AdaBoosting Neural Networks: Application to on-line Character Recognition

Holger Schwenk[1] and Yoshua Bengio[1,2]
email: {schwenk,bengioy}@iro.umontreal.ca

[1] University of Montreal, 2920 Chemin de la tour, Qc, H3C 3J7, Canada,
[2] AT&T Bell Laboratories, Holmdel, NJ

Abstract. "Boosting" is a general method for improving the performance of any weak 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* [4]. It has been applied with great success to several benchmark machine learning problems using rather simple learning algorithms [3], in particular decision trees [1,2,5]. In this paper we use AdaBoost to improve the performances of a strong learning algorithm: a neural network based on-line character recognition system. In particular we will show that it can be used to learn automatically a great variety of writing styles even when the amount of training data for each style varies a lot. Our system achieves about 1.4 % error on a handwritten digit data base of more than 200 writers.

1 Introduction

AdaBoost [3,4] constructs a composite classifier by sequentially training classifiers. 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. 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. In this case, examples with high probability 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. Neural networks can be trained directly with respect to a distribution over the learning data by weighting the cost function. 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 less than 50%, i.e., better than chance in the 2-class case. There is also a version, called *errorloss*-AdaBoost, that is meant for multi-class problems for which confidence scores for each class are available. Due to lack of space, we give only the algorithm (see figure 1, right) and we refer the reader to the references for more details [3,4].

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

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 $i$

Repeat:
  1. Train neural network with respect
     to distribution $D_t$ and obtain
     hypothesis $h_t : X \to Y$
  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 $\epsilon_t > \frac{1}{2}$
  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 $\delta_i = 1$ if $h_t(x_i) = y_i$ and 0 otherwise
     where $Z_t$ is a normalization constant

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

```
Init: let $B = \{(i, y) : i \in \{1, \ldots, N\}, y \neq y_i\}$
      $D_1(i, y) = 1/|B|$ 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 $D_t$
     $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \beta_t^{\frac{1}{2}((1 + h_t(x_i, y_i) - h_t(x_i, y))}$
     where $Z_t$ is a normalization constant

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

Fig. 1. AdaBoost algorithm (left) and multi-class extension using confidence scores (right)

fast to zero [4]. More importantly, however, bounds on the *generalization error* of such a system have been formulated [6]. These are based on a notion of *margin* of classification, defined as the difference between the score of the correct class and the best 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 [6] (and other classifiers obtained by convex combinations of other classifiers). Let H be a set of hypotheses (from which the h_t are chosen), with VC-dimenstion 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 $1 - \delta, \delta > 0$, over the random choice of the training set S from D, the following bound is satisfied for all $\theta > 0$:

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

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 [6].

AdaBoost has been applied to rather weak learning algorithms (with low capacity) [3] and to decision trees [1,2,5], and not yet, until now, to the best of our knowledge, to artificial neural networks.

2 Architecture of 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.

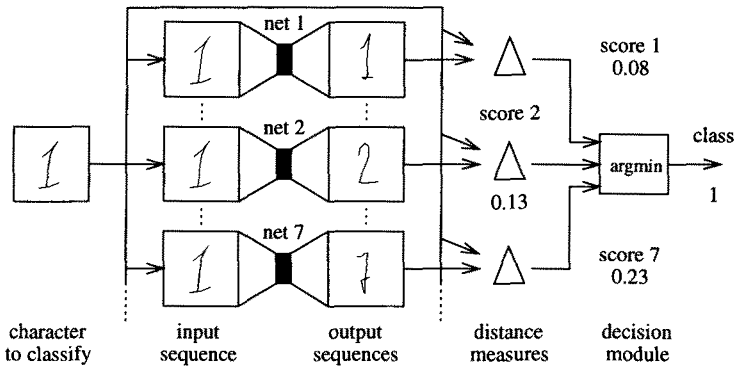

Fig. 2. Architecture of a Diabolo classifier

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).

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 [7], we have shown that the well-known tangent-distance [10] can be used in the objective function of the autoencoders. Furthermore, we can define a discriminant learning algorithm [8]: 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, similar to LVQ2. Interestingly, this intuitive algorithm has gained a theoretical justification by the recent theorem of equation 1, since this discriminant learning algorithm decreases the number of examples with low margin. This Diabolo classifier has achieved state-of-the-art results in off-line handwritten character recognition [7,8].

Recently, we have also extended the idea of a transformation invariant distance measure to on-line character recognition [9]. 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 [9]. 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.

3 Experimental Results

All experiments have been performed with a data set acquired at Paris 6 University. 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 (see figure 3). The characters were resampled to 11 points, centered and size normalized to a time sequence of (x,y)-coordinates. For the MLPs each input feature was normalized according to its mean and variance on the training set.

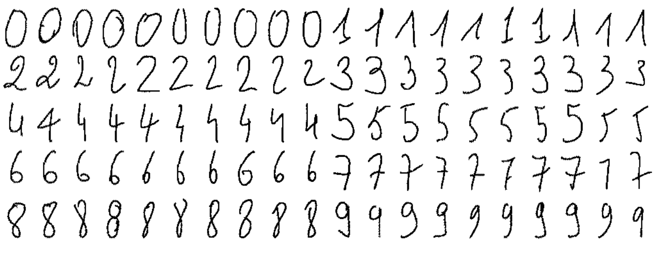

Fig. 3. Some examples of our database (test set).
The different stroke orders and writing directions are not shown.

We used the multi-class version of AdaBoost with MLPs: it performed consistently better than the simple version. However, the multi-class version was not useful with the Diabolo classifier. We used the sampling approximation rather than weighting the cost function in all the experiments. We also made some minor changes to the AdaBoost algorithm in our application. 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), therefore incorporating prior knowledge on the task.

Table 1. 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 (even without hand-selected sub-classes in the training set) performs quite well with respect to the multi-layer perceptrons. 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.

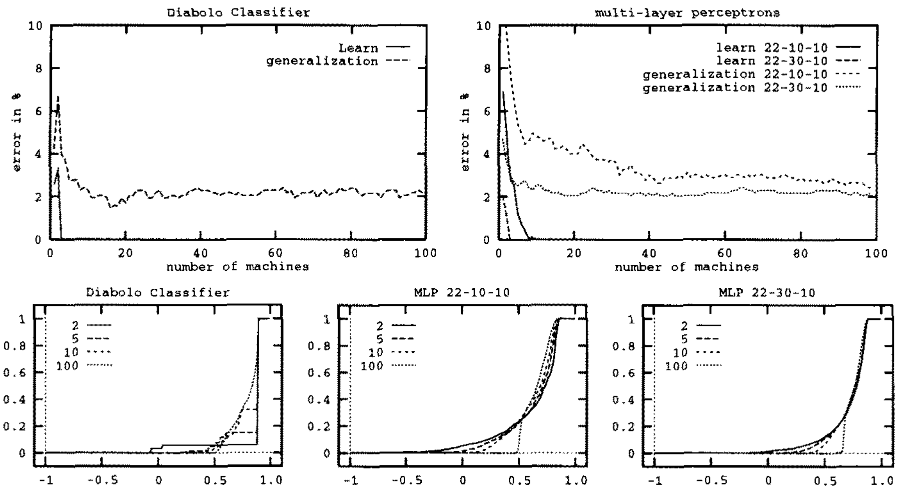

Fig. 4. top: error rates of the boosted classifiers
bottom: margin distributions using 2, 5, 10 and 100 machines respectively

Figure 4 shows the error rate 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% = 12 errors) which is in practice as good as the Diabolo classifier using hand-selected subclasses (1.2% = 10 errors).

The surprising effect of continuously decreasing generalisation error has already been observed by others when using AdaBoost with decision trees [1–3,5]. This seems to contradict Occam's razor, but it can be explained by the very recently proven theorem of Schapire et al. [6]: the bound on the generalization error (equation 1) 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 4 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 almost all examples have a margin higher than 0.5. 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 boosted Diabolo classifier. Since the margin distribution for 100 machines of the MLP 22-30-10 seems to be better than the one of the Diabolo classifier, 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. Note, finally, that the initial margin distribution of the Diabolo classifier is much better than those of the MLPs: only few examples have a low or even negative margin while all the others have a margin of more than 0.8.

4 Conclusion

As demonstrated in our handwriting recognition application, 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 [1–3,5], such as the continued generalization improvement after zero training error has been reached, and the associated improvement in the margin distribution. Note however that even after boosting the Diabolo classifier is still better than MLP networks. This suggests that even a very good learning algorithm should be complemented by incorporation of a-priori knowledge.

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. Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of Thirteenth International Conference*, pages 148–156, 1996.
4. 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.
5. J.R. Quinlan. Bagging, Boosting and C4.5. In *14th Ntnl Conf. on Artificial Intelligence*, 1996.
6. 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.
7. H. Schwenk and M. Milgram. Transformation invariant autoassociation with application to handwritten character recognition. *NIPS*7*, pages 991–998. MIT Press, 1995.
8. H. Schwenk and M. Milgram. Learning discriminant tangent models for handwritten character recognition. In *ICANN*96*, pages 585–590. Springer Verlag, 1995.
9. H. Schwenk and M. Milgram. Constraint tangent distance for online character recognition. In *International Conference on Pattern Recognition*, pages D 520–524, 1996.
10. P. Simard, Y. Le Cun, and J. Denker. Efficient pattern recognition using a new transformation distance. *NIPS*5*, pages 50–58. Morgan Kaufmann, 1993.