi-layer neural networks for function approximation and pattern classification by minimizing a suitably defined error metric (e.g., the mean square error between the desired and actual outputs of the network for a training set) using gradient descent. It does this by calculating the partial derivative of the overall error and changing each weight by a small amount (determined by the learning rate) in a direction that is expected to reduce the error. Despite its success on a number of real-world problems, backpropagation can be very slow (it requires hundreds of passes (epochs) through the training set). Also, its performance is extremely sensitive to the choice of parameters such as the learning rate.

The mathematical considerations that go into the derivation of backpropagation require that the learning rate be as small as possible. On the other hand, in order reduce the number training epochs required to learn the desired input-output mapping, it is desirable to make the weight changes as large as possible without causing the error to increase.

Carefully designed experiments with a number of different data sets show that the number of epochs required for learning is very sensitive to the choice of the (constant) learning rate parameter. The learning rate that works best for one data set may not work so well for a different data set. Furthermore, our simulations with the iris data set clearly demonstrate that the *best* learning rate for the same problem can be rather sensitive to the particular random choice of training examples as well as the initial weight settings! It is therefore desirable to have an algorithm that can change the learning rate dynamically so that it is close to optimal (in terms of reducing the error as much as possible given the

[*]This research was partially supported by the Iowa State University College of Liberal Arts and Sciences. Some of the resuls in this paper were presented in SPIE '93 conference at Orlando, Florida and the World Congress on Neural Networks, Portland, Oregon. The authors are grateful to Dr. Michael Pazzani for making the electronic versions of the data sets used in this study available through the University of California (Irvine) machine learning repository.

local gradient information) at each epoch (thereby eliminating the reliance on guesswork in the choice of the learning rate).

Several authors have proposed methods for adaptively adjusting the learning rate based on certain assumptions about the shape of the error surface (e.g., quadratic) and/or estimation of higher order derivatives of the error with respect to the weights at a given point in the weight space. The primary disadvantage of such methods is that the estimation of second order derivatives is not only computationally expensive but also prone to inaccuracy due to the approximation of derivatives by discrete differences.

In this paper we propose and evaluate a heuristically motivated method for adaptive modification of the learning rate in backpropagation that does not require the estimation of higher order derivatives. We present a modified version of the Backpropagation learning algorithm which uses a simple heuristic to come up with a learning parameter value at each epoch.

We present numerous simulations on real-world data sets, using our modified algorithm. We compare these results with results obtained with standard backpropagation learning algorithm, and also various modifications of the standard backpropagation algorithm (e.g., flat-spot elimination methods) that have been discussed in the literature. Our simulation results suggest that the adaptive learning rate modification helps substantially speed up the convergence of backpropagation algorithm. Furthermore, it makes the initial choice of the learning rate fairly unimportant as our method allows the learning rate to change and settle at a reasonable value for the specific problem.

1 Introduction

Neural network problems have been applied to many problems since powerful learning algorithms such as error-backpropogation learning were proposed. A broad definition of a learning algorithm is as follows:
A program is said to learn over time, if it's performance on a given task or a range of tasks, improves(with respect to some performance metric) as a result of experience.
The most common performance metrics are

- Efficiency of Learning : which is measured as the number of times the entire training set needs to be presented such that the network learns the input-output mapping to some desired accuracy. (learning speed).
- Efficacy of Learning: This is with respect to the generalization capability of thenetwork and measures how well the network responds to patterns it has not seen during the current test-phase.

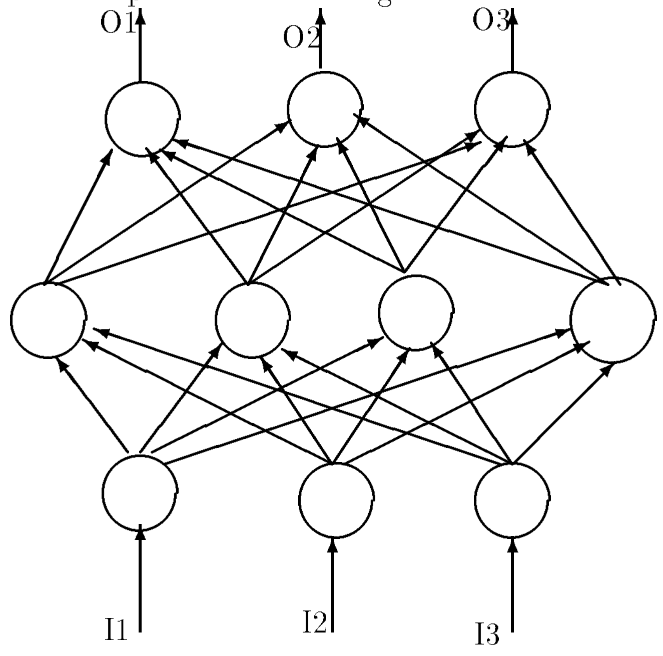
Backpropogation suffers from practical problems such as local minima and sometimes excrutiatingly slow convergence. This paper addresses the problem of speeding up the convergence of backpropagation as well as the related problem of selection of suitable learning parameters.

Two factors contributing to the slow convergence of the backpropagation are the small step-size and flat-spots in the sigmoid function where the derivative is near zero[Fahlman, 1988, Fahlman & Lebiere, 1991]. In [Fahlman, 1988], several methods to combat the flat-spot problem are examined; adding an offset to the derivative of the sigmoid function to prevent it from approaching zero gave the best results. Recently, a different solution for the flat-spot problem has been proposed and experimentally evaluated[Balakrishnan & Honavar, 1992]. A common modification to the backpropogation algorithm which helps alleviate the step-size problem is to use a momentum term[Rumelhart et al., 1986]. In this paper we have tried a different approach to solve the step-size problem. Our approach is to dynamically adjust the learning rate at the end of each epoch, based on a heuristically guided local search.

The goal of the current study is to develop and evaluate a learning rule which combines the modifications cited above to alleviate the step-size and flat-spots problem. We wish to determine which one of the various combinations leads to the maximum speed-up. The remainder of the paper is organized as follows: Section 2 gives a brief review of the backpropogation algorithm; Section 3 explains the Step-Size and the Flat-Spot problems and the proposed solutions; Section 4 explains the exprimental methodology and the Data-sets used; Section 5 discusses the experiment results and their implications; Section 6 briefly addresses the computational complexity of the proposed modification; Section 7 concludes with a summary and directions for future research.

2 Backpropagation Learning Algorithm

The backpropagation learning algorithm, or generalized delta rule [Rumelhart et al., 1986], is a supervised learning algorithm for feedforward neural networks. A feedforward neural network consists of two or more layers of processing units. The lowest layer is the input layer, the uppermost layer is the output layer and the intermediate layers, if any, are called hidden layers. Each non-input unit has a weighted connection from every unit in the layer directly below it.



2.1 Steps in the Training Algorithm

The back-propogation training algorithm is an iterative gradient-descent algorithm designed to minimize the mean square error between the actual output of a multilayer feed-forward perceptron and the desired output. It requires a continuous differentiable output function. The sigmoid function shown below, is a very common choice, and is the one used in this study.

$$o_j = \frac{1}{1 + e^{-net_j}} \tag{1}$$

Input units are activated directly by application of an input pattern. Each unit j in subsequent layers calculates its weighted sum, or net input, of the N inputs from the layer below:

$$net_j = \sum_{i=1}^{N} w_{ji} o_i. \tag{2}$$

- **Step 1. Initialize Weights and Node thresholds:**
  Set all weights and node offsets to small random values. In the implementation that we have, the weights are set randomly with values between -1.0 and +1.0.

- **Step 2. Present Inputs and desired Outputs:**
  Present a continuous valued input vector x_0, x_1, ..., x_{n-1}. Also specify the desired or

target outputs for these sets of inputs as t_0, ..., t_l. The input could be new on each trial or samples from a training set could be presented cyclically until weights stabilize.

- **Step 3. Calculate actual Outputs:**
  Use the sigmoid output function given above to calculate the actual outputs o_0, ..., o_l. This is the forward pass.

- **Step 4. Adapt weights:**
  Use a recursive algorithm starting at the output nodes and working back to the first hidden layer by
  $$w_{ij}(t+1) = w_{ij}(t) + \eta o_i \delta_j. \tag{3}$$
  In this equation $w_{ij}(t)$ is the weight from hidden node i or from an input to node j at time t, η is the gain term, and δ_j is an error term for node j. The error term is calculated as follows:
  $$\delta_j = \begin{cases} o'_j\,(t_j - o_j) & \text{for output units } j \\ o'_j \sum\limits_k \delta_k w_{kj} & \text{for hidden units } j \end{cases} \tag{4}$$
  o'_j is the derivative of the output function evaluated at the current net_j value. For the sigmoid function, it is given by
  $$o'_j = o_j(1 - o_j).$$
  This is called the backward pass and has the same computational complexity as the forward pass. Internal node thresholds are adapted in a similar manner by assuming they are connection weights on links from auxiliary constant-valued inputs. Convergence is sometimes faster if a momentum term is added and weight changes are smoothed by
  $$w_{ij}(t+1) = w_{ij}(t) + \eta\delta_j o_i \ + \alpha(w_{ij}(t) - w_{ij}(t-1)), where 0 < \alpha < 1. \tag{5}$$

- **Step 5. Repeat by Going to Step 2.**

The algorithm defined here is a learning procedure which could, in principle, find a set of weights that correspond to a local minimum of the mean squared error function (which hopefully also yields a sufficiently accurate approximation of an unknown input-output mapping based on the training samples.

3 Modifications to Back-Propogation

Two major problems have been identified which contribute to the slow convergence of back-propagation. This section explains these problems and presents the modifications to back-propagation which we have used in this study.

3.1 The Step Size Problem

Back-propagation is guaranteed to find a local minimum of the error surface, but only for infinitesimal weight changes. The magnitude of the weight changes, or step size, is determined by the learning rate η. In practice, we would like to take large steps whenever possible to get faster convergence. If the step size is set too large, however, the system error tends to oscillate, never reaching a minimum. The question, thus, is how to reconcile the apparently contradictory imperatives of stability and efficiency. Basically, there are two ways to approach the problem of speeding up back-propagation: analytically incorporating more information about the error surface into the algorithm or using heuristic methods that work most of the time. In what follows, we discuss some approaches to alleviating the step-size problem that have been discussed in the literature before going onto our proposed solution.

3.1.1 Higher-Order Derivatives

Standard back-propagation uses only the first partial derivative of the error surface at the current location in the weight space. To determine the appropriate step size, it would be useful to know something about the curvature of the error surface, i.e. the second partial derivative, at the current point in weight space. Many techniques have been proposed which explicitly calculate an approximation to the second derivative, and use this information to determine the appropriate step size [Becker & leCun, 1988, Fahlman, 1988]. These methods are quite computationally expensive compared to back-propagation. Also, when using an approximation to the second order derivative, any difference between the approximation and the true derivative introduces some error into the weight update and it is well-known that any numerical estimation of higher-order derivatives is more error-prone than the lower order derivatives [Press et al., 1988].

3.1.2 Momentum

The use of a momentum term α[Rumelhart et al., 1986] — a constant which determines the effect of past weight changes on the current direction of movement in weight space often helps alleviate the step size problem. The momentum term α simply adds a fraction of the previous weight change to the current weight adjustment value. This tends to filter out high frequency oscillations in the error surface. The effect of the momentum term can be explained by invoking the physical analogy involving a moving object: once the weight start changing in a certain direction, the update values accumulate so th weight vector tends to continue to change in the same direction. The incorporation of the momentum term yields the following rule for adjusting the weight from unit i to unit j at time t is:

$$\Delta w_{ji}(t) = \eta o_i \delta_j + \alpha \Delta w_{ji}(t-1) \tag{6}$$

where $0 \leq \alpha \leq 1$ is a constant value called the momentum rate.

3.1.3 Intuitive Heuristics

Other heuristics cited are, varying the learning rate for each weight connection, controlling the sequence in which training samples are presented, using different error metrics, and controlling problem difficulty.

Of these, the momentum strategy is probably the most widely used. Note that the difficulties involved in the choice of a good momentum setting is are analogous to those encountered in choosing a good learning rate for a given data set.

3.1.4 Dynamic Learning Rate Using Heuristically Guided Local Search

We wanted to eliminate the guesswork involved in choosing a learning rate for each problem and at the same time resolve the conflicting requirements of

- infinetesimal learning rate required by the mathematics of gradient-descent using only first-order derivative information about the error surface and
- the desirability of picking a learning rate yielding the maximum possible reduction in error at each step.

The solution that we came up is a heuristically motivated technique for changing the learning rate η, dynamically for every epoch. Assuming a roughly concave error surface, we seek to change the learning rate dynamically so that the weight changes approximate the largest permissible given the local gradient information.

- **Heuristic I:**
  We considered changing the η value as follows:
  Looking at the simulation results reported for the data-sets under consideration, we came up with an allowable range of η values and also a starting η value. We then ran some simulation runs on some real-life data sets. The heuristic which we applied was to double the η value whenever the current error was less than the previous error and half the η value when the error increased or remained the same. The simulation results showed that the application of the above heuristic did cause a marginal improvement in the learning rate for the runs which converged, but there were quite a number of runs which failed to converge.

  Looking at the error curve for these runs, it was seen that the error curve decreases and nearly reaches an acceptable minimum; but before it actually settles down, there is a sharp increase. We attributed this to the fact that on coming very close to the local minima, the η change was inappropriate, which resulted in oscillations and thus the prevention of convergence.

- **Heuristic II:**
  The experience with the heuristic discribed above prompted us to investigate a minor modification. We decided that we would allow the η value to adapt itself dynamically; but once the error reached a specific low value (we fixed it as 0.03 after looking at the error-curves for the runs which did not converge), we would turn-off the heuristic and let the learning proceed as in the standard back-prop rule.
  This modification did not give us good results. We attributed this to the fact that the η value to which it may have settled may not have been a good value for the particular weight settings.

- **Heuristic III: Three-comparison technique:**
  We tried another approach by doing a localised search for a good value of the learning rate at each epoch. Initially, we fixed the upper and lower limits within which η could vary(these limits were chosen based on published results of many experiments using back-propagation as well as our own experiments). Instead of just comparing the previous error

value and the present error value, we decided to compute the total error per epoch at three different η points; one at the current η value, the second one at double the current η value and lastly one at half the current η value. Looking at the total error accumulated over the entire epoch for the three different η values, we pick the one which yields weight changes that correspond to the lowest error. We set this value of η as the current η value. This method would ensure that we filter out some of the high-frequency curves on the error surface, by spreading out or area of search and not confining to just the previous error value as in Heuristic I.

Simulation runs with this approach gave us the best results, and so we decided to explore this more thoroughly.

As a reference against which we could evaluate the heuristic proposed above, we tried computing the *quasi-optimal* η value after each epoch. (The quasi-optimal η calculation detailed below aims to approximate the value of η that is optimal in the sense of producing weight changes that reduce the error by the largest amount given only the local gradient of the error surface).

- **Quasi-Optimal η value calculation:**
  As before we fixed the limits within which the η could vary. We also started off with the starting η value as the average of the limits mentioned above. The offset value was set to the absolute difference between the starting η value and the high η. We then used **Heuristic III (Three-comparison technique)** iteratively to compute the best η value. The best η value in the previous iteration is used as the starting η value in the next iteration. The new, high and low η are computed by adding and subtracting the new offset value to the starting η value, where the new offset is one half of the previous offset. This cycle is repeated until the offset reaches a small value, fixed as 0.01 for our experiments. Thus at each epoch, we choose the best η value using a geometric search taht successively halves the range of η values to be searched for locating the optimal η value. We call it the best η chosen by this process $quasi-optimal$ because this method searches for a real value within an interval by examining a discrete set of points , so we can only hope to find a reasonable approximation to the optimal η(assuming that the error is a smooth function of the weights — a reasonable assumption given the use of the sigmoid function at each node).

3.2 The Flat Spot Problem

During standard back-propagation training, output units may to get stuck in the wrong state for some training patterns. This is due to the properties of the sigmoid function. The output of unit i is taken as $o_i = f(net_i)$, where $f()$ is the sigmoid function. The derivative of the sigmoid function at point o_i, $f'(net_i)|_{o_i}$, is given by $o_i(1 - o_i)$, which approaches 0.0 as o_i approaches either 1.0 or 0.0, i.e. the sigmoid function has flat spots near one and zero. Since the error term, δ_i, for output unit i is the product of this sigmoid-prime function and the difference between t_i and o_i, the error signal for an output unit that is quite far from its target value will be small, due to $f'(net_i)|_{o_i}$ approaching zero. Thus the weight changes in spite of the large $(t_i - o_i)$. This may cause the unit to be "stuck" in the flat spot for several iterations.

One solution to the flat spots problem is to ensure that the sigmoid-prime function does not approache zero. As proposed in [Fahlman, 1988], this can be accomplished by simply adding a constant 0.1 to the sigmoid-prime function before using it to scale the error. This keeps the weight changes for an output unit that is far from its target value from approaching zero; the weight changes for units near their targets still reach zero due to the proximity of the output to the target value. Significant speedups are reported in [Fahlman, 1988] using this sigmoid-prime with offset function. The resulting weight update rule is the same as equation (7), but we use $o'_k = o_k(1 - o_k) + os$ in equation (7) to calculate the error δ_k for any unit k, where $os = 0.0$ or 0.1. An alternative solution to the flat spots problem has been explored in [Balakrishnan & Honavar, 1992].

4 Experimental methodology

The primary objective of the experiments discussed in this section are to compare the performance of the adaptive *eta* modification to the back-propagation proposed in this paper witha number of variants of standard back-propagation(e.g, adding momentum, flatspot elimination etc) on a variety of datasets which have been used by other researchers for experiments reported in the literature [Yang & Honavar, 1991, Shavlik et al., 1991].

4.1 Data-Sets used

The data sets chosen are listed below along with the network architecture used and a description of the problem. The three numbers following the data set name give the size of each layer of the network, starting at the input layer, e.g. 8–3–8 indicates 8 input units, 3 hidden units and 8 output units. In all the experiments involving real-world datasets(i.e Iris, Soybean, Audiology and Chess), two thirds of the dataset was chosen at random for training and the remainder was used as a test set to study generalization; the encoding of the dataset as well as the network architecture(i.e, the number of hidden units) was chosen to correspond exactly to the values reported in [Yang & Honavar, 1991].

8-bit Encoder: 8–3–8, One of the eight input bits is turned on, and the system must learn to turn on the corresponding output bit. Since there are 8 input/output pairs and $\log_2(8) = 3$ hidden units, the representations used by the network on the hidden layer must be very efficient.

10-bit Encoder: 10–5–10, One of the ten input bits is turned on, and the system must learn to turn on the corresponding output bit. This problem is similar to the 8-bit encoder but is easier in some sense because the number of hidden units is greater than $\log_2(10)$.

Iris Classification: 22–4–3, This data set is a real-world classification problem. It consists of 150 examples of iris plant samples which need to be classified into one of three species. Each example is a distributed representation of four plant attributes and the corresponding target output, where each output bit represents a diferent species. It is exactly the same data set used by [Yang & Honavar, 1991]. Two thirds of the data was chosen randomly as the training set, and the other one third was used as the test set to check the generalization performance of the rule.

Soybean Classification: 208–23–17, This data set is also a real-world classification problem. It consists of 289 examples of soybean plants with one of 17 diseases. each input pattern is a 208-bit representation of the 35 nominal valued attributes describing the plant. The output pattern isa 17-bit string with one bit set corresponding to the correct disease diagnosis. This is the same data set as used by [Shavlik et al., 1991].

Audiology Data: 87–11-24, It consists of 200 examples of the various disorders associated with the ear. It is from the Baylor College of Medicine. Since there were 24 categories possible, we had 24 ouput nodes. The number of hidden units was picked up from [Yang & Honavar, 1991] so that we could make direct comparisons.

Chess Data: 73–8–2, This data set consists of 3196 examples of a "king and rook vs king and pawn" end game which can be classified into two categories - **win** and **not win**. Each example has 36 attributes(of which one is three-valued and the rest are two-valued). Thus, the original data-set was encoded with 73 input nodes and 2 output nodes. The number of hidden nodes were used as reported for standard BP in [Yang & Honavar, 1991]. Since the data-set was very large for this case, we randomly chose 591 examples from the input set and again randomly partioned these 591 examples into two-thirds for the training set and one-third for the testing set.

4.2 Details of Training and Testing

Criteria for Successful learning: There are two criteria which we could use to determine when learning is complete.

- Threshold and margin metric [Fahlman, 1988] - The network is said to have correctly classified a training pattern if the actual network output is within a specified threshold for each of the components of the output vector.
- Highest output metric [Shavlik et al., 1991] - Count an example as correctly classified if the node with the highest output corresponds to the correct category. Learning is said to be complete when a high percentage of the training examples are correctly classified.

In our experiments, we have used the second metric with 99% or greater classification accuracy on the training set as the criterion for termination of the learning procedure.

Choice of Parameters for Experiments: Standard backprop is rather slow. Exhaustively scanning the 3-dimensional space defined by η, α, and weight-range is very time-consuming. Therefore, the choice was made as follows:
We used the parameter-setting of the best-average case results reported in

[Parekh & Balakrishnan & Honavar, 1992] for the Iris and Soybean data-sets and the Encoder-Decoder problems. For the Audiology and Chess data sets, we ran simulations for $\eta = 0.25, 0.5, 0.8, 0.9, 1.0, 1.25$ and $\alpha = 0.25, 0.5, 0.80.9$. The best-average case combination was then taken for our experiments.

For our heuristic, we had to come up with the range of η values which we would allow, and also the starting η value. In order to determine the η range, we initially started with the range being 0.25 - 2.00 and starting η value as 1.00. Trial runs were made and the η values which were being used in each epoch was observed. The average number of epochs using the above range was compared with the one got using 0.25 - 5.75 as the η range. It was observed that for all the data sets under consideration, the initial range sufficed and there was no improvement using a higher η value range.
Regarding the momentum term, we reasoned that since it had been initially introduced to produce the same effect on the step-size problem, we could do away with it. In order to substantiate our guesses, we ran experiments with non-zero values of α (0.25, 0.50, 0.80, 0.90) and the modified rule. Results for the best combination have been reported for all the data-sets.

Reporting Learning Speed: For all the data sets 100 trials were run. For the real-valued datasets, we randomly chose two-thirds of the data as our learning file. With each of the learning file we conducted 10 runs with variable initial weight settings. We performed the whole process 10 times, giving us 100 trial runs. The average of the runs which converged [trials which converged implies either it had satisfied the training metric or had exceeded the maximum number of epochs for the data set used] is reported. For the heuristic proposed, we have reported the average value got with $\alpha = 0.0$ and also the best η_{dyn} and η combination.
The learning speed is reported in terms of the mean number of epochs(and the corresponding standard deviation) taken for learning the training set(this does not include the runs that failed to converge). An **epoch** is defined as a single presentation of each of the I/O patterns in the training set.

5 Experimental Results

In this section, we present our experimental data to address the goals of our study. First we show the various interactions observed between the three modifications for all the data sets. Next we will examine Table to see how our proposed heuristic **Three-Comparison** technique performs against the **Quasi-Optimal** solution.

In the "Rule" column, BP denotes the original back-propagation rule using only the learning rate η but no momentum term. BP + x denotes the backpropagation rule with the modification x added, where "m" stands for the momentum modification, modification, "d" for the dynamic learning-rate, "op" for the optimal learning-rate, "o" the sigmoid-prime offset modification. The η column contains the initial setting of η in the dynamic η case and the fixed (experimentally determined best) value in rest of the cases. The percent speedup is calculated relative to BP rule with momentum added (we don't report results for BP without the momentum term because there is general consensus in the literature that the addition of momentum term substantially speeds up the convergence of BP [Fahlman, 1988]).

5.1 Modification Interactions

Toy Data Sets:

Rule	η	α	os	Ave. Epochs	% Converged	Test %	Speedup %
BP + m	1.25	0.8	0.0	43 ± 12.21	100	100	0
BP + m,o	1.25	0.8	0.1	24.0 ± 5.95	100	100	44.2
BP + o	1.50	0.0	0.1	86.58 ± 20.55	100	100	-101.3
BP + d,o	1.00	0.0	0.1	53.80 ± 12.47	100	100	-25.1
BP + d,m,o	1.00	0.80	0.1	28.0 ± 0.0	100	100	34.9
BP + op,m,o	1.00	0.0	0.1	79.70 ± 25.80	100	97.40	-85.3

Table 1: Modification Interactions for 8-bit Encoder

Rule	η	α	os	Ave. Epochs	% Converged	Test %	S peedup
BP + m	1.25	0.8	0.0	39.0 ± 11.55	100	100	0
BP + m,o	1.25	0.8	0.1	16.89 ± 3.70	100	100	56.7
BP + o	1.50	0.0	0.1	41.53 ± 8.83	100	100	-32.1
BP + d,o	1.0	0.0	0.1	27.60 ± 4.78	100	100	29.2
BP + d,m,o	1.0	0.8	0.1	19.0 ± 0.0	100	100	51.3
BP + op,m,o	1.0	0.0	0.1	48.40 ± 16.92	99	100	-24.1

Table 2: Modification Interactions for 10-bit Encoder

We have examined two toy data sets. The points to notice in the table are the % convergence, the classification accuracy of the test data and the speedup measured against the BP algorithm.
For the toy data sets, we see that there is no problem with respect to convergence and also the classification accuracy, since it is 100% over all modifications. We will therefore focus on the speedup column.

We see that the best combination for the toy data sets is the one with the constant η, α and the offset. The toy data sets perform poorly with our proposed heuristic, compared to the best combination mentioned above. Its performance degrades to the value obtained by using the ordinary BP algorithm. There is a significant improvement in the performance, if we add a momentum term to the η_{dyn} rule. Thus, our heuristic for adaptive choice of learning rates offers little advantage over standard methods in the case of toy datasets. Comparing the rows(3 & 4) in Table 1, we see that without the momentum term, the encoder-decoder problems favor a high η value. Also these rows suggest that the choice of the starting η value is not very crucial for the **three- comparison"** technique. With an approximate choice of the starting η_{dyn} value , the network is able to settle down to a good η value.

Real-World Data Sets:

Rule	η	α	os	Ave. Epochs	% Converged	Test %	Speedup %
BP + m	1.0	0.5	0.0	41.37 ± 69.05	87	91.8	0
BP + m,o	1.0	0.5	0.1	27.21 ± 19.11	99	89.56	34.2
BP + o	0.8	0.0	0.1	17.73 ± 13.0	100	93.33	57.1
BP + d,o	1.0	0.0	0.1	20.36 ± 16.09	100	90.78	50.8
BP + d,m,o	1.0	0.5	0.1	21.43 ± 16.85	100	90.20	48.2
BP + op,m,o	1.0	0.0	0.1	22.72 ± 17.42	100	90.64	45.1

Table 3: Modification Interactions for the Iris data set

Rule	η	α	os	Ave. Epochs	% Converged	Test %	Speedup %
BP + m	0.25	0.9	0.0	285.83 ± 119.64	39	76.33	0
BP + m,o	0.25	0.9	0.1	20.77 ± 3.95	100	72.15	92.7
BP + o	1.5	0	0.1	22.84 ± 3.15	100	74.78	92.1
BP + d,o	1.0	0.0	0.1	23.90 ± 5.47	100	74.30	91.6
BP + d,m,o	1.0	0.5	0.1	22.40 ± 4.91	100	74.57	92.2
BP + op,m,o	1.0	0.0	0.1	24.71 ± 14.55	100	74.30	91.4

Table 4: Modification Interactions for the Audiology data set

Looking at the convergence percentage and the classification accuracy, we find that they remain nearly constant over all the modifications. Therefore our main focus will be on the speedup measured relative to standard backpropagation (that is, BP with momentum, but no offset).

Observations:

– The first observation to make is that all modifications have moderate to substantial speedup over standard BP with momentum.

– The best η and α values are highly problem dependent if a fixed η is used. It can also be observed that a high η_{cons} value usually works well with a low α term. (Iris: $\eta = 1.0$, $\alpha = 0.5$; Audiology: $\eta = 0.25$, $\alpha = 0.9$).

– Adding an offset to the BP helps reduce the number of learning epochs. Exception is the Chess dataset. The most significant speedup can be seen by observing the Audiology data in Table 3. (speedup = 92.7 %).

– For the real-world data sets, BP with η_{cons}, α, and the offset works the best, with the chess dataset being an exception.

– Comparing the rows 2 & 4 in each table, we see that having only a η_{dyn} term (without the momentum term) and an offset, we are able to come very close to the performance of the case having a η_{cons} term, α, and an offset. This is a significant result because it can help simplify the choice of parameters while ensuring performance comparable to the best of the other methods: The momentum term α is not used; Since we will allow η to change (increase or decrease) dynamically for each epoch presentation, it makes the initial choice of the learning rate η fairly unimportant.

– Looking at rows 4 & 5, we see that adding a momentum term with the η_{dyn} rule does not help in increasing the speedup by a significant value as compared to just having the η_{dyn}; this suggests that we can do away with the momentum term(α) if we have a dynamically changing η.

– Looking at rows 4 & 6 in tables 2,3,4 we can observe that the performance results of the **three-comparison** technique and the **quasi-optimal** technique are comparable for real-world datasets.

Rule	η	α	os	Ave. Epochs	% Converged	Test %	Speedup %
BP + m	1.0	0.5	0.1	16.0 ± 5.23	100	93.1	0
BP + m,o	1.0	0.5	0.1	8.0 ± 1.22	100	92.5	50
BP + o	1.25	0.0	0.1	8.46 ± 1.31	100	91.94	47.1
BP + d,o	1.0	0.0	0.1	8.56 ± 1.61	100	91.31	46.5
BP + d,m,o	1.0	0.5	0.1	8.00 ± 1.21	100	90.7	50
BP + op,m,o	1.0	0.0	0.1	8.38 ± 1.38	100	92.96	47.63

Table 5: Modification Interactions for Soybean data set

6 Complexity Analysis of the Adaptive Learning Rate Modification

In this section we will examine the complexity of our heuristic proposed. We have shown through the simulation runs that having an η_{dyn} with the offset but without the momentum term yields performance comparable to that obtained with η_{cons}, α and the offset.

6.1 Time Complexity:

The change in weight is calculated as follows:

$$\Delta w_{ji} = \eta o_i \delta_j. \tag{7}$$

At each point we compute three different Δw; one with the η_{curr}, second with $2 * \eta_{curr}$ and lastly with $0.5 * \eta_{curr}$. Thus, the processing time requirements for each pass through the data set is increased by a small multiplicative constant (3.0) but this is offset by the reduction in the total number of passes (epochs) through the training data.

6.2 Space Complexity:

Instead of one set of arrays, we have three sets of the arrays holding the Δw, δ and output values. Thus, the storage requirements are increased by a small multiplicative constant (3.0).

7 Directions for Future Research

The comparable performance of the adaptive selection of the learning-rate parameter with that of a good approximation of the optimal learning algorithm, leads us to believe that we could pursue extensions of our method to other gradient-descent based learning algorithms for neural networks (e.g., backpropagation for recurrent networks that can be trained to recognize temporal and spatio-temporal patterns; stochastic gradient-descent algorithms).

References

[Balakrishnan & Honavar, 1992] Balakrishnan, K., & Honavar, V., "Improving Convergence of Backpropagation by Handling Flat-spots in the Output Layer", In: Proceedings of the Second International Conference on Artificial Neural Networks, Brighton, U.K., 1992.

[Parekh & Balakrishnan & Honavar, 1992] Parekh, R., & Balakrish nan, K., & Honavar, V., "An Empirical Comparison of Flat-Spot Elimination Tec hniques in Back-Propogation Networks", in*Proceedings of the Conference on Artificial Neural Networks*, Houston, TX, 1992.

[Becker & leCun, 1988] Becker, S. and leCun, Y., "Improving the Convergence of Back-Propagation Learning with Second-Order Methods", in *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann, 1988.

[Fahlman, 1988] Fahlman, Scott E., "An Empirical Study of Learnin g Speed in Back-Propagation Networks",Technical Report CMU-CS-88-162, Carnegie Mellon University, Computer Science Department,Pittsburgh, PA, 1988.

[Fahlman & Lebiere, 1991] Fahlman, Scott E., and Lebiere, Christi an, "The Cascade-Correlation Learning Architecture", Technical Report CMU-CS-90-100, Carnegie Mellon University, Computer Science Department,Pittsburgh, PA, 1991.

[Knight, 1989] Knight, Kevin, "A Gentle Introduction to Subsymbolic Computation: Connectionism for the A. I. Researcher", Technical Report CMU-CS-89-150, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1989.

[Press et al., 1988] Press, W. H., Flannery, B. P., Teukolsky, S. A., & Vetterling, W. T., *Numerical Recipies in C*, Cambridge University Press, New York, NY, 1988.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Wil liams, R. J. "Learning Internal Representations by Error Propagation", *Parallel Distributed Processing, Vol. I*, Rumelhart D. E., and McClelland, J. L. (Eds.), MIT Press, Cambridge, MA, 1986.

[Shavlik et al., 1991] Shavlik, Jude W., Mooney, Raymond J, and Towell, Geoffrey G., "Symbolic and Neural Learning Algorithms: An Experimental Comparison", *Machine Learning*, Volume 6, 1991.

[Yang & Honavar, 1991] Yang, Jihoon, and Honavar, Vasant, "Exper iments with the Cascade-Correlation Algorithm",Proceedings of the 4th UNB Artificial Intelligence Symposium, Fredericton, NB, Canada, 1991.

.

Rule	η	α	os	Ave. Epochs	% Converged	Test %	Speedup %
BP + m	1.00	0.0	0.0	34.70 $\pm$ 10.15	100	94.90	0
BP + m,o	1.00	0.0	0.1	47.07 $\pm$ 29.0	100	94.70	-37.5
BP + o	0.25	0.0	0.1	30.0 $\pm$ 0.0	100	94.00	13.5
BP + d,o	1.0	0.0	0.1	28.35 $\pm$ 9.09	100	94.01	18.3
BP + d,m,o	1.0	0.8	0.1	39.50 $\pm$ 12.03	100	94.71	-13.8
BP + op,m,o	1.0	0.0	0.1	78.47 $\pm$ 78.63	100	94.16	-126.13

Table 6: Modification Interactions for the Chess data set