AN EMPIRICAL ANALYSIS OF THE EXPECTED SOURCE VALUES RULE

Richard Spartz & Vasant Honavar
Department of Computer Science
Iowa State University
Ames, IA 50010

ABSTRACT

Despite its notoriously slow learning time, back-propagation (BP) is one of the most widely used neural network training algorithms. Two major reasons for this slow convergence are the step size problem nd the flat spot problem [Fahlman, 1988]. In [Samad, 1991] a simple modification, the expected source values (ESV) rule, is proposed for speeding up the BP algorithm. We have extended the ESV rule by coupling it with a flat-spot removal strategy presented in [Fahlman, 1988], as well as incorporating a momentum term to combat the step size problem. The resulting rule has shown dramatically improved learning time over standard BP, measured in training epochs.

Two versions of the ESV modification are mentioned in [Samad, 1991], on-demand and up-front. but simulation results are given mostly for the on-demand case. Our results indicate that the up-front version works somewhat better than the on-demand version in terms of learning speed. We have also analyzed the interactions between the three modifications as they are used in various combinations.

1 INTRODUCTION

The generalized delta rule [Rumelhart et al., 1986], also known as the back-propagation (BP) learning algorithm, is one of the most popular multilayer neural network training algorithms. Although mathematically sound, the learning rule suffers from sometimes excrutiatingly slow convergence. Two factors contributing to the slow convergence of BP are the step size problem and flat spots in the sigmoid function where the derivative is near zero [Fahlman, 1988]. In section 3 we define these problems and give two modifications to BP to help alleviate them. Section 3 also presents the expected source values (ESV) modification.

The goals of the current study are to determine which of two variations of the ESV rule gives the best overall performance and to evaluate the three modifications of BP as they are used in all possible combinations.

Section 2 gives a brief review of the BP algorithm. Section 3 explains the step size and flat spot problems and the proposed solutions along with the ESV rule. Section 4 explains the data sets and experimental method used to evaluate the rules. Section 5 presents our simulation results and section 6 gives the conclusions drawn from this study, as well as some possible directions for future work.

2 THE BACK PROPAGATION ALGORITHM

The BP algorithm [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 input layer, the output layer, and possibly intermediate layers called hidden layers. Each non-input unit has a weighted connection from every unit in the layer directly below it.

2.1 Training Paradigm

A learning task, or data set, consists of p input-output pattern pairs; each pattern is a vector of numeric, often binary, values. Each input pattern is the same size as the input layer of the network, and likewise for output patterns. The learning rule must adjust the weights so the network produces the output pattern when presented with the input pattern for each of the p pairs. All training patterns are repeatedly presented to the network and the weights adjusted until the system responds correctly to most of the training patterns (success) or a specified number of presentations has been reached (failure). One full presentation of the training set, with each pattern being presented once, is called a training epoch.

2.2 Network Operation Rules

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

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

where w_{ji} is the weight of the connection from unit i to unit j in the layer above and o_i is the ouput of unit i. The output for unit j is calculated as a function of the net input; the sigmoid function is used in this study:

$$o_j = \frac{1}{1 + e^{-net_j}} \tag{2}$$

The goal of BP is to reduce the sum-squared error of the network, given by:

$$E = \sum_{i=1}^{p} \sum_{j=1}^{k} (t_j^i - o_j^i)^2,$$

where p is the number of patterns in the training set, k is the number of output units, t_j^i is the target response of output unit j to pattern i, and o_j^i is the actual response of output unit j to pattern i. Each weight w_{ji} from a lower unit i to a unit j in the layer above is changed by Δw_{ji}:

$$\Delta w_{ji} = \eta o_i \delta_j \tag{3}$$

where η is a the learning rate or step size constant, o_i is the output of the "source" unit i and δ_j is an error term for unit j:

$$\delta_j = \begin{cases} o_j' \, (t_j - o_j) & \text{for output units } j \\ o_j' \displaystyle\sum_k w_{kj} \delta_k & \text{for hidden units } j \end{cases} \tag{4}$$

o_j' is the derivative of the output function evaluated at the current net_j value.

See [Rumelhart et al., 1986] for the derivation of the above formulas.

2.3 Training Algorithm

The BP learning algorithm proceeds as follows. Each component of the input vector is mapped to the activation of the corresponding input unit. All units, j, in the first hidden layer then calculate their o_j values by equations (1) and (2). Subsequent layers are similarly activated, until an output pattern is produced on the output layer. This part of the algorithm is referred to as the forward phase. To start the backward phase, the δ_j and all Δw_{ji} are calculated for all output units j by equations (3) and (4). Once the errors and weight changes have been calculated for the output layer, the same process is repeated for the next layer down. continuing until the errors and weight changes are calculated for the first hidden layer. Note that the actual weight values are not yet updated, only the changes have been calculated. The Δw_{ji} values may be accumulated for an entire training epoch before updating the weights, or the weights may be updated after each pattern presentation.

3 MODIFICATIONS TO BACK-PROPAGATION

This section explains the step size and flat spot problems and presents the three modifications to BP 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.

A simple way to help alleviate the step size problem is to use a momentum term [Rumelhart et al., 1986]. The momentum term adds a fraction of the previous weight change to the current weight adjustment value. The modifed rule for adjusting the weight from unit i to unit j at time t is:

$$\Delta w_{ji}(t) = \eta o_i \delta_j + \alpha \Delta w_{ji}(t-1) \tag{5}$$

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

3.2 The Flat Spot Problem

In standard BP training, output units may get stuck in the wrong state for some training patterns. 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 for an output unit that is quite far from its target value will be small. Thus the weight changes will be small 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 add on offset to the sigmoid-prime function to ensure that it does not approach zero [Fahlman, 1988]. The resulting weight update rule is the same as equation (3), but we use $o_k' = o_k(1-o_k)+os$ in equation (4) to calculate the error δ_k for any unit k, where os = 0.0 or 0.1. We will call this the sigmoid-prime offset (SO) rule. An alternative solution to the flat spots problem has been explored in [Balakrishnan & Honavar, 1992].

3.3 The Expected Source Values Modification

Equation (3) decreases the sum-squared error after each update, driving the error toward zero. Let us look at the calculation of δ_j. For an output unit j, δ_j is the difference between j's target response and the actual response, multiplied by the derivative of o_j. So δ_j is the scaled error of unit j. For hidden units j, δ_j is the weighted sum of unit j's contribution to the error terms for units in the layer above scaled by the derivative of o_j. We can thus think of δ_j as the error term for hidden units as well. Note that input units have no error.

As noted earlier, the net effect of equation (3) is to decrease the δ_j term for each non-input unit in the network (for small η). The decrease in δ_j is effective only as long as each source unit's output, o_i, remains constant for future presentations of the current input pattern, because the derivation of the delta rule assumes infinitesimal weight changes. For hidden units i, changes in o_i due to the modification of weights w_{ih} from units in the layer below cannot be ignored. In practice we use the largest learning rate we can get away with to speed up convergence, so the weight changes made according to equation (3) may turn out to be incorrect.

Samad's proposal to alleviate this problem is to use the future value of o_i in equation (3). We can use the error term, δ_i, to approximate the future response of unit i. Since the incoming weights for unit i are modified as a function of the error term δ_i, and o_i is a function of the incoming weights, we can approximate the "expected value" of o_i as the sum of the current value and the error term δ_i. This leads us to the following modified BP rule [Samad, 1991]:

$$\Delta w_{ji} = \eta(o_i + \beta\delta_i)\delta_j \qquad (6)$$

where β is a constant. The new rule reverts back to the original rule when β equals zero or when i is an input unit ($\delta_i = 0.0$).

In actual implementations, there are two variation of this rule, based on when the error terms are calculated. In the original BP rule, each unit j only needed its own δ_j value and the output o_i of the source unit in order to update weight w_{ji}. In the ESV rule, the unit j also needs the error term δ_i for the source unit, but this value is not calculated until the next step of the backward phase. One solution to this dilemma is to calculate the error term for each unit in the network immediately after the forward phase. We will call this the up-front version of the ESV rule.

An alternative method would be to calculate error terms on an as-needed basis. In this method, the error terms for the output layer and the top hidden layer are calculated, then the weights between these two layers are updated. Next, the error terms for the top hidden layer are recalculated and the error terms for the second highest hidden layer (if there is one) are calculated and the weights between these two layers adjusted. This process continues until the weights between the first hidden layer and the input layer have been adjusted. This method requires calculating the error term for all hidden units twice: once to modify the outgoing weights, and once for the incoming weights since the error has now changed based on the outgoing weight changes. We will call this the on-demand version of the ESV rule.

3.4 Combined Rule

Combining the three modifications to the BP rule gives:

$$\Delta w_{ji}(t) = \eta(o_i + \beta\delta_i)\delta_j + \alpha\Delta w_{ji}(t-1) \qquad (7)$$

When calculating δ_k for any unit k, we use $o'_k = o_k(1-o_k)+os$, where $os = 0.0$ or 0.1.

4 DESCRIPTION OF EXPERIMENTS

We judge one rule against another based on convergence speed, generalization performance, and convergence percentage, each defined as follows.

Convergence Speed: the average number of epochs needed for the rule to successfully classify a given percentage of the training patterns. A lower average number of epochs means a faster convergence speed.

Generalization Performance: the average percentage of the test pattern set that the network responds to correctly. Two-thirds of the data set is used as a training set and the other third as a test set. The network is trained using only the training set.

Convergence Percentage: the percentage of the trials which successfully classified a given percentage of the training patterns.

A rule is said to have superior performance over another if it results in a higher convergence percentage, and a lower average number of training epochs. For the iris and soybean data, the rule must also have the same or higher generalization performance as the other.

The purpose of this study is three-fold:

1. Determine if one of the two variations of the ESV rule is superior to the other.
2. Determine how the modifications interact as they are used in various combinations.
3. Compare the results for small benchmark problems with those on real classification tasks.

In the remainder of this section, we will present the data sets which were used to evaluate the various modifications, then we will explain the learning trials we have run to address the above three concerns.

4.1 Data Sets

The data sets chosen are listed below along with the network architecture used and a description of the problem. We chose these data sets because they are popular benchmarks in connectionist research [Rumelhart et al., 1986, Fahlman, 1988, Yang & Honavar, 1991], thus giving many previous results to compare our results against. The three numbers following the data set name give the size of each layer of the network, starting at the input layer, e.g. 12–6–4 indicates 12 input units, 6 hidden units, and 4 output units.

8-bit and 10-bit Encoders: 8–3–8, 10–5–10; One of the input bits is turned on, and the system must learn to turn on the corresponding output bit.

8-bit and 10-bit Complement Encoders: 8–3–8, 10–5–10; All but one of the input bits is turned on, and the network must learn to produce the same pattern on the output units.

Iris Classification: 22–4–3, This data set is a real classification problem. It consists of 150 examples of iris plant samples which need to be classified into one of three species. This is the same data set as used in [Yang & Honavar, 1991].

Soybean Classification: 208–23–17, This data set is also a real classification problem. It consists of 289 examples of soybean plants with one of 17 diseases. This is the same data set as used in [Yang & Honavar, 1991].

4.2 Training Criteria

In order to address the goals of this study, we need to test the learning rule on all six data sets for a range of param-

Data Set	Rule	η	β	Ave. Epochs	% Converged	Correct Test %
8-Bit Encoder	UF	4.0	9.0	207.7 ± 54.6	100	100 ± 0.00
	OD	5.0	5.0	193.1 ± 73.8	100	100 ± 0.00
10-Bit Encoder	UF	4.0	4.0	82.2 ± 19.2	100	100 ± 0.00
	OD	4.0	4.0	114.9 ± 24.4	100	100 ± 0.00
8-Bit Compl. Encoder	UF	3.0	6.0	289.9 ± 97.2	90	100 ± 0.00
	OD	2.0	3.0	451.3 ± 192.0	80	100 ± 0.00
10-Bit Compl. Encoder	UF	4.0	1.0	99.9 ± 25.5	100	100 ± 0.00
	OD	1.6	3.0	303.1 ± 117.4	95	100 ± 0.00
Iris	UF	2.0	1.0	227.5 ± 260.9	75	92.3 ± 3.5
	OD	1.5	0.5	318.9 ± 303.2	65	91.1 ± 4.9
Soybean	UF	1.0	2.0	14.3 ± 6.0	100	92.0 ± 3.4
	OD	1.0	2.0	18.0 ± 10.4	100	91.5 ± 3.0

Table 1: Up-front versus On-demand error calculation

eter values. Note that a modification is deactivated when the corresponding parameter is set to zero.

Preliminary rough sampling of the parameter space for each problem indicated that the best ranges for the parameters are somewhat problem-dependent, so we have used different ranges for different data sets. In general, smaller values of η, α, and β gave better performance, while larger values tended to give oscillations and non-convergence.

Each training episode consisted of training a network on a single data set using the same set of initial weights while varying η, α, and β through a range of values. For each tuple of η, α, and β, the network was trained until it correctly classified a given percentage of the training patterns (success) or a specified number of training epochs were performed (failure). We used two different metrics of correct classification:

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; we used a threshold of 0.25.

Maximum output metric: [Yang & Honavar, 1991], the network is said to have correctly classified a training pattern if the output unit with maximum activation corresponds to the position of the maximum value in the target pattern. This criterion is a reasonable choice for classification tasks which have only a single active unit in the output pattern.

4.3 Trials Run

Twenty training episodes were run for each of the six data sets, using different initial weights for each episode. We used the threshold and margin classification metric for the enocoder and complement encoder problems and the maximum output unit metric for the iris and soybean data sets. The convergence criteria and parameter values for each data set were as follows:

8-bit encoder and complement encoders: training was performed until the system correctly classified 100% of the training set, or 1000 epochs were reached. η: 0.5, 1.0, 2.0, 3.0 4.0 5.0; α: 0.0, 0.4, 0.8; β: 0.0, 0.3, 0.6, 9.0.

10-bit encoder and complement encoders: training was performed until the system correctly classified 100% of the training set, or 2000 epochs were reached. η: 0.4, 0.8, 1.2, 1.6, 2.0, 3.0, 4.0; α: 0.0, 0.3, 0.6, 0.9; β: 0.0, 0.5, 1.0, 1.5, 2.0, 3.0, 4.0.

Iris data set: training was performed until the system correctly classified 99.5% of the training set, or 1000 epochs were reached. η: 0.5, 1.0, 1.5, 2.0; α: 0.0, 0.3, 0.6; β: 0.0, 0.5, 1.0, 1.5, 2.0.

Soybean data set: training was performed until the system correctly classified 99.5% of the training set, or 500 epochs were reached. η: 0.4, 0.7, 1.0; α: 0.0, 0.3, 0.6; β: 0.0, 0.5, 1.0, 1.5, 2.0.

5 EXPERIMENTAL RESULTS

In this section, we present our experimental data to address the three goals of this study. First, we show the results comparing the up-front and the on-demand versions of the ESV modification. Next, we show the various interactions observed between the three modifications. Finally, we compare the behavior of the modifications between the small benchmark data sets versus the real data sets.

5.1 Up-Front vs. On-Demand Error Calculation

Here we compare the up-front and the on-demand versions of the ESV rule. Table 1 shows the relevant information from the trials performed. For each data set, the best trial for both up-front (UF) and on-demand (OD) versions of the

ESV modification are shown. Neither the SO rule nor the momentum term were used. The average and standard deviation training epochs over convergent trials, the percent of the twenty trials which converged, and the average and standard deviation test set classification are given. For the small benchmarks, the entire data set was used for both the training and the test set.

It is apparent from table 1 that up-front error calculation is somewhat better than on-demand. In only one case, the 8-bit encoder, was the average epochs for the on-demand version lower than up-front, and this was not by a significant margin. The convergence percentages for the up-front version are always at least as high as for on-demand. Finally, on the iris and soybean data sets, the up-front rule had a slightly better classification percentage.

We believe the better performance of the up-front version of the ESV rule versus the on-demand version can be explained as follows. Consider updating the weights for hidden unit i. In the up-front rule, the error term δ_i is calculated once; this value is used to update both the incoming weights and the outgoing weights for unit i. It is used both as a predictor of the unit's future response and to actually change the unit's future response via the incoming weight updates. For the on-demand rule, a different value is used to predict the change in the unit's response than is used to actually update the incoming weights. Thus the predictive utility of the δ_i value used for updating the outgoing weights is reduced.

We will now narrow our consideration of the ESV modification to the up-front version. When we mention the ESV rule in the ensuing sections, it can be assumed we are referring to the up-front version.

5.2 Modification Interactions

In this section we present the results pertaining to the various combinations of the three modifications. We will focus our attention on the percent speedup; the convergence percentage and test pattern classification percentage were essentially equivalent across modifications.

There are eight possible combinations of the three modifications. For each data set we determined the best performance for each of the eight rules across the parameter ranges examined and calculated the percent speedup as compared to the original BP rule. The second column of table 2 shows the speedup values averaged over all six data sets.

In the table, BP + x denotes the back-propagation rule with the modification x, where "m" stands for the momentum modification, "e" for the ESV modification, and "o" the SO modification. The percent speedup is calculated using the BP rule as the standard of measure; it shows the percent fewer training epochs needed by the given rule compared to original BP.

The SO modification was the single most useful modification overall; it resulted in at least twice the speedup as the other two modifications. The most effective combination of two modifications was the ESV rule coupled with the SO rule. Overall, any combination of two modifications results in better speedup than any single modification. Of all eight rules, however, the one using all three modifications gave the most speedup; it appears that the modifications result in a better approximation to the optimal weight adjustment value.

5.2.1 Discussion

It is instructive at this point to look at the ESV modification at a deeper level than the intuitive argument presented above in order to account for its moderate usefulness.

The premise of the rule is that in the original weight update equation for BP, it would be useful to use the future value of the output of the hidden unit for weights between hidden units and the layer above. Let us denote the current pattern presentation time by t and the next presentation of the *same* pattern by $t+1$. Let i denote a hidden unit and j a unit in the layer above. Using this time notation, the ESV rule is an attempt to approximate

$$\Delta w_{ji}(t) = \eta o_i(t+1)\delta_j(t). \qquad (8)$$

Since $o_i(t+1)$ is completely determined by the current output of unit i, $o_i(t)$, and the current weight changes,Δw_{ih}, we can derive an exact value for $o_i(t+1)$ for the hidden units of a network with one hidden layer as follows:

$$o_i(t{+}1) = f(\sum_{h=1}^{n_I} w_{ih}(t{+}1)o_h) = f(\sum_{h=1}^{n_I} w_{ih}(t)o_h + \sum_{h=1}^{n_I} \Delta w_{ih}(t)o_h)$$

$$= f(o_i(t) + \delta_i(t)(\eta \sum_{h=1}^{n_I}(o_h)^2) = f(o_i(t) + \delta_i(t)(\eta \|I\|^2))$$

where f is the sigmoid function, n_I is the number of input units, and $\|I\|$ is the norm of the input vector I.

The actual ESV rule uses $(o_i(t)+\beta\delta_i(t))$ to approximate $o_i(t+1)$. This is a crude approximation, and for the large values of β appropriate to some data sets, it is a very crude estimate. The rule is useful either because an approximation to $o_i(t+1)$ is better than an exact value, or the rule is doing something different than the intuitive argument suggests. The ESV rule could be useful simply because it dynamically changes the step size. Further theoretical and empirical work is needed to explain the speedup observed.

5.3 Small Benchmarks vs. Real Classification Tasks

In this section, we will compare the performance of the various combinations across the two classes of data sets. Once again, we will focus our attention on the percent speedup, since the convergence percentage and test pattern classification percentage did not vary significantly across modifications.

The results in table 2 show that for both the small benchmarks and the real classification data sets, the SO rule is the

Rule	Overall	Small Benchmarks	Real Classification Tasks
BP	0.0%	0.0%	0.0%
BP + m	22.4%	28.9%	9.4%
BP + e	14.2%	11.9%	18.7%
BP + o	42.6%	30.9%	65.9%
BP + m,e	31.8%	33.2%	29.2%
BP + m,o	46.7%	37.8%	64.5%
BP + e,o	57.8%	53.9%	65.7%
BP + e,m,o	62.5%	60.3%	66.8%

Table 2: Percent speedup averaged over the two classes of data sets

single most useful modification, but by a much greater margin for the real classification data sets. The momentum term does not significantly improve performance for the real classification data sets, but improves performance by almost one third for the small benchmarks. The ESV modification was somewhat more effective for the real classification data sets than for the small benchmarks, giving a modest improvement.

For the small benchmarks, the most notable combination of two modifications is the one using both the SO and the ESV rules. The two modifications complement each other: the resulting speedup is more than the sum of the speedup of the individual modifications.

For the real classification data sets, the SO modification dominates the other two. Any combination using the SO rule has a speedup at least twice that of any combination not using it. All combinations using the SO rule yield essentially equivalent speedup values.

The combination of all three modifications gave the best speedup for the small benchmarks and the real classification data sets. For the real problems, however, most of the speedup is due to the SO modification.

The effectiveness of the various combinations of modifications varied appreciably across the two classes of data sets. The variation in performance underscores the need to evaluate a learning rule across many data sets, including some real classification tasks.

6 CONCLUSIONS AND FUTURE DIRECTIONS

We have found that the up-front version of the ESV rule is consistently better than the on-demand version in terms of number of training epochs, convergence percentage, and test pattern classification. Further experiments with the ESV rule only need focus on the up-front version.

The three modifications generally work well together, most combinations for all data sets resulted in better speedup than the individual modifications. The most notable combination is the SO rule coupled with the ESV rule. For any application of the BP rule, we recommend the use of all three modifications. The computational overhead is small, and the gain in learning speed outweighs this small overhead.

We need to further understand the ESV modification. It is not clear from the intuitive argument given in [Samad, 1991] why we get the improvements that were observed. Further empirical and theoretical investigation is needed to determine whether the rule gives better performance simply due to roughly and dynamically adjusting the step size, or if it is approximating a more useful weight update value.

References

[Balakrishnan & Honavar, 1992] Balakrishnan, K., & V. Honavar. 1992. "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.

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

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

[Samad, 1991] Samad, T. 1991. "Back Propagation With Expected Source Values." *Neural Networks,* Vol. 4: 615-618.

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