Boosting Neural Networks

paper No 1806

Holger Schwenk
LIMSI-CNRS, bat 508, BP 133, 91403 Orsay cedex, FRANCE

Yoshua Bengio
DIRO, University of Montréal, Succ. Centre-Ville, CP 6128
Montréal, Qc, H3C 3J7, CANADA

To appear in Neural Computation

Abstract

"Boosting" is a general method for improving the performance of learning algorithms. A recently proposed boosting algorithm is *AdaBoost*. It has been applied with great success to several benchmark machine learning problems using mainly decision trees as base classifiers. In this paper we investigate whether AdaBoost also works as well with neural networks, and we discuss the advantages and drawbacks of different versions of the AdaBoost algorithm. In particular, we compare training methods based on sampling the training set and weighting the cost function. The results suggest that random resampling of the training data is not the main explanation of the success of the improvements brought by AdaBoost. This is in contrast to Bagging which directly aims at reducing variance and for which random resampling is essential to obtain the reduction in generalization error. Our system achieves about 1.4% error on a data set of online handwritten digits from more than 200 writers. A boosted multi-layer network achieved 1.5% error on the UCI Letters and 8.1% error on the UCI satellite data set, which is significantly better than boosted decision trees.

Keywords: AdaBoost, boosting, Bagging, ensemble learning,
multi-layer neural networks, generalization

1 Introduction

”Boosting” is a general method for improving the performance of a learning algorithm. It is a method for finding a highly accurate classifier on the training set, by combining “weak hypotheses” (Schapire, 1990), each of which needs only to be moderately accurate on the training set. See an earlier overview of different ways to combine neural networks in (Perrone, 1994). A recently proposed boosting algorithm is *AdaBoost* (Freund, 1995), which stands for “Adaptive Boosting”. During the last two years, many empirical studies have been published that use decision trees as base classifiers for AdaBoost (Breiman, 1996; Drucker and Cortes, 1996; Freund and Schapire, 1996a; Quinlan, 1996; Maclin and Opitz, 1997; Bauer and Kohavi, 1998; Dietterich, 1998b; Grove and Schuurmans, 1998). All these experiments have shown impressive improvements in the generalization behavior and suggest that AdaBoost tends to be robust to overfitting. In fact, in many experiments it has been observed that the generalization error continues to decrease towards an apparent asymptote after the training error has reached zero. (Schapire et al., 1997) suggest a possible explanation for this unusual behavior based on the definition of the *margin of classification*. Other attemps to understand boosting theoretically can be found in (Schapire et al., 1997; Breiman, 1997a; Breiman, 1998; Friedman et al., 1998; Schapire, 1999). AdaBoost has also been linked with game theory (Freund and Schapire, 1996b; Breiman, 1997b; Grove and Schuurmans, 1998; Freund and Schapire, 1998) in order to understand the behavior of AdaBoost and to propose alternative algorithms. (Mason and Baxter, 1999) propose a new variant of boosting based on the direct optimization of margins.

Additionally, there is recent evidence that AdaBoost may very well overfit if we combine several hundred thousand classifiers (Grove and Schuurmans, 1998). It also seems that the performance of AdaBoost degrades a lot in the presence of significant amounts of noise (Dietterich, 1998b; Rätsch et al., 1998). Although much useful work has been done, both theoretically and experimentally, there is still a lot that is not well understood about the impressive generalization behavior of AdaBoost. To the best of our knowledge, applications of AdaBoost have all been to decision trees, and no applications to multi-layer artificial neural networks have been reported in the literature. This paper extends and provides a deeper experimental analysis of our first experiments with the application of AdaBoost to neural networks (Schwenk and Bengio, 1997; Schwenk and Bengio, 1998).

In this paper we consider the following questions: does AdaBoost work as well for neural networks as for decision trees? short answer: yes, sometimes even better. Does it behave in a similar way (as was observed previously in the literature)? short answer: yes. Furthermore, are there particulars in the way neural networks are trained with gradient back-propagation which should be taken into account when choosing a particular version of AdaBoost? short answer: yes, because it is possible to directly weight the cost function of neural networks. Is overfitting of the individual neural networks a concern? short answer: not as much as when not using boosting. Is the random resampling used in previous implementations of AdaBoost critical or can we get similar performances by weighing the training criterion (which can easily be done with neural networks)? short answer: it is not critical for generalization but helps to obtain faster convergence of individual networks when coupled with stochastic gradient descent.

The paper is organized as follows. In the next section, we first describe the AdaBoost algorithm and we discuss several implementation issues when using neural networks as base classifiers. In section 3, we present results that we have obtained on three medium-sized tasks: a data set of handwritten on-line digits and the "letter" and "satimage" data set of the UCI repository. The paper finishes with a conclusion and perspectives for future research.

2 AdaBoost

It is well known that it is often possible to increase the accuracy of a classifier by averaging the decisions of an ensemble of classifiers (Perrone, 1993; Krogh and Vedelsby, 1995). In general, more improvement can be expected when the individual classifiers are diverse and yet accurate. One can try to obtain this result by taking a base learning algorithm and by invoking it several times on different training sets. Two popular techniques exist that differ in the way they construct these training sets: Bagging (Breiman, 1994) and boosting (Freund, 1995; Freund and Schapire, 1997). In Bagging, each classifier is trained on a bootstrap replicate of the original training set. Given a training set S of N examples, the new training set is created by resampling N examples uniformly with replacement. Note that some examples may occur several times while others may not occur in the sample at all. One can show that, on average, only about 2/3 of the examples occur in each bootstrap replicate. Note also that the individual training sets are independent and the classifiers could be trained in parallel. Bagging is known to be particularly effective when the classifiers are "unstable", i.e., when perturbing the learning set can cause significant changes in the classification behavior. classifiers. Formulated in the context of the bias/variance decomposition (Geman et al., 1992), Bagging improves generalization performance due to a reduction in variance while maintaining or only slightly increasing bias. Note, however, that there is no unique bias/variance decomposition for classification tasks (Kong and Dietterich, 1995; Breiman, 1996; Kohavi and Wolpert, 1996; Tibshirani, 1996).

AdaBoost, on the other hand, constructs a composite classifier by sequentially training classifiers while putting more and more emphasis on certain patterns. For this, AdaBoost maintains a probability distribution $D_t(i)$ over the original training set. In each round t the classifier is trained with respect to this distribution. Some learning algorithms don't allow training with respect to a weighted cost function. In this case, sampling 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.

```
Input:  sequence of $N$ examples $(x_1, y_1), \ldots, (x_N, y_N)$
        with labels $y_i \in Y = \{1, \ldots, k\}$

Init:   let $B = \{(i, y) : i \in \{1, ..., N\}, y \neq y_i\}$
        $D_1(i, y) = 1/|B|$ for all $(i, y) \in B$

Repeat:
  1. Train neural network with respect to distribution $D_t$ and obtain
     hypothesis $h_t : X \times Y \to [0, 1]$
  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 $\beta_t = \epsilon_t/(1 - \epsilon_t)$
  4. update distribution $D_t$
         $D_{t+1}(i, y) = \frac{D_t(i,y)}{Z_t} \beta_t^{\frac{1}{2}((1+h_t(x_i,y_i)-h_t(x_i,y))}$
     where $Z_t$ is a normalization constant

Output: final hypothesis:
         $f(x) = \arg\max_{y \in Y} \sum_t \left(\log \frac{1}{\beta_t}\right) h_t(x, y)$
```

Table 1: *Pseudo-loss AdaBoost (AdaBoost.M2).*

After each AdaBoost round, the probability of incorrectly labeled examples is increased and the probability of correctly labeled examples is decreased. The result of training the t^{th} classifier is a *hypothesis* $h_t : X \to Y$ 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 $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$ 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 probabilities are changed so that the error of the t^{th} classifier using these new "weights" D_{t+1} would be 0.5. In this way, the classifiers are optimally decoupled. The global decision f is obtained by weighted voting. This basic AdaBoost algorithm 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.

In general, neural network classifiers provide more information than just a class label. It can be shown that the network outputs approximate the a-posteriori probabilities of classes, and it might be useful to use this information rather than to perform a hard decision for one recognized class. This issue is addressed by another version of AdaBoost, called *Ada-Boost.M2* (Freund and Schapire, 1997). It can be used when the classifier computes confidence scores[1] for each class. The result of training the t^{th} classifier is now a hypothesis $h_t : X \times Y \to [0, 1]$. Furthermore, we use a distribution $D_t(i, y)$ over the set of all *miss-labels*:

[1]The scores do not need to sum to one.

$B = \{(i, y) : i \in \{1, ..., N\}, y \neq y_i\}$, where N is the number of training examples. 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 improving the discrimination between the correct class and the incorrect class that competes with it. 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 the training set. (Freund and Schapire, 1997) define the *pseudoloss* of a learning machine as:

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

It is minimized if the confidence scores of 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 the machines (all the hypotheses h_1, h_2, ...). Table 1 summarizes the AdaBoost.M2 algorithm. This multi-class boosting algorithm converges if each classifier yields a pseudo-loss 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 as the number of combined classifiers is increased (Freund and Schapire, 1997). Many empirical evaluations of AdaBoost also provide an analysis of the so-called *margin distribution*. The margin is defined as the difference between the ensemble score of the correct class and the strongest ensemble 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 output of the composite classifier and y the correct label. The classification is correct if the margin is positive. Discussions about the relevance of the margin distribution for the generalization behavior of ensemble techniques can be found in (Freund and Schapire, 1996b; Schapire et al., 1997; Breiman, 1997a; Breiman, 1997b; Grove and Schuurmans, 1998; Rätsch et al., 1998).

In this paper, an important focus is on whether the good generalization performance of AdaBoost is partially explained by the random resampling of the training sets generally used in its implementation. This issue will be addressed by comparing three versions of AdaBoost, as described in the next section, in which randomization is used (or not used) in three different ways.

2.1 Applying AdaBoost to neural networks

In this paper we investigate different techniques of using neural networks as base classifiers for AdaBoost. In all cases, we have trained the neural networks by minimizing a quadratic criterion that is a weighted sum of the squared differences $(z_{ij} - \hat{z}_{ij})^2$, where $\boldsymbol{z}_i = (z_{i1}, z_{i2}, \ldots, z_{ik})$ is the desired output vector (with a low target value everywhere except at the position corresponding to the target class) and $\hat{\boldsymbol{z}}_i$ is the output vector of the network. A score for class j for pattern i can be directly obtained from the j-th element

$\hat{z}_{ij}$ of the output vector $\hat{\boldsymbol{z}}_i$. When a class must be chosen, the one with the highest score is selected. Let $V_t(i,j) = D_t(i,j)/max_{k \neq y_i} D_t(i,k)$ for $j \neq y_i$ and $V_t(i,y_i) = 1$. These weights are used to give more emphasis to certain incorrect labels according to the Pseudo-Loss Adaboost.

What we call *epoch* is a pass of the training algorithm through all the examples in a training set. In this paper we compare three different versions of AdaBoost:

(R) Training the t-th classifier with a fixed training set obtained by resampling with replacement once from the original training set: before starting training the t-th network, we sample N patterns from the original training set, each time with a probability $P_t(i)$ of picking pattern i. Training is performed for a fixed number of iterations always using this same resampled training set. This is basically the scheme that has been used in the past when applying AdaBoost to decision trees, except that we used the Pseudo-loss AdaBoost. To approximate the Pseudo-loss the training cost that is minimized for a pattern that is the i-th one from the original training set is $\frac{1}{2}\sum_j V_t(i,j)(z_{ij} - \hat{z}_{ij})^2$.

(E) Training the t-th classifier using a different training set at each epoch, by resampling with replacement after each training epoch: after each epoch, a new training set is obtained by sampling from the original training set with probabilities $P_t(i)$. Since we used an on-line (stochastic) gradient in this case, this is equivalent to sampling a new pattern from the original training set with probability $P_t(i)$ before each forward/backward pass through the neural network. Training continues until a fixed number of pattern presentations has been performed. Like for (R), the training cost that is minimized for a pattern that is the i-th one from the original training set is $\frac{1}{2}\sum_j V_t(i,j)(z_{ij} - \hat{z}_{ij})^2$.

(W) Training the t-th classifier by directly weighting the cost function (here the squared error) of the t-th neural network, i.e., all the original training patterns are in the training set, but the cost is weighted by the probability of each example: $\frac{1}{2}\sum_j D_t(i,j)(z_{ij} - \hat{z}_{ij})^2$. If we used directly this formulae, the gradients would be very small, even when all probabilities $D_t(i,j)$ are identical. To avoid having to scale learning rates differently depending on the number of examples, the following "normalized" error function was used:

$$\frac{1}{2} \frac{P_t(i)}{max_k P_t(k)} \sum_j V_t(i,j)(z_{ij} - \hat{z}_{ij})^2 \tag{2}$$

In (E) and (W), what makes the combined networks essentially different from each other is the fact that they are trained with respect to different weightings D_t of the original training set. Rather, in (R), an additional element of diversity is built-in because the criterion used for the t-th network is not exactly the errors weighted by $P_t(i)$. Instead, more emphasis is put on certain patterns while completely ignoring others (because of the initial random sampling of the training set). The (E) version can be seen as a stochastic version of the (W) version, i.e., as the number of iterations through the data increases and the learning rate decreases, (E) becomes a very good approximation of (W). (W) itself is closest to the recipe mandated by the AdaBoost algorithm (but, as we will see below, it suffers from numerical problems). Note that (E) is a better approximation of the weighted cost function than (R), in particular when many epochs are performed. If random resampling of the training data explained a good part of the generalization performance of AdaBoost, then the weighted training version (W) should perform worse than the resampling versions, and the fixed sample version (R) should perform better than the continuously resampled version (E). Note that for Bagging, which directly aims at reducing variance, random resampling is essential to obtain the reduction in generalization error.

3 Results

Experiments have been performed on three data sets: a data set of online handwritten digits, the UCI *Letters* data set of off-line machine-printed alphabetical characters, and the UCI *satellite* data set that is generated from Landsat Multi-spectral Scanner image data. All data sets have a predefined training and test set. All the p-values that are given in this section concern a pair $(\hat{p}_1, \hat{p}_2)$ of test performance results (on n test points) for two classification systems with unknown true error rates p_1 and p_2. The null hypothesis is that the true expected performance for the two systems is not different, i.e., $p_1 = p_2$. Let $\hat{p} = 0.5(\hat{p}_1 + \hat{p}_2)$ be the estimator of the common error rate under the null hypothesis. The alternative hypothesis is that $p_1 < p_2$, so the p-value is obtained as the probability of observing such a large difference under the null hypothesis, i.e., $P(Z > z)$ for a Normal Z, with $z = \frac{\sqrt{n}(\hat{p}_1 - \hat{p}_2)}{\sqrt{2\hat{p}(1-\hat{p})}}$. This is based on the Normal approximation of the Binomial which is appropriate for large n (however, see (Dietterich, 1998a) for a discussion of this and other tests to compare algorithms).

3.1 Results on the online data set

The online data set was collected at Paris 6 University (Schwenk and Milgram, 1996). A WACOM A5 tablet with a cord-less pen was used in order to allow natural writing. Since we wanted to build a writer-independent recognition system, we tried to use many writers and to impose as few constraints as possible on the writing style. In total, 203 students wrote down isolated numbers that have been divided into learning set (1200 examples) and test set (830 examples). Note that the writers of the training and test sets are completely distinct. A particular property of this data set is the notable variety of writing styles that are not equally frequent at all. There are, for instance, 100 zeros written counterclockwise, but only 3 written clockwise. Figure 1 gives an idea of the great variety of writing styles of this data set. We only applied a simple preprocessing: the characters were resampled to 11 points, centered and size-normalized to an (x,y)-coordinate sequence in $[-1, 1]^{22}$.

Table 2 summarizes the results on the test set before using AdaBoost. Note that the dif-

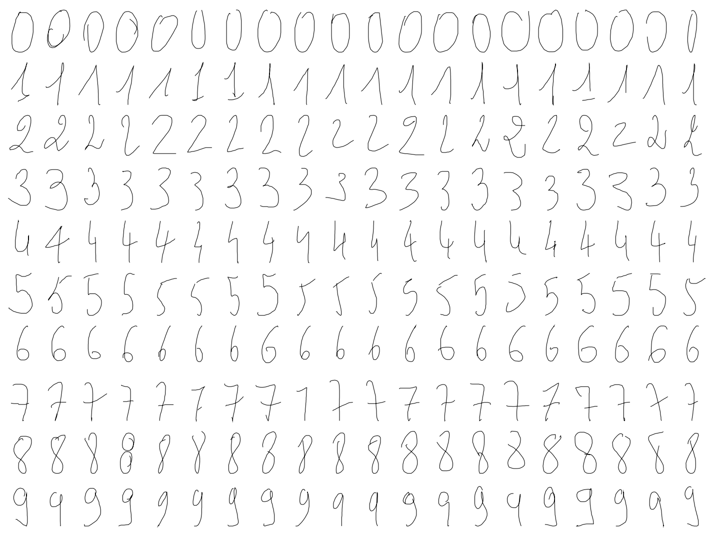

Figure 1: *Some examples of the on-line handwritten digits data set (test set).*

Table 2: *Online digits data set error rates for fully connected MLPs (not boosted).*

architecture[2]	22-10-10	22-30-10	22-50-10	22-80-10
train:	5.7%	0.8%	0.4%	0.08%
test:	8.8%	3.3%	2.8%	2.7%

ferences among the test results on the last three networks are not statistically significant (p-value $>$ 23%), whereas the difference with the first network is significant (p-value $< 10^{-5}$). 10-fold cross-validation within the training set was used to find the optimal number of training epochs (typically about 200). Note that if training is continued until 1000 epochs, the test error increases by up to 1%.

Table 3 shows the results of bagged and boosted multi-layer perceptrons with 10, 30 or 50 hidden units, trained for either 100, 200, 500 or 1000 epochs, 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). In all cases, 100 neural networks were combined.

AdaBoost improved in all cases the generalization error of the MLPs, for instance from 8.8 % to about 2.7 % for the 22-10-10 architecture. Note that the improvement with 50 hidden units from 2.8% (without AdaBoost) to 1.6% (with AdaBoost) is significant (p-value of 0.38%), despite the small number of examples. Boosting was also always superior to Bagging, although the differences are not always very significant, because of the small number

[2]The notation 22-h-10 designates a fully connected neural network with 22 input nodes,

one hidden layer with h neurons and a 10 dimensional output layer.

Table 3: *Online digits test error rates for boosted MLPs.*

architecture	22-10-10			22-30-10			22-50-10		
version:	R	E	W	R	E	W	R	E	W
Bagging: 500 epochs	5.4%			2.8%			1.8%		
AdaBoost:									
100 epochs	2.9%	3.2%	6.0%	1.7%	1.8%	5.1%	2.1%	1.8%	4.9%
200 epochs	3.0%	2.8%	5.6%	1.8%	1.8%	4.2%	1.8%	1.7%	3.5%
500 epochs	2.5%	2.7%	3.3%	1.7%	1.5%	3.0%	1.7%	1.7%	2.8%
1000 epochs	2.8%	2.7%	3.2%	1.8%	1.6%	2.6%	1.6%	1.5%	2.2%
5000 epochs	-	-	2.9%	-	-	1.6%	-	-	1.6%

of examples. Furthermore, it seems that the number of training epochs of each *individual* classifier has no significant impact on the results of the combined classifier, at least on this data set. AdaBoost with weighted training of MLPs (W version), however, doesn't work as well if the learning of each individual MLP is stopped too early (1000 epochs): the networks didn't learn well enough the weighted examples and ϵ_t rapidly approached 0.5. When training each MLP for 5000 epochs, however, the weighted training (W) version achieved the same low test error rate.

AdaBoost is less useful for very big networks (50 or more hidden units for this data) since an individual classifier can achieve zero error on the original training set (using the (E) or (W) method). Such large networks probably have a very low bias but high variance. This may explain why Bagging - a pure variance reduction method - can do as well as AdaBoost, which is believed to reduce bias and variance. Note, however, that AdaBoost can achieve the same low error rates with the smaller 22-30-10 networks.

Figure 2 shows the error rates of some of the boosted classifiers as the number of networks is increased. AdaBoost brings training error to zero after only a few steps, even with an MLP with only 10 hidden units. The generalization error is also considerably improved, and it continues to decrease to an apparent asymptote after zero training error has been reached.

The surprising effect of continuously decreasing generalization error even after training error reaches zero has already been observed by others (Breiman, 1996; Drucker and Cortes, 1996; Freund and Schapire, 1996a; Quinlan, 1996). This seems to contradict Occam's razor, but a recent theorem (Schapire et al., 1997) suggests that the margin distribution may be relevant to the generalization error. Although previous empirical results (Schapire et al., 1997) indicate that pushing the margin cumulative distribution to the right may improve generalization, other recent results (Breiman, 1997a; Breiman, 1997b; Grove and Schuurmans, 1998) show that "improving" the whole margin distribution can also yield to worse generalization. Figure 3 and 4 show several margin cumulative distributions, i.e. the fraction of examples whose margin is at most x as a function of $x \in [-1, 1]$. The networks had be trained for 1000 epochs (5000 for the W version).

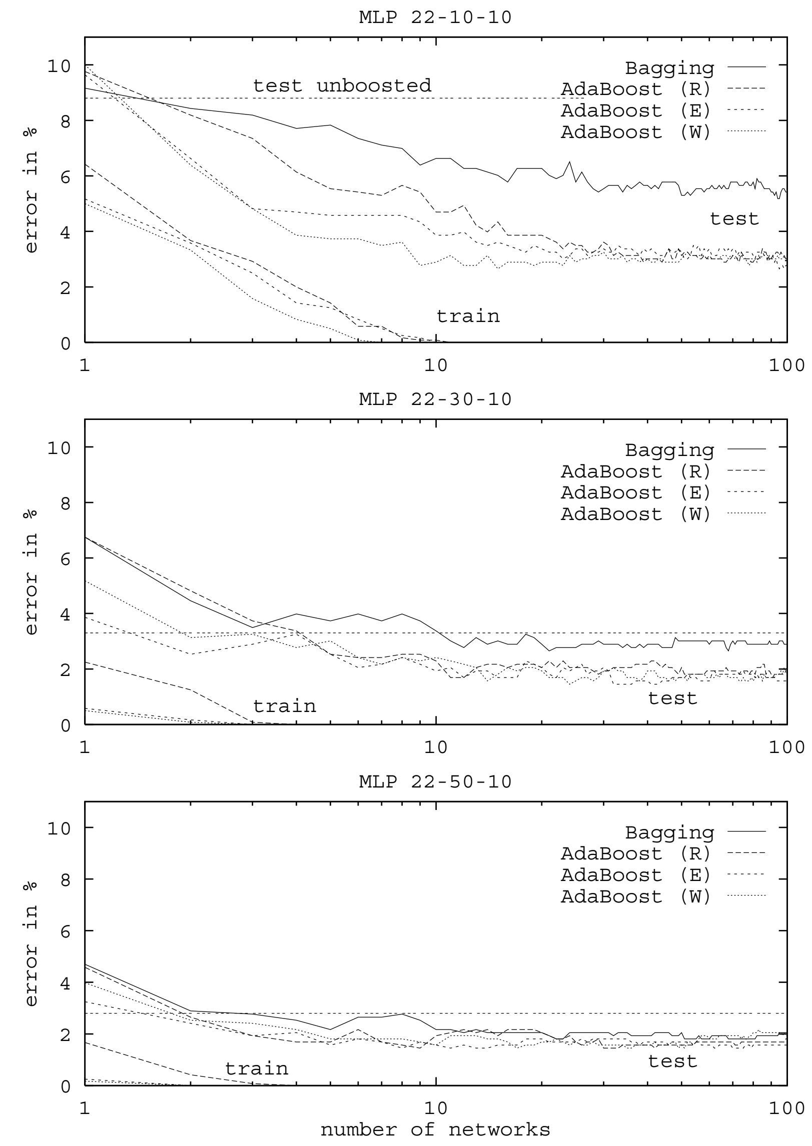

Figure 2: *Error rates of the boosted classifiers for increasing number of networks. For clarity the training error of Bagging is not shown (it overlaps with the test error rates of AdaBoost). The dotted constant horizontal line corresponds to the test error of the unboosted classifier. Small oscillations are not significant since they correspond to few examples.*

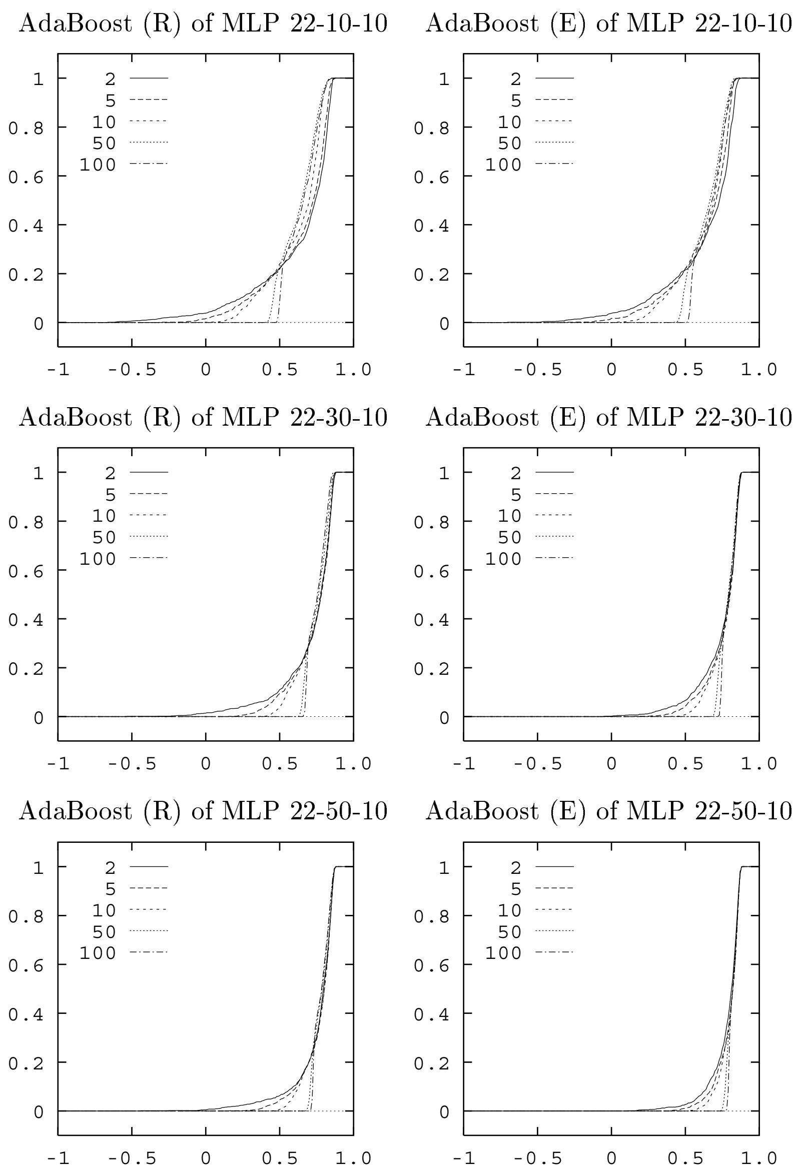

Figure 3: *Margin distributions using 2, 5, 10, 50 and 100 networks, respectively.*

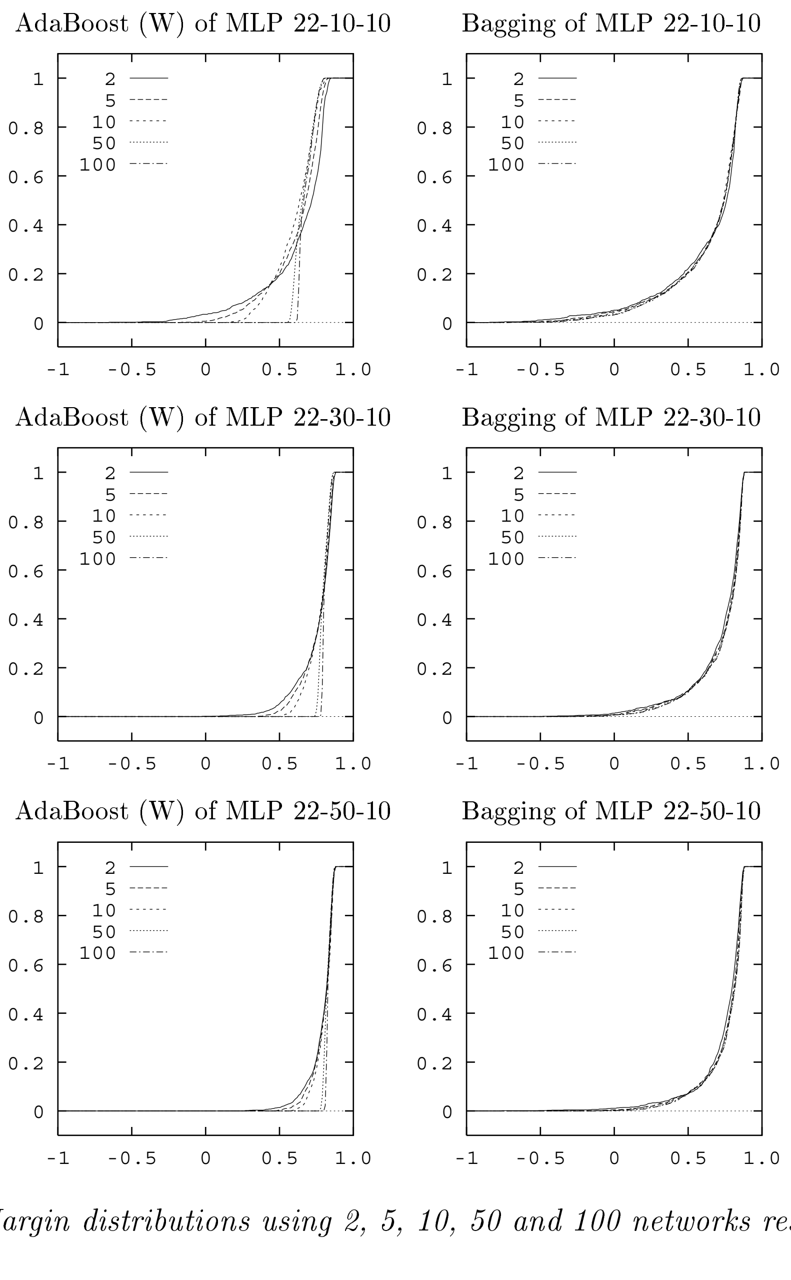

Figure 4: *Margin distributions using 2, 5, 10, 50 and 100 networks respectively.*

It is clear in the Figures 3 and 4 that the number of examples with high margin increases when more classifiers are combined by boosting. When boosting neural networks with 10 hidden units, for instance, there are some examples with a margin smaller than -0.5 when only two networks are combined. However, all examples have a positive margin when 10 nets are combined, and all examples have a margin higher than 0.5 for 100 networks. Bagging, on the other hand, has no significant influence on the margin distributions. There is almost no difference between the margin distributions of the (R), (E) or (W) version of AdaBoost either.[3] Note, however, that there is a difference between the margin distributions and the test set errors when the complexity of the neural networks is varied (hidden layer size). Finally, it seems that sometimes AdaBoost must allow some examples with very high margins in order to improve the minimal margin. This can best beseen for the 22-10-10 architecture.

One should keep in mind that this data set contains only small amounts of noise. In application domains with high amounts of noise, it may be less advantageous to improve the minimal margin at any price (Grove and Schuurmans, 1998; Rätsch et al., 1998), since this would mean putting too much weight to noisy or wrongly labeled examples.

3.2 Results on the UCI Letters and Satimage Data Sets

Similar experiments were performed with MLPs on the "Letters" data set from the UCI Machine Learning data set. It has 16,000 training and 4,000 test patterns, 16 input features, and 26 classes (A-Z) of distorted machine-printed characters from 20 different fonts. A few preliminary experiments on the training set only were used to choose a 16-70-50-26 architecture. Each input feature was normalized according to its mean and variance on the training set.

Two types of experiments were performed: (1) doing resampling after each epoch (E) and using stochastic gradient descent, and (2) without resampling but using re-weighting of the squared error (W) and conjugate gradient descent. In both cases, a fixed number of training epochs (500) was used. The plain, bagged and boosted networks are compared to decision trees in Table 4.

Table 4: *Test error rates on the UCI data sets.*

	CART†			C4.5‡			MLP		
data set	alone	bagged	boosted	alone	bagged	boosted	alone	bagged	boosted
letter	12.4 %	6.4 %	3.4 %	13.8 %	6.8 %	3.3 %	6.1 %	4.3 %	**1.5 %**
satellite	14.8 %	10.3 %	8.8 %	14.8 %	10.6 %	8.9 %	12.8 %	8.7 %	**8.1 %**

† results from (Breiman, 1996)
‡ results from (Freund and Schapire, 1996a)

In both cases (E and W) the same final generalization error results were obtained (1.50% for

[3] One may note that the (W) and (E) versions achieve slightly higher margins than (R).

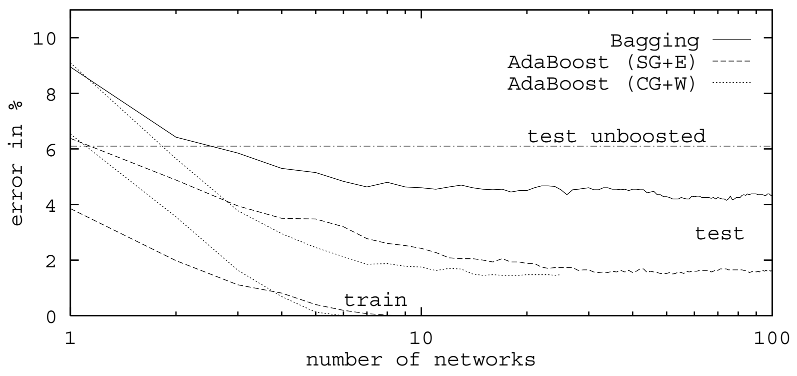

Figure 5: *Error rates of the bagged and boosted neural networks for the UCI letter data set (log-scale). SG+E denotes stochastic gradient descent and resampling after each epoch. CG+W means conjugate gradient descent and weighting of the squared error. For clarity, the training error of Bagging is not shown (it flattens out to about 1.8%). The dotted constant horizontal line corresponds to the test error of the unboosted classifier.*

E and 1.47% for W), but the training time using the weighted squared error (W) was about 10 times greater. This shows that using random resampling (as in E or R) is not necessary to obtain good generalization (whereas it is clearly necessary for Bagging). However, the experiments show that it is still preferable to use a random sampling method such as (R) or (E) for numerical reasons: convergence of each network is faster.

For this reason, for the "E" experiments with stochastic gradient descent, 100 networks were boosted, whereas we stopped training on the "W" network after 25 networks (when the generalization error seemed to have flattened out), which took more than a week on a fast processor (SGI Origin-2000). We believe that the main reason for this difference in training time is that the conjugate gradient method is a batch method and is therefore slower than stochastic gradient descent on redundant data sets with many thousands of examples, such as this one. See comparisons between batch and on-line methods (Bourrely, 1989) and conjugate gradients for classification tasks in particular (Moller, 1992; Moller, 1993). For the (W) version with stochastic gradient descent, the weighted training error of individual networks does not decrease as much as when using conjugate gradient descent, so that AdaBoost itself did not work as well. We believe that this is due to the fact that it is difficult for stochastic gradient descent to approach a minimum when the output error is weighted with very different weights for different patterns (the patterns with small weights make almost no progress). On the other hand, the conjugate gradient descent method can approach a minimum of the weighted cost function more precisely, but inefficiently, when there are thousands of training examples.

The results obtained with the boosted network are extremely good (**1.5%** error, whether using the (W) version with conjugate gradients or the (E) version with stochastic gradient)

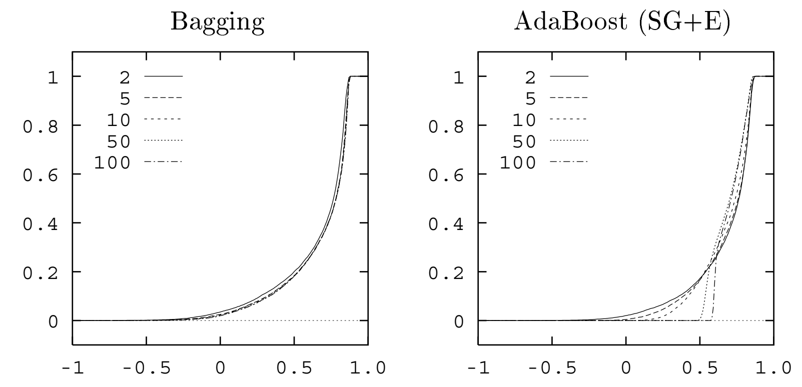

Figure 6: *Margin distributions for the UCI letter data set.*

and are the best ever published to date, as far as the authors know, for this data set. In a comparison with the boosted trees (3.3% error), the p-value of the null hypothesis is less than 10^{-7}. The best performance reported in STATLOG (Feng et al., 1993) is 6.4%. Note also that we need to combine only a few neural networks to get immediate important improvements: with the (E) version, 20 neural networks suffice for the error to fall under 2 %, whereas boosted decision trees typically "converge" later. The (W) version of AdaBoost actually converged faster in terms of number of networks (figure 5: after about 7 networks the 2 % mark was reached, and after 14 networks the 1.5 % apparent asymptote was reached), but converged much slower in terms of training time. Figure 6 shows the margin distributions for Bagging and AdaBoost applied to this data set. Again, Bagging has no effect on the margin distribution, whereas AdaBoost clearly increases the number of examples with large margins.

Similar conclusions hold for the UCI "satellite" data set (Table 4), although the improvements are not as dramatic as in the case of the "Letter" data set. The improvement due to AdaBoost is statistically significant (p-value $< 10^{-6}$) but the difference in performance between boosted MLPs and boosted decision trees is not (p-value $> 21\%$). This data set has 6435 examples, with the first 4435 used for training and the last 2000 used for testing generalization. There are 36 inputs and 6 classes, and a 36-30-15-6 network was used. Again, the two best training methods are the epoch resampling (E) with stochastic gradient or the weighted squared error (W) with conjugate gradient descent.

4 Conclusion

As demonstrated here in three real-world applications, AdaBoost can significantly improve neural classifiers. In particular, the results obtained on the UCI Letters data set (1.5% test error) are significantly better than the best published results to date, as far as the authors know. The behavior of AdaBoost for neural networks confirms previous observations on other learning algorithms, e.g. (Breiman, 1996; Drucker and Cortes, 1996; Freund and

Schapire, 1996a; Quinlan, 1996; Schapire et al., 1997), such as the continued generalization improvement after zero training error has been reached, and the associated improvement in the margin distribution. It seems also that AdaBoost is not very sensitive to over-training of the individual classifiers, so that the neural networks can be trained for a fixed (preferably high) number of training epochs. A similar observation was recently made with decision trees (Breiman, 1997b). This apparent insensitivity to over-training of individual classifiers simplifies the choice of neural network design parameters.

Another interesting finding of this paper is that the "weighted training" version (W) of AdaBoost gives good generalization results for MLPs, but requires many more training epochs or the use of a second-order (and, unfortunately, "batch") method, such as conjugate gradients. We conjecture that this happens because of the weights on the cost function terms (especially when the weights are small), which could worsen the conditioning of the Hessian matrix. So in terms of generalization error, all three methods (R, E, W) gave similar results, but training time was lowest with the E method (with stochastic gradient descent), which samples each new training pattern from the original data with the AdaBoost weights. Although our experiments are insufficient to conclude, it is possible that the "weighted training" method (W) with conjugate gradients might be faster than the others for small training sets (a few hundred examples).

There are various ways to define "variance" for classifiers, e.g. (Kong and Dietterich, 1995; Breiman, 1996; Kohavi and Wolpert, 1996; Tibshirani, 1996). It basically represents how the resulting classifier will vary when a different training set is sampled from the true generating distribution of the data. Our comparative results on the (R), (E) and (W) versions add credence to the view that randomness induced by resampling the training data is not the main reason for AdaBoost's reduction of the generalization error. This is in contrast to Bagging, which is a pure variance reduction method. For Bagging, random resampling is essential to obtain the observed variance reduction.

Another interesting issue is whether the boosted neural networks could be trained with a criterion other than the mean squared error criterion, one that would better approximate the goal of the AdaBoost criterion (i.e., minimizing a weighted classification error). See (Schapire and Singer, 1998) for recent work that addresses this issue.

Acknowledgments

Most of the work was done while the first author was doing a post-doctorate at the University of Montreal. The authors would like to thank the National Science and Engineering Research Council of Canada and the Government of Quebec for financial support.

References

Bauer, E. and Kohavi, R. (1998). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *to appear in Machine Learning.*

Bourrely, J. (1989). Parallelization of a neural learning algorithm on a hypercube. In *Hypercube and distributed computers*, pages 219–229. Elsiever Science Publishing, North Holland.

Breiman, L. (1994). Bagging predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (1996). Bias, variance, and arcing classifiers. Technical Report 460, Statistics Department, University of California at Berkeley.

Breiman, L. (1997a). Arcing the edge. Technical Report 486, Statistics Department, University of California at Berkeley.

Breiman, L. (1997b). Prediction games and arcing classifiers. Technical Report 504, Statistics Department, University of California at Berkeley.

Breiman, L. (1998). Arcing classifiers. *Annuals of Statistics*, 26(3):801–849.

Dietterich, T. (1998a). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1924.

Dietterich, T. G. (1998b). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *submitted to Machine Learning.*
available at `ftp://ftp.cs.orst.edu/pub/tgd/papers/tr-randomized-c4.ps.gz`.

Drucker, H. and Cortes, C. (1996). Boosting decision trees. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems*, pages 479–485. MIT Press.

Feng, C., Sutherland, A., King, R., S.Muggleton, and Henery, R. (1993). 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.

Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285.

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

Freund, Y. and Schapire, R. E. (1996b). Game theory, on-line prediction and boosting. In *Proceedings of the Ninth Annual Conference on Computational Learning Theory*, pages 325–332.

Freund, Y. and Schapire, R. E. (1997). A decision theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Science*, 55(1):119–139.

Freund, Y. and Schapire, R. E. (1998). Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, to appear.

Friedman, J., Hastie, T., and Tibshirani, R. (1998). Additive logistic regression: a statistical view of boosting. Technical report, Department of Statistics, Stanford University.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58.

Grove, A. J. and Schuurmans, D. (1998). Boosting in the limit: Maximizing the margin of learned ensembles. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. to appear.

Kohavi, R. and Wolpert, D. H. (1996). Bias plus variance decomposition for zero-one loss functions. In *Machine Learning: Proceedings of Thirteenth International Conference*, pages 275–283.

Kong, E. B. and Dietterich, T. G. (1995). Error-correcting output coding corrects bias and variance. In *Machine Learning: Proceedings of Twelfth International Conference*, pages 313–321.

Krogh, A. and Vedelsby, J. (1995). Neural network ensembles, cross validation and active learning. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7*, pages 231–238. MIT Press.

Maclin, R. and Opitz, D. (1997). An empirical evaluation of bagging and boosting. In *Proceedings of the Fourteenth National Conference on Artificial Intelligenc*, pages 546–551.

Mason, L. and Baxter, P. B. J. (1999). Direct optimization of margins improves generalization in combined classifiers. In TODO, editor, *Advances in Neural Information Processing Systems 11*. MIT Press. in press.

Moller, M. (1992). Supervised learning on large redundant training sets. In *Neural Networks for Signal Processing 2*. IEEE press.

Moller, M. (1993). *Efficient Training of Feed-Forward Neural Networks*. PhD thesis, Aarhus University, Aarhus, Denmark.

Perrone, M. P. (1993). *Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure Optimization*. PhD thesis, Brown University, Institute for Brain and Neural Systems.

Perrone, M. P. (1994). Putting it all together: Methods for combining neural networks. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems*, volume 6, pages 1188–1189. Morgan Kaufmann Publishers, Inc.

Quinlan, J. R. (1996). Bagging, boosting and C4.5. In *Machine Learning: Proceedings of the fourteenth International Conference*, pages 725–730.

Rätsch, G., Onoda, T., and Müller, K.-R. (1998). Soft margins for adaboost. Technical Report NC-TR-1998-021, Royal Holloway College.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5(2):197–227.

Schapire, R. E. (1999). Theoretical views of boosting. In *Computational Learning Theory: Fourth European Conference, EuroCOLT.* to appear.

Schapire, R. E., Freund, Y., Bartlett, P., and Lee, W. S. (1997). Boosting the margin: A new explanation for the effectiveness of voting methods. In *Machine Learning: Proceedings of Fourteenth International Conference*, pages 322–330.

Schapire, R. E. and Singer, Y. (1998). Improved boosting algorithms using confidence rated predictions. In *Proceedings of the 11th Annual Conference on Computational Learning Theory.*

Schwenk, H. and Bengio, Y. (1997). Adaboosting neural networks: Application to on-line character recognition. In *International Conference on Artificial Neural Networks*, pages 967 – 972. Springer Verlag.

Schwenk, H. and Bengio, Y. (1998). Training methods for adaptive boosting of neural networks. In Jordan, M. I., Kearns, M. J., and Solla, S. A., editors, *Advances in Neural Information Processing Systems*, pages 647 –653. The MIT Press.

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

Tibshirani, R. (1996). Bias, variance and prediction error for classification rules. Technical report, Departement od Statistics, University of Toronto.