CB3: An Adaptive Error Function for Backpropagation Training

MICHAEL RIMER and TONY MARTINEZ
Computer Science Department, Brigham Young University, 3361 TMCB, PO Box 26576, Provo, UT 84602-6576, USA. e-mails: mrimer@axon.cs.byu.edu, martinez@cs.byu.edu

Abstract. Effective backpropagation training of multi-layer perceptrons depends on the incorporation of an appropriate error or objective function. Classification-based (CB) error functions are heuristic approaches that attempt to guide the network directly to correct pattern classification rather than using common error minimization heuristics, such as sum-squared error and cross-entropy, which do not explicitly minimize classification error. This work presents CB3, a novel CB approach that learns the error function to be used while training. This is accomplished by learning pattern confidence margins during training, which are used to dynamically set output target values for each training pattern. On eleven applications, CB3 significantly outperforms previous CB error functions, and also reduces average test error over conventional error metrics using 0-1 targets without weight decay by 1.8%, and by 1.3% over metrics with weight decay. CB3 also exhibits lower model variance and tighter mean confidence interval.

Key words. neural network, backpropagation, classification, error functions, adaptive targets

Abbreviations. CB – classification-based; CE – cross-entropy; SSE – sum-squared error

1. Introduction

Multi-layer feed-forward neural networks trained through error backpropagation [11] have received substantial attention as robust learning models for classification tasks. Classification-based (CB) error functions [9, 10] are a relatively new method of training multi-layer perceptrons. CB functions heuristically seek to directly minimize classification error by backpropagating network error only on misclassified patterns. In doing so, they perform relatively minimal updates to network parameters in order to discourage premature weight saturation and overfitting. This is conducive to higher accuracy in classification problems than optimizing with respect to commonly used error functions, such as sum-squared error (SSE) and cross-entropy (CE). This work presents a novel CB error function, CB3, which improves on existing CB functions. It is an adapting error function that dynamically sets output target values during training by learning confidence margins on each pattern in the training set. These confidence margins guide the network in learning each pattern according to the ability of the network, selectively learning patterns that appear to provide better generalization while avoiding those that would encourage weight saturation and possible overfit without improving generalization.

Performance of networks trained with the CB3 error function is compared against previous CB error functions, SSE and CE with and without weight decay [7], on a corpus of eleven benchmark machine learning datasets. CB3 shows a significant reduction in average test

error of 1.8% over standard backpropagation using conventional 0-1 target values without weight decay, and a significant decrease of 1.3% average test error over backpropagation augmented by weight decay regularization. CB3 also exhibits lower model variance with smaller standard deviation.

Section 2 reviews related work and motivation for this new approach. Sections 3 and 4 present the CB3 algorithm and a working example. Section 5 describes experiments performed and Section 6 gives empirical results and discussion.

2. Motivation for CB3

A prime goal of classification learning is to achieve high recognition rates on unseen data. To generalize well, a learner must use a proper objective function. The validity of using common differentiable metrics like sum-squared-error (SSE) relies on the assumption that sample outputs are offset by inherent Gaussian noise, being normally distributed about a cluster mean. For function approximation of an arbitrary signal, this presumption often holds. However, this assumption is invalid for classification tasks, where assigned real-valued target vectors are arbitrary values used to represent class labels. This suggests that other error metrics (e.g. cross-entropy) are more suited to classification problems. Likewise, cross-entropy (CE) is preferable to SSE when output class distributions are not balanced. When this is not the case, CE and SSE may perform equivalently.

Traditionally, classification problems are learned through error backpropagation by providing a vector of strict ("hard") 0/1 target values to represent the class label of a particular pattern. Minimizing an error function with hard target values tends to a saturation of weights, often equated with overfitting. There is evidence that the magnitude of the weights in a network plays a more important role in generalization than the number of hidden nodes [1]. It follows that overfit might be reduced by keeping the weights smaller. Regularization methods such as weight decay [7,14] are commonly used to discourage weight saturation and overfit. These methods generally assume that overfitting is a global phenomenon. However, overfit can vary significantly in different regions of the model. Proper early stopping methods that take advantage of this information can further improve generalization [6].

An alternate method of discouraging weight saturation is to provide a maximum error tolerance threshold, d_{max}, and not backpropagate any error when network output values are within this range of the target values. That is, for a given d_{max}, target value, t_k, and network output, o_k, no weight update occurs if the absolute error $| t_k - o_k | < d_{max}$. This threshold is arbitrarily chosen to indicate the point at which a sample has been sufficiently approximated. Using an error threshold, a network is permitted to converge with smaller weights [12].

Rankprop [5] provides an alternative method to training with hard target values and empirically shows that it improves generalization. Rankprop records the output of the learner for each training pattern. It then sorts the samples in the training set based on class, then according to output values. Thus, a rank of the samples consistent with the current model is

developed and used to define the target values on the next epoch. The idea behind Rankprop is that in the case of complex nonlinear solutions a simpler, *less nonlinear* function is provided to learn instead. The resulting simpler model often generalizes better.

Prior work has shown [8, 9, 10] that methods of calculating softer values for each training pattern based on the network's output vector improves generalization and reduces variance on classification problems over a corpus of benchmark learning problems. One of these, called lazy training or CB1, focuses on classification accuracy backpropagates an error signal through the network only when a pattern is misclassified. CB2, starts with the "lazy" targets used in CB1 and gradually separates them until they reach the 0-1 targets used in standard training. Other approaches involve using an "oracle" teacher network to provide target output values to simpler networks that can learn to emulate its behavior.

This work extends CB1 and CB2 by providing a heuristic to learn how much error can be tolerated in each training pattern based on how well the network is learning in order to improve generalization.

3. CB3 Algorithm

Learning how much error to backpropagate based on the performance of the network being trained is, in effect, a meta-learning algorithm. In other words, the error function itself is learned based on the ability of the network to learn it. CB3 accomplishes this by learning how confident the network is in classifying each training pattern as learning progresses. This method for dynamically learning pattern confidence margins is shown in Figure 1. CB3 modifies the standard back-propagation algorithm in the following three ways:

- For each pattern-output node pair, a confidence value is stored and modified over time. This value represents an interval in the range of the squashing function that reflects the numeric amount by which the node is assisting in classifying the pattern correctly or incorrectly.
- As training progresses, each pattern's learned confidence values are used in calculating the target output values for the pattern.
- The objective function is modified based on these target values to decide how large of an error signal to backpropagate through the network for each pattern.

Without loss of generality, in this work it is assumed that a single, distinct output node in the network represents each class label. Let N be the number of output nodes (and distinct class labels). On a given pattern, let o_j be the output value of the j^{th} output node of the network ($0 < o < 1$, $1 \le j \le N$). Let T designate the target output class for that pattern and c_j signify the class label of the j^{th} output node. For the output node corresponding to the pattern's class label, $c_j = T$. We refer to this output node as c_T for short. For non-target output nodes, $c_j \neq T$. Non-target output nodes are called *competitors*. Let o_T denote the value of the target output node. Let $o_{\sim T\max}$ denote the value of the highest-outputting competitor.

Initialization.
Set expected confidence values, $C_{i,j}$, for each pattern-output node *ij* pair. Set d_{max}.

Training.
Present a training pattern, *i*, to the network.
Determine o_T and $o_{\sim T\max}$.

1. For each output node *j*, set its target output, T_j:

$$T_j \equiv \begin{cases} \min(1-d_{\max}, o_{\sim T\max} + \alpha C_{i,j}) & \text{if } c_j = T \text{ and } C_{i,j} > 0 \\ \max(d_{\max}, o_T - \alpha C_{i,j}) & \text{if } c_j \neq T \text{ and } C_{i,j} > 0 \\ o_{\sim T\max} & \text{if } c_j = T \text{ and } C_{i,j} \leq 0 \\ o_T & \text{if } c_j \neq T \text{ and } C_{i,j} \leq 0 \end{cases} \quad (1)$$

2. From each output node *j*, backpropagate error, ε_j.

$$\varepsilon_j \equiv \begin{cases} T_j - o_j & \text{if } c_j = T \text{ and } o_j < T_j \\ T_j - o_j & \text{if } c_j \neq T \text{ and } o_j > T_j \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

3. For all *j*, update confidence value $C_{i,j}$ to make it closer to the observed interval.

$$C_{i,j} = C_{i,j} + \eta_C (OI_j - C_{i,j}) \quad (3)$$

where the observed interval, OI_j, is defined as

$$OI_j \equiv \begin{cases} o_j - o_{\sim T\max} & \text{if } c_j = T \\ o_T - o_j & \text{if } c_j \neq T \end{cases} \quad (4)$$

Continue training until stopping criterion is satisfied.

Figure 1. CB3 algorithm.

In the initialization phase, for each combination of training pattern and network output node, a confidence value, *C*, is stored. With *I* training patterns and *J* output nodes in the network, this results in *IJ* values being stored. These values indicate the amount of confidence the network's output nodes expect to have in classifying the corresponding pattern correctly. Positive values indicate that a pattern is expected to be correctly classified by the output node while negative values mean it is expected to be misclassified by the node. These values are updated as training progresses. We have found initial confidence values above approximately 0.2 to generalize better than values below 0.2 on all applications we tested (see Figure 2).

During training, patterns are shuffled each epoch and stochastically presented to the network. The vector of target values for a pattern's class outputs is determined as shown in Figure 1, equation 1. Each target value is calculated differently depending on whether $c_j = T$ and

whether the confidence value for that node, $C_{i,j}$, is positive or negative. With a positive confidence, the target value for c_T is set to $o_{\sim T\max} + \alpha C_{i,j}$. That is, it is set to the maximum competitor's output value plus the confidence value on node j for pattern i, multiplied by α, a multiplicative factor greater than or equal to one. This factor intuitively refers to how aggressively CB3 will try to separate the target values for opposing classes. A value of one will allow targets to remain closer together while a greater value will separate them more. Conversely, competitor class targets are set to $o_T - \alpha C_{i,j}$, i.e., c_T's output value minus the confidence value, multiplied by α. We have found values for α above 1.5 to produce better generalization than lower α values on all tested applications (see Figure 3).

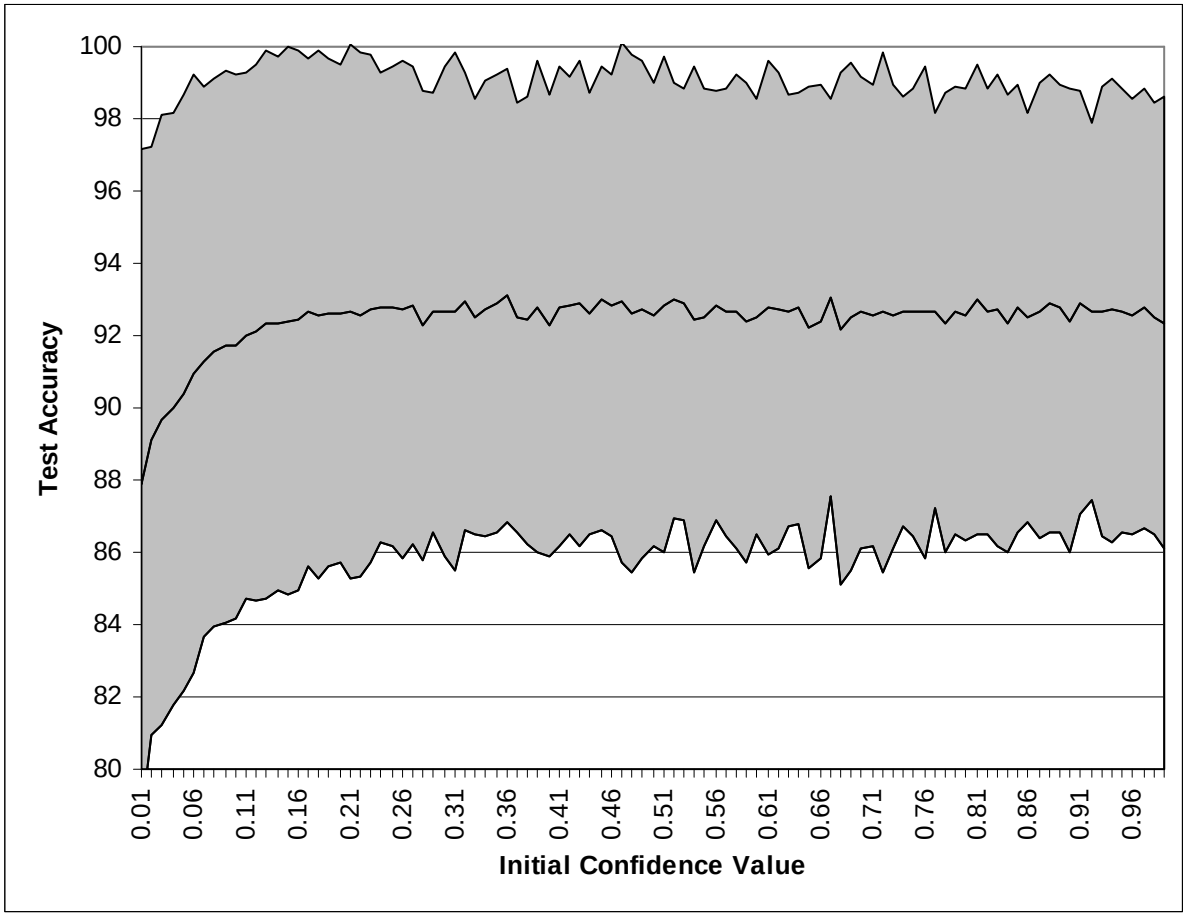

Figure 2. Influence of CB3's initial confidence values on the test accuracy of selected applications. The highlighted area is the 95% confidence interval for an observation.

The $\min(1-d_{max},\cdot)$ and $\max(d_{max},\cdot)$ operators are used to keep the target values within the range of possible output values (*e.g.* [0,1] for the sigmoid activation function) while also applying the error tolerance threshold d_{max}, which can be set to a small positive value to discourage weights from entering the saturation range, as discussed above.

When a negative confidence value exists, $o_{\sim T\max}$ and o_T are used as the target values. The reason for this is that a negative confidence indicates that this output node has learned to consistently misclassify this pattern over time. This could happen if the network either does

not have enough hidden nodes to learn the problem sufficiently, or a noisy or incorrectly labeled pattern is encountered. In either case, further effort to learn these "problem" patterns could lead to premature weight saturation or overfit. On the other hand, it is possible that the network's hidden nodes have simply not yet learned to model this area of the problem space correctly, in which case some effort should still be made to learn to classify this pattern correctly. If this is not the case, however, undue resources (in the form of network parameters) have not been squandered in trying to learn a pattern that will probably not improve generalization and could even be detrimental to it.

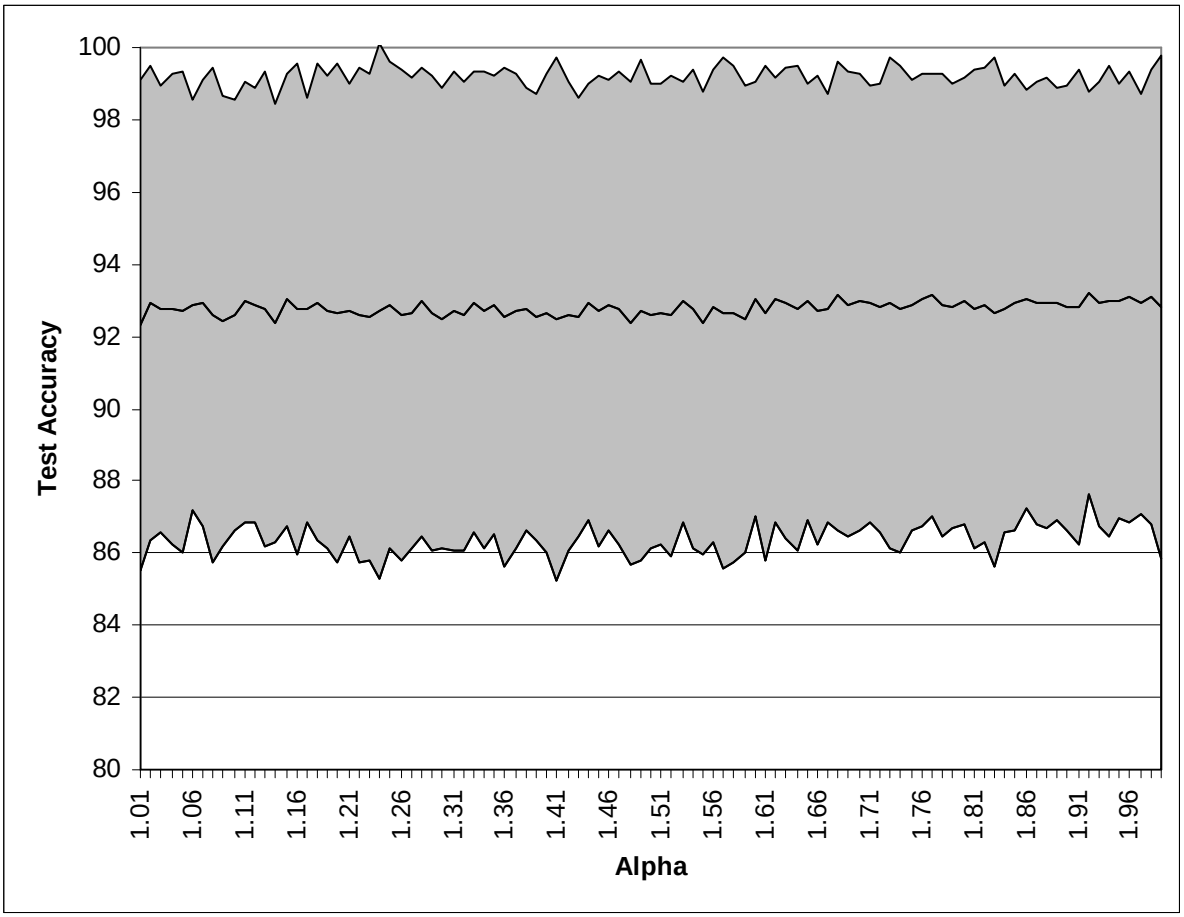

Figure 3. Influence of CB3's α parameter on the test accuracy of selected applications. The highlighted area is the 95% confidence interval for an observation.

Once the target vector has been calculated, the error signal for backpropagation through the network is determined (see Figure 1, equation 2). Observe that no error is backpropagated by CB3 when an output value is already better than the dynamically set target value (i.e., a value higher than the target value for the target class node, or a value lower then the target for nodes of other classes). That is, backpropagating error depends on whether the pattern is currently being classified correctly or not, as in CB1 [10]. This is akin to setting the d_{max} error threshold for each pattern based on the value of the highest outputting competitor. This selective error signal avoids updating network weights when doing so would not necessarily lead to improved accuracy (the pattern is already being classified correctly) while risking

premature weight saturation and overfit. On misclassified patterns, error is backpropagated only from output nodes that fall short of their dynamic target values and are considered in some way responsible for the misclassification.

Following the determination and backpropagation of error terms comes the meta-learning step, where the learned confidences used in calculating the dynamic target values are also iteratively updated (see Figure 1, equation 3). The current pattern's observed interval vector, OI_i, is calculated as shown in Figure 1, equation 4, indicating the amount of split between the output values of positive and negative class labels on this pattern, respective to each output node.

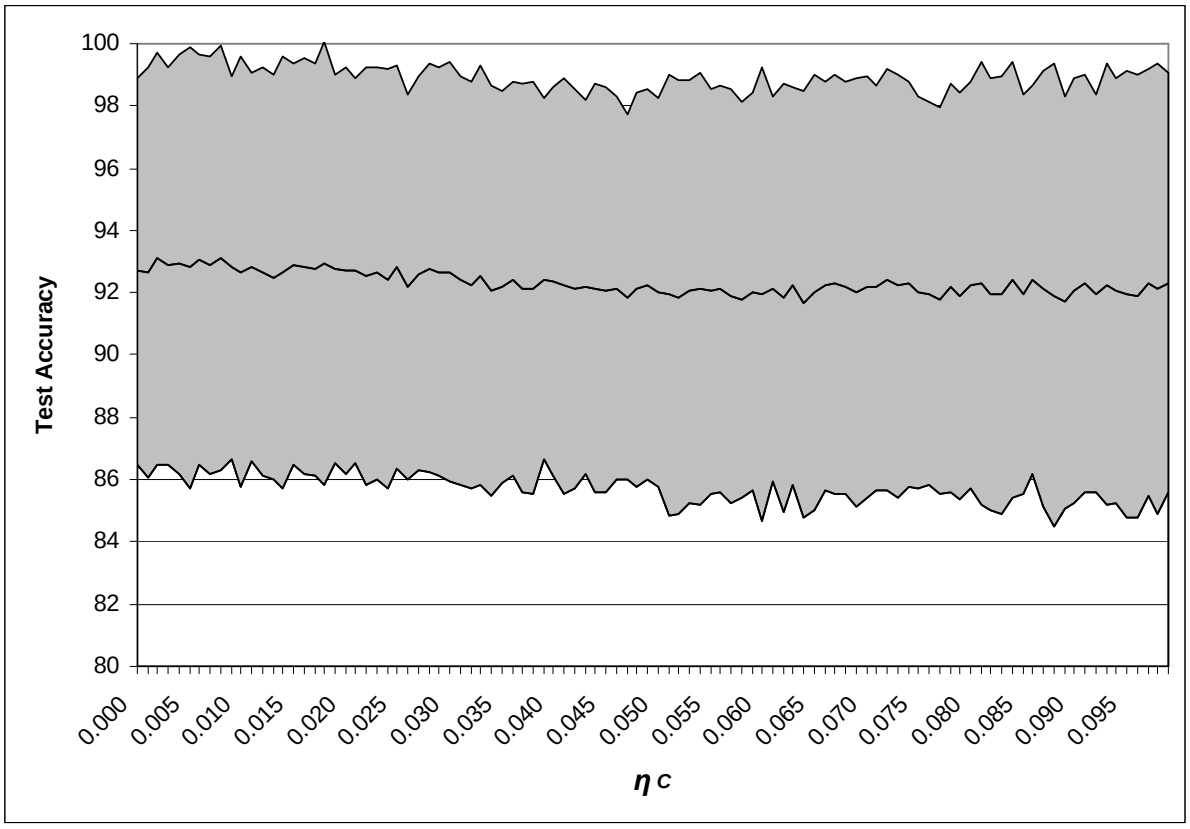

Figure 4. Influence of η_C on the test accuracy of selected applications. The highlighted area is the 95% confidence interval for an observation.

As an error delta is calculated for iterative weight updating to bring output values closer to the target values, likewise the learned confidence value $C_{i,j}$ is subtracted from the actual observed interval OI_j and multiplied by learning rate η_C to calculate a confidence value delta. This confidence delta is added to the learned confidence. We observed values of η_C less than 0.025 to have the highest generalization and lowest variance on all tested applications (see Figure 4).

As training continues, $C_{i,j}$ will be iteratively learned. It reflects the ability of the network output nodes to locally distinguish the target class from the others. Learning this value has the practical purpose of hinting to the network which patterns can be classified with

confidence (such as cluster centers), which patterns need to be learned more (fringe patterns), and which patterns appear difficult to learn correctly with the current network at all (noise or overlapping classes). This enables the training process to guide the network to spend more resources on learning training patterns that most probably contribute to higher generalization accuracy while selectively ignoring those that lead the network to overfit and weight saturation as it attempts to learn them.

CB3 requires an O(n) scan through the n network outputs to determine the highest target and competitor values, to set the target value for each node before backpropagation, and then to update the observed confidence interval following backpropagation. However, this additional overhead to the standard backpropagation is negligible compared to the computation requirements of O(ih) for feed-forwarding a pattern vector and O(ihn) for backpropagation, where i is the number of inputs and h is the number of hidden nodes. In fact, CB3 saves time by omitting the error backpropagation step for correctly classified patterns with sufficient output confidences. The number of epochs required to converge is similar for CB3 and CE training, and CB3 generally converges in about half as many epochs as SSE training.

The following section presents a hypothetical example of using the CB3 algorithm.

4. CB3: An Example

Given a three-class problem with class labels A, B, and C, a pattern i is labeled as type B. Let $d_{max} = 0.1$, $\alpha = 1.5$ and $\eta_C = 0.01$.

Let the current confidence values for this pattern be [0.1, 0.3, 0.1], indicating the network output nodes all believe they are performing correctly but with only a moderate margin separating them. If the network were to output [0.4, 0.7, 0.3] on this pattern, o_T is 0.7 and $o_{\sim T\max}$ is 0.4. The target vector T would be calculated to be

$$\begin{aligned} T &= [o_T - mC_{i,j} \, , \, o_{\sim T\max} + \alpha C_{i,j} \, , \, o_T - \alpha C_{i,j}] \\ &= [0.7 - 1.5(0.1) \, , \, 0.4 + 1.5(0.3) \, , \, 0.7 - 1.5(0.1)] \\ &= [0.55, 0.85, 0.55]. \end{aligned}$$

All values are within the imposed ranges, so the min/max operators are omitted here for clarity. Next, the error vector ε is calculated. The first and third outputs are satisfactorily outside the learned confidence margins but the second node has a target value exceeding its output. Thus ε is [0,0.15,0]. Error is backpropagated from the second output node only. Its observed confidence may increase over subsequent iterations.

The pattern's confidence values are now each updated by the delta η_C ($OI_j - C_{i,j}$), where the observed interval vector OI is

$$OI = [o_T - o_j \, , \, o_j - o_{\sim T\max} \, , \, o_T - o_j]$$

= [0.7 – 0.4, 0.7 – 0.4, 0.7 – 0.3]
= [0.3, 0.3, 0.4]

and the update delta is

= [0.01(0.3 – 0.1), 0.01(0.3 – 0.2), 0.01(0.4 – 0.1)]
= [0.002, 0.001, 0.003].

The learned confidences are increased to [0.102, 0.201, 0.103]. As training progresses, if the network continues to output similar values, the confidences on this pattern will continue to grow. As these confidences get large, greater error is backpropagated and the observed interval over network outputs will tend to reflect these confidences.

5. Experiments

Several well-known benchmark classification problems were selected from the UC Irvine Machine Learning Repository (UCI MLR) [2]. The problems were selected so as to have a wide variety of characteristics (number of patterns, number and type of features, and complexity) in order to analyze the robustness of the learning algorithm.

Experiments were performed using a privately developed C++ library comparing feed-forward multi-layer perceptron networks optimizing SSE and cross-entropy (CE), both with standard 0-1 targets with and without weight decay regularization, to the CB1-3 algorithms. It was observed that SSE and CE optimizing networks yielded nearly identical results using both static 0-1 targets and with the CB1-3 algorithms. Therefore, only the results of training with CE are presented for brevity.

The multi-layer perceptrons had a single, fully connected hidden layer and were trained through on-line backpropagation. The optimal number of hidden nodes was empirically determined for each task based on holdout set accuracy, searching layer sizes within the range from one to fifty hidden nodes. In all experiments, network weights were initialized to uniform random values within the range [-0.1,0.1] [13]. The learning rate was 0.1, momentum was 0.7, and the error tolerance, d_{max}, was 0.1 (*i.e.*, target values of 0.1 and 0.9 were used). Weight decay values between $\lambda = 0.00001$ to 0.0001 were used, optimized for each application [7]. For CB3, $\alpha = 1.5$, $\eta_C = 0.01$, and the initial pattern confidence values were 0.25.

Feature values (both nominal and continuous) were normalized between zero and one. Training patterns were presented to the network in a random order each epoch. The same initial random seed for network weight initialization and sample shuffling was used for all experiments on a given dataset.

Pattern classification was determined by *winner-take-all* (the class of the highest outputting node is chosen). Training continued until the training set was successfully learned or training

set classification error ceased to decrease for a substantial number of epochs. The resultant number of epochs trained was comparable among all approaches within a factor of three. The model selected for test evaluation was the network on the epoch with the best holdout set accuracy, where the holdout set consisted of 20% of the original training data.

6. Results and Discussion

Table 1 lists the results of testing a multi-layer perceptron backpropagating error maximizing cross-entropy without weight decay (BP), with weight decay (BPw), and CB1-3 on the selected applications. Each field lists first the average test set accuracy using 30-fold stratified cross validation. Neural network experiments were averaged over 30 runs with random initial weights. The first value in each cell is the average accuracy over these runs. The second value is the 95% Student's *t* confidence interval for these means. The best generalization for each problem is underlined.

Table 1. Results on UCI MLR datasets using stratified cross-validation.

Data set	ann	bal-ance	bcw	derm	ecoli	ionos phere	iris	musk 2	pima	sonar	wine	**avg**
BP	98.1	95.0	97.1	97.2	85.6	91.6	94.9	<u>99.4</u>	72.1	80.6	98.5	91.8
	0.18	1.80	1.25	1.66	3.89	2.90	3.72	0.13	3.19	3.21	1.80	0.37
BPw	95.2	96.6	96.7	97.2	86.1	92.8	96.7	97.0	75.6	83.2	98.1	92.3
	0.21	1.59	0.74	1.72	3.82	1.65	1.54	0.25	1.63	2.88	1.09	0.36
CB1	97.4	<u>97.4</u>	97.2	96.1	84.2	90.6	96.7	99.2	76.3	84.1	97.8	92.5
	0.30	0.89	1.10	2.00	4.30	2.82	2.97	0.15	2.63	3.00	2.99	0.37
CB2	98.2	97.1	96.9	<u>97.8</u>	86.0	92.0	96.0	99.3	75.5	83.7	98.3	92.8
	0.19	1.25	1.14	1.34	4.30	2.31	3.48	0.10	2.88	2.27	2.88	0.37
CB3	<u>98.3</u>	97.2	<u>97.5</u>	<u>97.8</u>	<u>86.6</u>	<u>92.9</u>	<u>97.3</u>	98.9	<u>78.1</u>	<u>86.1</u>	<u>98.6</u>	<u>93.6</u>
	0.15	1.00	0.81	1.39	3.70	2.43	2.51	0.21	2.78	2.32	1.71	<u>0.35</u>

CB3 has higher test accuracy and tighter mean confidence interval then CE without weight decay (BP) on ten of the eleven datasets tested. BP outperformed CB3 on the *musk2* dataset. CB3 reduces average test error by 1.8% over BP, significant with a pairwise Student's *t* confidence of $p < 0.05$. CB3 has a tighter confidence interval, which indicates it has a smaller standard deviation and is more robust to perturbations in the initial network parameter values and pattern presentation order.

CB3 has higher accuracy than BPw on all eleven datasets and exhibits an average decrease in test error of 1.3%, significant with a pairwise *t* confidence of $p < 0.05$. CB3 has a tighter confidence interval than BPw on six of the eleven datasets and is slightly higher than BPw on average. CB3 outperforms CB1 and CB2 on nine of eleven datasets, with an average decrease in test error of 1.1% and 0.8%, respectively, significant with a pairwise *t* confidence of $p < 0.05$. CB3 has a tighter confidence interval than CB1 and CB2.

For a given function *f(x)*, there may exist a function *g(x)* that also solves the given problem but is easier for backpropagation to learn [3, 4, 5]. Recall that CB3 does not modify the actual backpropagation algorithm used for updating the network parameters in any way. CB3's power comes through the modification of target values as a tool for smoother training. Most often, conventional target values of 0 and 1 are used in classification tasks to learn proper class labels. CB3, using target values greater than 0 and less than 1, consequently calculates much smaller error terms during the initial phase of training. Less error may result in a function that is easier for backpropagation to learn.

7. Conclusions and Future Work

CB3 is shown to be superior to multi-layer backpropagation networks trained with previous CB error functions and CE with hard targets without weight decay (BP) and with weight decay (BPw) over a corpus of eleven applications. CB3 significantly reduces average test error by 1.8% over BP, by 1.3% over BPw, and by 1.1% and 0.8% over CB1 and CB2, respectively. It is surmised that CB3, learning to approximate iteratively learned target values, provides a function that is easier for backpropagation to learn than the strict conventional 0-1 classification function.

Since the learned confidence values are able to implicitly represent noisy patterns, they could be used to explicitly mark overlapping class regions. This knowledge is useful to reduce local uncertainty for problems with regions of the feature space that are inherently multi-class. A method for such an approach and analysis of its efficacy is forthcoming. An ROC analysis of CB3 is planned on applications where the cost of false positives is different than false negatives.

Furthermore, preliminary tests have shown that pattern misclassifications are not highly correlated between CB3 and BP. We will experiment with combining BP- and CB-trained networks in hybrid ensembles and voting committees to further improve generalization.

References

1. Bartlett, P. L.: The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network, *IEEE Transaction Information Theory* 44(2) (1998), pp. 525-536.
2. Blake, C. and Merz, C.: UCI machine learning repository, http://www.ics.uci.edu/~mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science (1998).
3. Caruana, R.: Algorithms and applications for multitask learning, In: *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 87-95, Bari, Italy (1996).
4. Caruana, R.: Multitask Learning. Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University. Pittsburg, PA (1997).

5. Caruana, R., Baluja, S., & Mitchell, T.: Using the future to 'sort out' the present: rankprop and multitask learning for medical risk evaluation, In: *Advances in Neural Information Processing Systems 8*, pp. 959-965, MA: MIT Press, Cambridge (1996).
6. Caruana, R., Lawrence, S. & Giles, C.: Overfitting in neural networks: backpropagation, conjugate gradient, and early stopping, *Neural Information Processing Systems*, Denver, Colorado (2000).
7. Krogh, A. & Hertz, J.: A simple weight decay can improve generalization, *Advances in Neural Information Processing Systems 4*, pp. 950-957, San Mateo, CA (1995).
8. Menke, J., Peterson, A., Rimer, M. & Martinez, T.: Neural network simplification through oracle learning, In: *Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'02*, pp. 2482-2497 (2002).
9. Rimer, M. and Martinez, T.: Softprop: softmax neural network backpropagation learning, In: *Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'04*, pp. 979-984 (2004).
10. Rimer, M. and Martinez T.: Classification-based objective functions, *Machine Learning* 63(2) (2006), pp. 183-205.
11. Rumelhart, D. E., Hinton, G. and Williams, R.: *Learning Internal Representations by Error Propagation*, Institute for Cognitive Science, University of California, San Diego; La Jolla, CA (1985).
12. Schiffmann, W., Joost, M. and Werner, R.: *Optimization of the Backpropagation Algorithm for Training Multilayer Perceptions*, University of Koblenz: Institute of Physics (1994).
13. Thimm, G. and Fiesler, E.: High-order and multilayer perceptron initialization, *IEEE Transactions on Neural Networks* 8(2) (1997), 249-259.
14. Werbos, P.: Backpropagation: past and future, In: *Proceedings of the IEEE International Conference on Neural Networks*, IEEE Press, New York, pp. 343-353 (1988).