Training Methods for Adaptive Boosting of Neural Networks

Holger Schwenk
Dept. IRO
Université de Montréal
2920 Chemin de la Tour,
Montreal, Qc, Canada, H3C 3J7
schwenk@iro.umontreal.ca

Yoshua Bengio
Dept. IRO
Université de Montréal
and *AT&T Laboratories, NJ*
bengioy@iro.umontreal.ca

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], and decision trees [1, 2, 6]. In this paper we use AdaBoost to improve the performances of neural networks. 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 1.5% error on the UCI Letters and 8.1% error on the UCI satellite data set.

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). In doing so, we also compare three dif-

ferent versions of AdaBoost: (R) training each classifier with a fixed training set obtained by resampling with replacement 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 (E) is a better approximation of the weighted cost function than the first one (R), in particular when many epochs are performed. If the variance reduction induced by averaging the hypotheses from very different models explains a good part 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. 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. This is particularly true in the simple resampling version (labeled "R" earlier), and 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* $h_t : X \to Y$ 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 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. 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 [4, 5].

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] as the number of combined classifiers is increased. 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 any classifier obtained by a convex combination of a set of classifiers). Let H be a set of hypotheses (from which the h_t hare 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 $1 - \delta$ over the random choice of the training set S from D, the following bound is satisfied for all $\theta > 0$:

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

Note that this bound is independent of the number of combined hypotheses and how they

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 $\delta_i = (h_t(x_i) = y_i)$ and Z_t 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 $B = \{(i, y) : i \in \{1, ..., N\}, y \neq y_i\}$ $D_1(i, y) = 1/|B|$ 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:

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

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

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. Figure 2 summarizes the basic architecture. It shows also typical classification behavior for an online character recognition task. 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.

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. 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

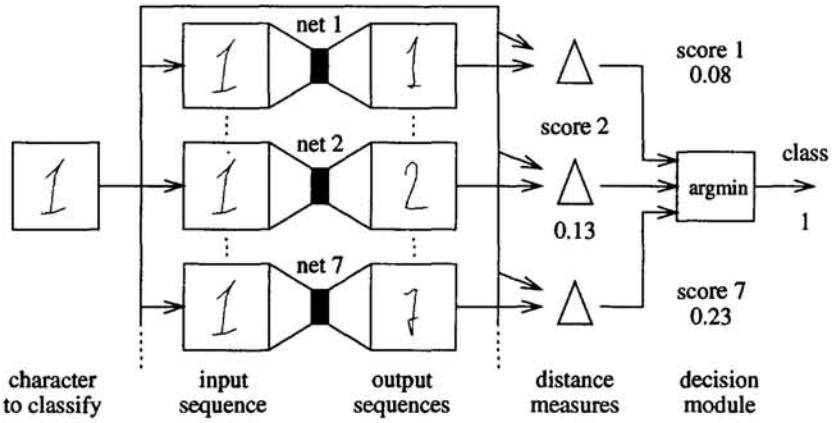

Figure 2: Architecture of a Diabolo classifier

measure to online 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 three data sets: a data base of online handwritten digits, the UCI *Letters* database of offline machine-printed alphabetical characters and the UCI *satellite* database that is generated from Landsat Multi-spectral Scanner image data. All 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. 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 with tangent distance [10] is invariant to small transformations we don't need to extract further features.

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.