Adaptive Boosting of Neural Networks for Character Recognition

Holger Schwenk Yoshua Bengio

Dept. Informatique et Recherche Opérationnelle
Université de Montréal, Montreal, Qc H3C-3J7, Canada
{schwenk,bengioy}@iro.umontreal.ca

May 29, 1997

Technical Report #1072,
Département d'Informatique et Recherche Opérationnelle,
Université de Montréal

Abstract

"Boosting" is a general method for improving the performance of any learning algorithm that consistently generates classifiers which need to perform only slightly better than random guessing. A recently proposed and very promising boosting algorithm is *AdaBoost* [5]. It has been applied with great success to several benchmark machine learning problems using rather simple learning algorithms [4], in particular decision trees [1, 2, 6]. In this paper we use AdaBoost to improve the performances of neural networks applied to character recognition tasks. We compare training methods based on sampling the training set and weighting the cost function. Our system achieves about 1.4% error on a data base of online handwritten digits from more than 200 writers. Adaptive boosting of a multi-layer network achieved 2% error on the UCI Letters offline characters data set.

keywords: AdaBoost, learning algorithms, artificial neural networks, bounds on generalization error, margin distributions, autoencoders, handwritten character recognition.

1 Introduction

AdaBoost [4, 5] (for *Ada*ptive *Boost*ing) constructs a composite classifier by sequentially training classifiers, while putting more and more emphasis on certain patterns. AdaBoost has been applied to rather weak learning algorithms (with low capacity) [4] and to decision trees [1, 2, 6], and not yet, until now, to the best of our knowledge, to artificial neural networks. These experiments displayed rather intriguing generalization properties, such as continued decrease in generalization error after training error reaches zero. Previous workers also disagree on the reasons for the impressive generalization performance displayed by AdaBoost on a large array of tasks. One issue raised by Breiman [1] and the authors of AdaBoost [4] is whether some of this effect is due to a reduction in variance similar to the one obtained from the Bagging algorithm.

In this paper we explore the application of AdaBoost to Diabolo (auto-associative) networks and multi-layer neural networks (MLPs) on two character recognition tasks. In doing so, we also compare three different versions of AdaBoost: (R) training each hypothesis with a fixed training set obtained by resampling with replacement (with probabilities D_t) from the original training set (as in [1]), (E) training by resampling after each epoch a new training set from the original training set, and (W) training by directly weighting the cost function (here the squared error) of the neural network. Note that the second version is a better approximation of the weighted cost function. If the variance reduction induced by averaging the hypotheses from very different models explains some of the generalization performance of AdaBoost, then the weighted training version (W) should perform worse then the resampling versions, and the fixed sample version (R) should perform better then the continuously resampled version (E).

2 AdaBoost

AdaBoost combines the hypotheses generated by a set of classifiers trained one after the other. The t^{th} classifier is trained with more emphasis on certain patterns, using a cost function weighted by a probability distribution D_t over the training data ($D_t(i)$ is positive and $\sum_i D_t(i) = 1$). Some learning algorithms don't permit training with respect to a weighted cost function, e.g. decision trees. In this case resampling with replacement (using the probability distribution D_t) can be used to approximate a weighted cost function. Examples with high probability would then occur more often than those with low probability, while some examples may not occur in the sample at all although their probability is not zero. This is particularly true in the simple resampling version (labeled "R" earlier), and probably unlikely when a new training set is resampled after each epoch ("E" version). Neural networks can be trained directly with respect to a distribution over the learning data by weighting the cost function (this is the "W" version): the squared error on the i-th pattern is weighted by the probability $D_t(i)$. The result of training the t^{th} classifier is a *hypothesis* $h_t : X \rightarrow Y$ where $Y = \{1, ..., k\}$ is the space of labels, and X is the space of input features. After the t^{th} round the weighted error ϵ_t of the resulting classifier is calculated and the distribution D_{t+1} is computed from D_t, by increasing the probability of incorrectly labeled examples. The global decision f is obtained by weighted voting. Figure 1 (left) summarizes the basic AdaBoost algorithm. It converges (learns the training set) if each classifier yields a weighted error that is less than 50%, i.e., better than chance in the 2-class case.

There is also a multi-class version, called *pseudoloss*-AdaBoost, that can be used when the classifier computes confidence scores for each class [4, 5]. The result of training the t^{th} classifier is now a hypothesis $h_t : X \times Y \rightarrow [0, 1]$. Furthermore we use a distribution over the set of all *miss-labels*: $B = \{(i, y) : i \in \{1, ..., N\}, y \neq y_i\}$. Therefore $|B| = N(k-1)$. AdaBoost modifies this distribution so that the next learner focuses not only on the examples that are hard to classify, but more specifically on the incorrect labels that are hardest to discriminate. Note that the miss-label distribution D_t induces a distribution over the examples: $P_t(i) = W_i^t / \sum_i W_i^t$ where $W_i^t = \sum_{y \neq y_i} D_t(i, y)$. $P_t(i)$ may be used for resampling using the above described version "R" or "E". Freund and Schapire define the *pseudoloss* of a learning machine as [5]:

$$\epsilon_t = \frac{1}{2} \sum_{(i,y) \in B} D_t(i, y)(1 - h_t(x_i, y_i) + h_t(x_i, y)) \tag{1}$$

It is minimized if the confidence scores in the correct labels are 1.0 and the confidence scores of all the wrong labels are 0.0. The final decision f is obtained by adding together the weighted confidence scores of all machines. Figure 1 right summarizes the algorithm. For more details we refer the reader to the references [4, 5]. This multi-class boosting algorithm converges if each classifier yields a pseudoloss that is less than 50%, i.e., better than any constant hypothesis.

AdaBoost has very interesting theoretical properties, in particular it can be shown that the error of the composite classifier on the training data decreases exponentially fast to zero [5]. More importantly, however, bounds on the *generalization error* of such a system have been formulated [7]. These are based on a notion of *margin* of classification, defined as the difference between the score of the correct class and the strongest score of a wrong class. In the case in which there are just two possible labels $\{-1, +1\}$, this is $yf(x)$, where f is the composite classifier and y the correct label. Obviously, the classification is correct if the margin is positive. We now state the theorem bounding the generalization error of Adaboost [7] (and other classifiers obtained by

Input: sequence of N examples $(x_1, y_1), \ldots, (x_N, y_N)$ with labels $y_i \in Y = \{1, \ldots, k\}$	
Init: $D_1(i) = 1/N$ for all i	**Init:** let $B = \{(i, y) : i \in \{1, ..., N\}, y \neq y_i\}$ $D_1(i, y) = 1/\lvert B\rvert$ for all $(i, y) \in B$
Repeat:	**Repeat:**
1. Train neural network with respect to distribution D_t and obtain hypothesis $h_t : X \to Y$	1. Train neural network with respect to distribution D_t and obtain hypothesis $h_t : X \times Y \to [0, 1]$
2. calculate the weighted error of h_t: $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$ abort loop if $\epsilon_t > \frac{1}{2}$	2. calculate the pseudo-loss of h_t: $\epsilon_t = \frac{1}{2} \sum_{(i,y) \in B} D_t(i, y)(1 - h_t(x_i, y_i) + h_t(x_i, y))$
3. set $\beta_t = \epsilon_t/(1 - \epsilon_t)$	3. set $\beta_t = \epsilon_t/(1 - \epsilon_t)$
4. update distribution D_t $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \beta_t^{\delta_i}$ with $\delta_i = (h_t(x_i) = y_i)$ and Z_t a normalization constant	4. update distribution D_t $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \beta_t^{\frac{1}{2}((1+h_t(x_i,y_i)-h_t(x_i,y))}$ where Z_t is a normalization constant
Output: final hypothesis: $f(x) = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t}$	**Output:** final hypothesis: $f(x) = \arg\max_{y \in Y} \sum_t \left(\log \frac{1}{\beta_t}\right) h_t(x, y)$

Figure 1: AdaBoost algorithm (left), multi-class extension using confidence scores (right)

convex combinations of other classifiers). Let H be a set of hypotheses (from which the h_t are chosen), with VC-dimension d. Let f be any convex combination of hypotheses from H. Let S be a sample of N examples chosen independently at random according to a distribution D. Then with probability at least $1 - \delta$ over the random choice of the training set S from D, the following bound is satisfied for all $\theta > 0$:

$$P_D[yf(x) \leq 0] \ \leq \ P_S[yf(x) \leq \theta] + O\left(\frac{1}{\sqrt{N}}\sqrt{\frac{d\log^2(N/d)}{\theta^2} + \log(1/\delta)}\right) \tag{2}$$

Note that this bound is independent of the number of combined hypotheses and how they are chosen from H. The distribution of the margins however plays an important role. It can be shown that the AdaBoost algorithm is especially well suited to the task of maximizing the number of training examples with large margin [7].

3 The Diabolo Classifier

Normally, neural networks used for classification are trained to map an input vector to an output vector that encodes directly the classes, usually by the so called "1-out-of-N encoding". An alternative approach with interesting properties is to use auto-associative neural networks, also called autoencoders or *Diabolo networks*, to learn a model of each class. In the simplest case, each autoencoder network is trained only with examples of the corresponding class, i.e., it learns to reconstruct all examples of one class at its output. The distance between the input vector and the reconstructed output vector expresses the likelihood that a particular example is part of the corresponding class. Therefore classification is done by choosing the best fitting model, i.e. the class of the network with the smallest reconstruction error. Figure 2 summarizes the basic architecture. It shows also typical classification behavior. The input and output vectors are (x, y)-coordinate sequences of a character. The visual representation in the figure is obtained by connecting these points. In this example the "1" is correctly classified since the network for this class has the smallest reconstruction error. It is also interesting to see how the other networks find a stroke sequence of their class that is relatively close to the net input (2 and 7 in this example).

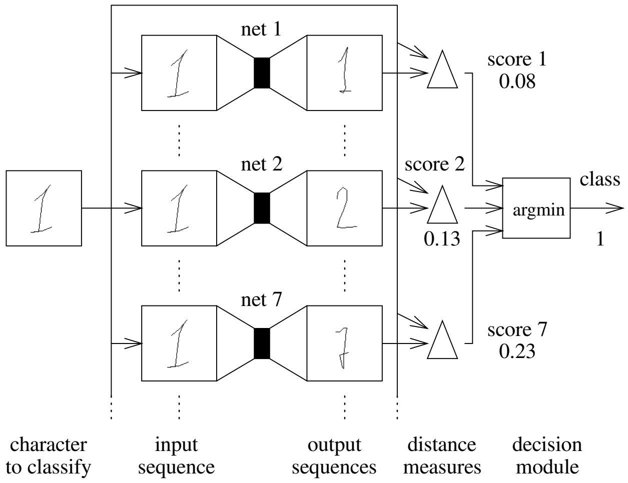

Figure 2: Architecture of a Diabolo classifier

The Diabolo classifier uses a *distributed representation* of the models which is much more compact than the enumeration of references often used by distance-based classifiers like nearest-neighbor or RBF networks. Furthermore, one has to calculate only one distance measure for each class to recognize. This allows to incorporate knowledge by a domain specific distance measure at a very low computational cost. In previous work [8], we have shown that the well-known tangent-distance [11] can be used in the objective function of the autoencoders. Furthermore, we used a discriminant learning algorithm [9]: the network weights are updated so that the reconstruction distance is minimized for the network of the desired class and maximized for the closest incorrect one, like in LVQ2. Interestingly, this intuitive algorithm has gained a theoretical justification by the recent theorem of equation 2, since this discriminant learning algorithm decreases the number of examples with low margin. This Diabolo classifier has achieved state-of-the-art results in handwritten OCR [8, 9].

Recently, we have also extended the idea of a transformation invariant distance measure to on-line character recognition [10]. One autoencoder alone, however, can not learn efficiently the model of a character if it is written in many different stroke orders and directions. The architecture can be extended by using several autoencoders per class, each one specializing on a particular writing style (subclass). For the class "0", for instance, we would have one Diabolo network that learns a model for zeros written clockwise and another one for zeros written counterclockwise. The assignment of the training examples to the different subclass models should ideally be done in an unsupervised way. However, this can be quite difficult since the number of writing styles is not known in advance and usually the number of examples in each subclass varies a lot. Our training data base contains for instance 100 zeros written counterclockwise, but only 3 written clockwise (there are also some more examples written in other strange styles). Classical clustering algorithms would probably tend to ignore subclasses with very few examples since they aren't responsible for much of the error, but this may result in poor generalization behavior. Therefore, in previous work we have manually assigned the subclass labels [10]. Of course, this is not a generally satisfactory approach, and certainly infeasible when the training set is large. In the following, we will show that the emphasizing algorithm of AdaBoost can be used to train multiple Diabolo classifiers per class, performing a soft assignment of examples of the training set to each network.

4 Results with Diabolo and MLP Classifiers

Experiments have been performed on two data sets for character recognition. The first one is a data base of online handwritten digits (10 classes) and the second one is the UCI *Letters* database of offline machine-printed alphabetical characters (26 classes). Both data sets have a pre-defined training and test set. The Diabolo classifier was only applied to the online data set (since it takes advantage of the structure of the input features).

The online data set was collected at Paris 6 University [10]. It is writer-independent (different writers in training and test sets) and there are 203 writers, 1200 training examples and 830 test examples. Each writer gave only one example per class. Therefore, there are many different writing styles, with very different frequencies. In particular, the writers of the training and test sets are completely distinct. Figure 3 gives an idea of the great variety of writing styles of this database.

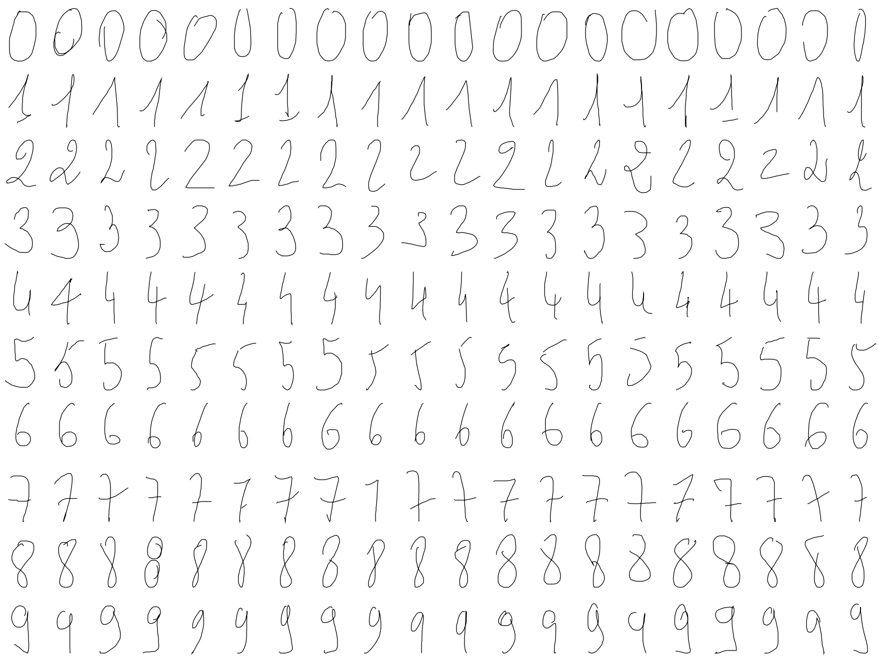

Figure 3: Some examples of the on-line handwritten digits database (test set)

We only applied a simple preprocessing: the characters were resampled to 11 points, centered and size normalized to a (x,y)-coordinate sequence in $[-1,1]^{22}$. Since the Diabolo classifier is invariant to small transformations we don't need to extract further features. We also made some minor changes to the resampling procedure used in the AdaBoost algorithm applied to the online data set, to take advantage of the geometrical structure of the input variables. When resampling patterns from the training set, we applied random local affine transformations to the original characters (for which we know that the classification is invariant),

Table 1: Online digits data set error rates for different unboosted classifiers

	Diabolo classifier		fully connected MLP		
	no subclasses	hand-selected	22-10-10	22-30-10	22-80-10
train:	2.2%	0.6%	6.5%	1.3%	0.1%
test:	3.3%	1.2%	9.5%	4.1%	2.1%

Table 1 summarizes the results on the test set of different approaches before using AdaBoost. The Diabolo classifier with hand-selected sub-classes in the training set performs best since it is invariant to transformations and since it can deal with the different writing styles. The experiments suggest that fully connected neural networks are not well suited for this task: small nets do poorly on both training and test sets, while large nets overfit.

Table 2: Online digits test error rates for boosted MLPs

architecture:	22-10-10			22-30-10		
version:	R	E	W	R	E	W
1000 iter., best	not tried			1.9%	1.9%	3.0%
final				2.1%	2.1%	4.3%
5000 iter, best	2.4%	3.0%	2.4%	1.9%	1.9%	2.0%
final	2.6%	3.2%	2.5%	2.4%	2.8%	2.9%

Table 2 shows the results of boosted multi-layer perceptrons with 10 or 30 hidden units, trained for either 1000 or 5000 iterations, and using either the ordinary resampling scheme (R), resampling with different random selections at each epoch (E), or training with weights D_t on the squared error criterion for each pattern (W). The final performance is with at most 100 machines. The multi-class version of the AdaBoost algorithm was used in all the experiments with MLPs: it yielded considerably better results than the basic version. Pseudoloss-AdaBoost was however not useful for the Diabolo classifier since it uses a powerful discriminant learning algorithm. Note that AdaBoost with weighted training of MLPs doesn't work if the learning of each individual MLP is stopped too early (1000 iterations): the networks didn't learn well enough the weighted examples and ϵ_t approached rapidly 0.5. When training each MLP for 5000 iterations, however, the weighted training (W) version achieved the same low test error, and we need about half (or less) machines than with the fixed resampling version (R). Finally, AdaBoost was not very useful if the VC-dimension of the basic classifier is very large: the test error decreased only from 2.1% to 1.8% for the 22-80-10 MLP. The results of the unboosted MLP was however obtained when training was stopped at the lowest test error and it's unlikely to achieve this using early stopping on a validation set. On the other hand, we have never observed that early stopping was necessary when using AdaBoost with MLPs or Diabolo classifiers.

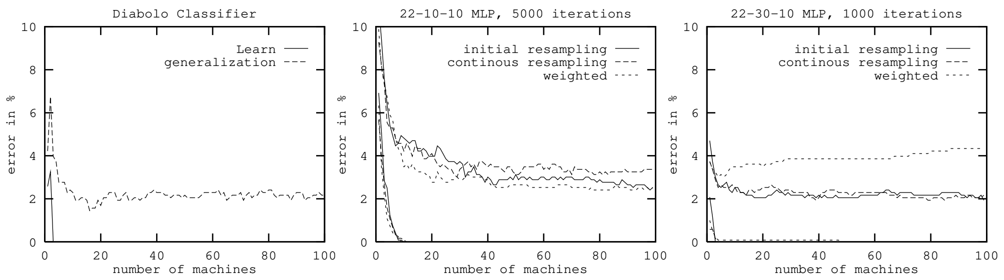

Figure 4: error rates of the boosted classifiers for increasing number of network

Figure 4 shows the error rate of some of the boosted classifiers as the number of machines is increased. AdaBoost brings training error to zero after only a few steps, even with a MLP with only 10 hidden units. The generalization error is also considerably improved and it continues to decrease asymptotically after zero training error has been reached. The Diabolo classifier performs best when combining 16 classifiers (**1.4%** error = 12 errors) which is almost as good as the Diabolo classifier using hand-selected subclasses (1.2% = 10 errors). Since we know that one autoencoder can't learn a model of the different writing style within one class, this seems to be evidence that the example emphasizing of AdaBoost was able to assign them automatically to different machines.

The surprising effect of continuously decreasing generalization error even after training error reaches zero has already been observed by others [1, 2, 4, 6]. This seems to contradict Occam's razor, but it may be explained by the recently proven theorem of Schapire et al. [7]: the bound on the generalization error (equation 2) depends only on the margin distribution and on the VC-dimension of the basic learning machine (one Diabolo classifier or MLP respectively), not on the number of machines combined by AdaBoost.

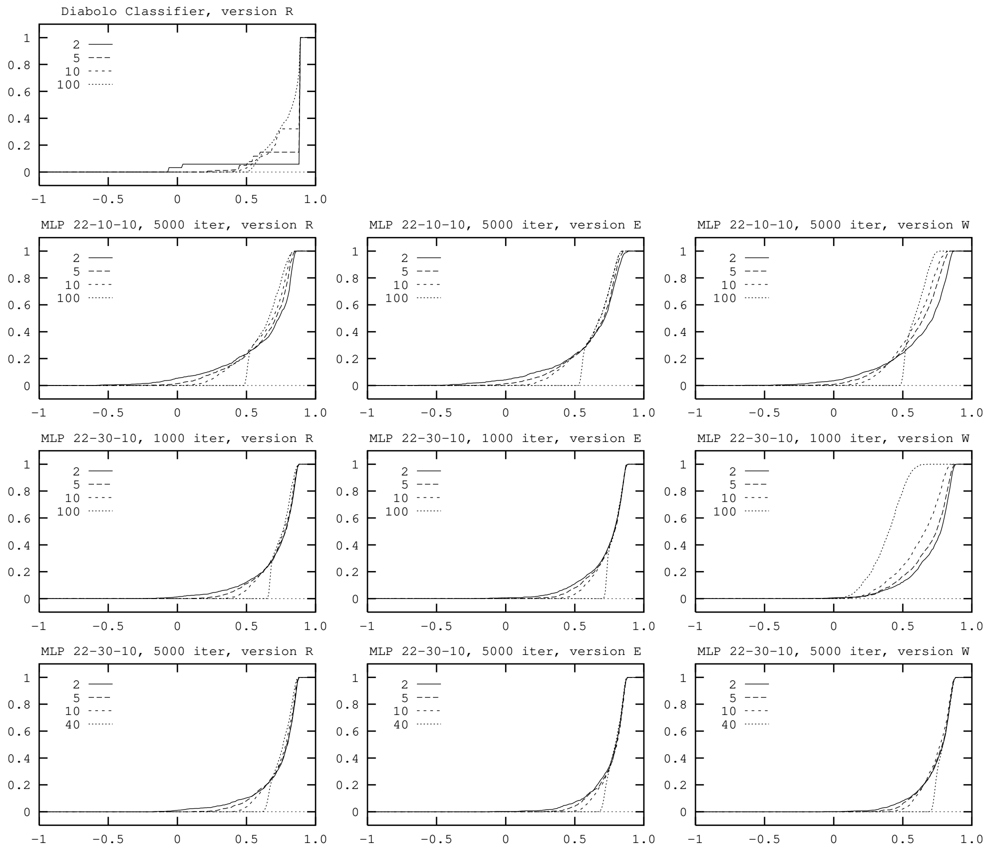

Figure 5: margin distributions for the different resampling versions using an increasing number of networks

Figure 5 shows the margins distributions, i.e. the fraction of examples whose margin is at most x as a function of $x \in [-1, 1]$. It is clearly visible that AdaBoost increases the number of examples with high margin: with 100 machines all examples have a margin higher than 0.5. This improvement is achieved with all versions of AdaBoost ("R", "E" or "W"). AdaBoost with weighted training of the MLP 22-10-10, however, seems to decrease the margins of examples that already had large margins before boosting. Remember that this version doesn't work if training is stopped too early (right graph in the third line).

Although AdaBoost improves spectacularly the generalization error of the MLPs, for instance from 9.5 % to 2.4 % for the 22-10-10 architecture, we were not able to get results as good as for the unboosted Diabolo classifier (1.4%). Note that the margin distribution using hundred 22-30-10 MLPs is at least as good as the one from 100 Diabolo classifiers, but not the error rate. We hypothesize that the difference in performance may be due in part to a lower effective VC-dimension of the Diabolo classifier. It may also be that the generalization error bounds of Freund et al. are too far away from the actual generalization error.

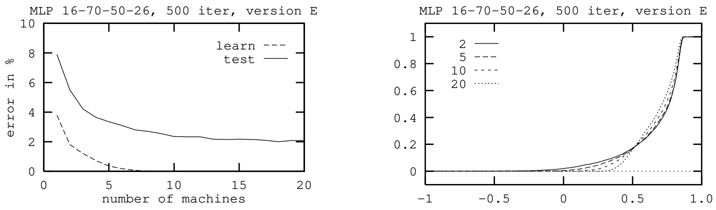

Figure 6: error rates and margin distributions for the UCI letters data base

Similar experiments were performed with MLPs on the "Letters" data set from the UCI Machine Learning database. It has 16000 training and 4000 test patterns, 16 input features, and 26 classes (A-Z) of machine-printed characters from 20 different fonts. The experiments were performed with a 16-70-50-26 MLP with 500 online back-propagation epochs on the training set. Each input feature was normalized according to its mean and variance on the training set. The plain and boosted networks are compared to the 1-nearest neighbor classifier as well as to plain and boosted C4.5 decision trees (results from [4]).

Table 3: Error rates on the UCI Letters data set

	nearest neighbor	fully connected MLP		C4.5 (results from [4])	
		alone	adaboosted†	alone	adaboosted‡
train:	-	3.4%	0.0%	unknown	0.0%
test:	4.8%	6.2%	2.0%	13.8%	3.3%

† using 20 machines
‡ using $>$ 100 machines

The results obtained with the boosted network are extremely good (2% error) and are the best that the authors know to be published for this data set. The best performance reported in STATLOG [3] is 6.4%. Adaboosted decision trees are reported to achieve 3.3% [4, 7], but it seems that many trees are necessary (more than 100) while we need only 20 MLPs (see table 3 and figure 6).

5 Conclusion

As demonstrated in two character recognition applications, AdaBoost can significantly improve neural classifiers such as multi-layer networks and Diabolo networks. The behavior of AdaBoost for neural networks confirms previous observations on other learning algorithms [1, 2, 4, 6, 7], such as the continued generalization improvement after zero training error has been reached, and the associated improvement in the margin distribution. In general, AdaBoost appears to "help" most the "weaker" models (such as the 10-hidden units networks in our experiments).

Another interesting finding of this paper is that the "weighted training" version of AdaBoost works well for MLPs but requires many more training epochs (because of the weights the conditioning of the Hessian matrix is probably worse). However, less machines are required to achieve the best test set error, which makes the total training time similar to the resampling version, and shorter recognition time. Furthermore these results add credence to the view of Freund and Schapire that the improvement in generalization error brought by AdaBoost is mainly due to the emphasizing (that increases the margin), rather than to a variance reduction due to the randomization of the resampling process (as was suggested in [1]).

References

[1] L. Breiman. Bias, variance, and Arcing classifiers. Technical Report 460, Statistics Departement, University of California at Berkeley, 1996.

[2] H. Drucker and C. Cortes, Boosting decision trees. In *NIPS*8*, pages 479–485, 1996.

[3] Feng. C., Sutherland, A., King, R., Muggleton, S., & Henery, R. Comparison of machine learning classifiers to statistics and neural networks. In *Proceedings of the Fourth International Workshop on Artificial Intelligence and Statistics*, pages 41–52, 1993.

[4] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of Thirteenth International Conference*, pages 148–156, 1996.[1]

[5] Y. Freund and R.E. Schapire. A decision theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Science*, to appear.[1]

[6] J.R. Quinlan. Bagging, Boosting and C4.5. In *14th National Conference on Artificial Intelligence*, 1996.

[7] R.E. Schapire, Y. Freund, P. Bartlett, and W.S. Leee. Boosting the margin: A new explanation for the effectiveness of voting methods. In *Machines That Learn - Snowbird*, 1997.[1]

[8] H. Schwenk and M. Milgram. Transformation invariant autoassociation with application to handwritten character recognition. *NIPS*7*, pages 991–998. MIT Press, 1995.

[9] H. Schwenk and M. Milgram. Learning discriminant tangent models for handwritten character recognition. In *ICANN*96*, pages 585–590. Springer Verlag, 1995.

[10] H. Schwenk and M. Milgram. Constraint tangent distance for online character recognition. In *International Conference on Pattern Recognition*, pages D 520–524, 1996.

[11] P. Simard, Y. Le Cun, and J. Denker. Efficient pattern recognition using a new transformation distance. *NIPS*5*, pages 50–58. Morgan Kaufmann, 1993.

[1] electronically available at htttp://www.research.att.com/˜yoav