IDIAP RESEARCH REPORT

IDIAP

Martigny - Valais - Suisse

DynaBoost: Combining Boosted Hypotheses in a Dynamic Way

Perry Moerland * Eddy Mayoraz

IDIAP–RR 99-09

May 99

submitted for publication

Dalle Molle Institute for Perceptual Artificial Intelligence • P.O.Box 592 • Martigny • Valais • Switzerland

phone +41 – 27 – 721 77 11
fax +41 – 27 – 721 77 12
e-mail secretariat@idiap.ch
internet http://www.idiap.ch

* e-mail: Perry.Moerland@idiap.ch

IDIAP Research Report 99-09

DynaBoost: Combining Boosted Hypotheses in a Dynamic Way

Perry Moerland Eddy Mayoraz

May 99

submitted for publication

Abstract. We present an extension of Freund and Schapire's AdaBoost algorithm that allows an input-dependent combination of the base hypotheses. A separate weak learner is used for determining the input-dependent weights of each hypothesis. The error function minimized by these additional weak learners is a margin cost function that has also been shown to be minimized by AdaBoost. The weak learners used for dynamically combining the base hypotheses are simple perceptrons. We compare our dynamic combination model with AdaBoost on a range of binary and multi-class classification problems. It is shown that the dynamic approach significantly improves the results on most data sets when (rather weak) perceptron base hypotheses are used, while the difference in performance is small when the base hypotheses are MLPs.

1 Introduction

Ensemble methods such as bagging [1] and boosting [2] operate by taking a base algorithm and invoking it repeatedly with different training sets. A main characteristic of boosting is that it maintains a distribution on the original training sample and sequentially adapts this distribution to emphasize patterns that have been frequently misclassified. Boosting algorithms, such as AdaBoost [2] and its variants have shown very good performance on a wide range of classification problems and with many different base algorithms (also called weak learners) [3–5].

The combination of the different hypotheses generated by a boosting algorithm is often based on some form of weighted voting. In AdaBoost, for example, the higher weights are given to the hypotheses with the lower (weighted on the hypothesis' distribution of the training sample) error. An alternative for using fixed weights when combining the base hypotheses, is to allow the weights to be input-dependent. This idea can be traced back to Quinlan [4] who proposed to define a confidence measure for an input pattern when using C4.5 as base hypothesis. This heuristic led to an improvement on a large majority of the data sets evaluated by Quinlan. A related approach is described in [6] where MLPs are dynamically combined using a confidence measure based on the difference between the highest and the second highest output of a MLP on a multi-class problem. Waterhouse used boosting to initialise a mixture of MLP experts and then retrained these initial experts and a randomised gate in the mixture of experts framework [7]. The gating network, thus, provides a dynamic combination of the base hypotheses. On a large speech recognition problem, Waterhouse observed that only when both experts and the gate were retrained, performance improved with respect the boosted model used for initializing the experts.

In this paper, we present a more principled approach for dynamically combining boosted hypotheses (called *Dynaboost*). Our approach is based on the recent idea that various boosting algorithms can be interpreted as a stage-wise gradient descent optimization of a cost function of the *margins* [8–11]. The margin concept plays an important role in explaining the resistance to overfitting of boosting algorithms: larger margins imply lower generalization error [12]. Schapire *et al.* also show that AdaBoost tends to increase the margins of the training examples.

Our DynaBoost algorithm extends other boosting algorithms by using a separate weak learner for determining the input-dependent weights of each hypothesis. This means that the combined DynaBoost hypothesis resembles a mixture of experts with a gating network dynamically combining the base hypotheses. The error function minimized by the weak learners in the gating network is exactly the cost function of the margins that has been shown to be minimized by AdaBoost. In section 2, we recall the definition of AdaBoost and describe how to extend it to DynaBoost. The error function of the margins is described in more detail.

In section 3, we describe the experimental results for the DynaBoost algorithm and compare it with AdaBoost on a range of binary and multi-class classification problems from the UCI repository [13]. In all experiments discussed here, the weak learners used for dynamically combining the base hypotheses are simple perceptrons. It is shown that the dynamic approach of DynaBoost significantly improves the results on most data sets when rather weak base hypothesis are used (perceptrons in this case). With MLP base hypotheses the difference in performance between DynaBoost and AdaBoost is small.

In section 4, we point out various directions for future research. This is very much "work in progress" and both on the theoretical and experimental side, there is still much research to be done to fully understand the possible benefits and drawbacks of the DynaBoost approach.

2 AdaBoost and DynaBoost

Algorithm 1 gives the pseudo-code for the AdaBoost algorithm. It takes as input a training set of m examples $S = \{(x_1, t_1), \ldots, (x_m, t_m)\}$ where x_i is drawn from the input space X and $t_i \in Y$ is the class label associated with x_i. AdaBoost starts with a uniform weighting D_1 of the training data and adapts the weighting such that the misclassified examples get more weight (eq. (1) in Algorithm 1).

Algorithm 1 : AdaBoost.M1 for k-class problems [2]

Require:

- A training set S of m examples: $\{(x_1,t_1),\ldots,(x_m,t_m)\}$ with labels $t_i \in Y = \{1,\ldots,k\}$.
- A weak learning algorithm $L(S,D)$ that returns a base hypothesis which minimizes the weighted (on a distribution D) error on the training set S.

Ensure: Combined hypothesis $H_j(x) = \arg\max_{t\in Y} \sum_{i\le j: h_i(x)=t} \alpha_i$.

$D_1(n) := 1/m$ for all $n = 1,\ldots,m$
for $j := 1$ to T **do**
  $h_j := L(S,D)$
  $\varepsilon_j := \sum_{n:h_j(x_n)\neq t_n} D_j(n)$ $\{\varepsilon_j$ is the weighted error of $h_j\}$
  if $\varepsilon_j \ge 1/2$ **then**
    return H_{j-1}
  end if
  $\alpha_j := \frac{1}{2}\ln(\frac{1-\varepsilon_j}{\varepsilon_j})$ $\{\alpha_j > 0\}$
  $Z_j := 2\sqrt{\varepsilon_j(1-\varepsilon_j)}$
  for all $n = 1,\ldots,m$ **do**

$$D_{j+1}(n) := \begin{cases} D_j(n)\, e^{-\alpha_j}/Z_j & \text{if } h_j(x_n) = t_n \\ D_j(n)\, e^{\alpha_j}/Z_j & \text{if } h_j(x_n) \neq t_n \end{cases} \qquad (1)$$

    $\{Z_j$ is a normalization constant$\}$
  end for
end for

On round j, a base hypothesis $h_j : X \to Y$ is trained to minimize the error on the training data weighted according to the distribution D_j. The hypothesis weights α_j are chosen such that greater weight is given to hypothesis with lower weighted error (that is, ε_j). The combined hypothesis is a weighted vote over the base hypotheses.

One of the nice properties of AdaBoost is that if the base hypotheses have error ε_j smaller than $1/2$, then the training error of the combined hypothesis goes to zero exponentially fast [2]. While, this theorem only addresses the training error, there is also strong experimental evidence that AdaBoost is quite robust to overfitting and generalizes well [3,4]. Recent explanations of this phenomenon are based on the notion of *margins*. The margin of an example is defined as the difference between the total weight assigned to the correct label and the total weight assigned to a wrong label[1]:

$$m(H_j,x) = \left(\sum_{i\le j:h_i(x)=t} \alpha_i\right) - \left(\sum_{i\le j:h_i(x)\neq t} \alpha_i\right). \qquad (2)$$

In the two-class case when encoding the two class labels as -1 and 1 respectively, this reduces to the compact: $m(H_j,x) = t\sum_{i=1}^{j} \alpha_i h_i(x) = tH_j(x)$. Schapire *et al.* have shown that larger margins imply lower generalization error and that AdaBoost tends to increase the margins of the training examples. This is intuitively clear when comparing the update of D (eq. (1)) with the definition of the margin (2): most weight is placed on the examples for which the the margin is smallest.

Another recent break-trough in the understanding of boosting algorithms, has been the proof that various boosting algorithms can be interpreted as a stage-wise gradient descent optimization of cost

[1]This is actually a lower bound on the definition of the margin in Schapire *et al.* [14].

functions of the margins [8–11]. The margin cost function for AdaBoost.M1 at round j is:

$$C_j = \sum_{n=1}^{m} e^{-m(H_j,x_n)} \tag{3}$$

and it can be shown that the updates of the distribution D_j, the hypothesis weights α_j and the stopping criteria on ε_j, all follow from a gradient descent optimization of the margin cost function defined in (3) (see, for example, [11]).

Now, we are ready to replace the fixed hypothesis weights α_j of the AdaBoost algorithm, by input-dependent weights $\alpha_j(x)$ and present the DynaBoost algorithm. The combined hypothesis becomes:

$$H_j^{\mathrm{Dyna}}(x) = \arg\max_{t\in Y} \sum_{i\leq j:h_i(x)=t} \alpha_i(x).$$

The main problem is how to determine $\alpha_j(x)$. We decided to use separate weak learners for determining the input-dependent weights: $\alpha_j(x) = g_j(x, \vartheta_j)$, with parameters ϑ_j. The error function that has to be minimized for these new weak learners is readily obtained by generalizing (2) and (3):

$$C_j^{\mathrm{Dyna}} = \sum_{n=1}^{m} e^{\left(\sum_{i\leq j:h_i(x_n)\neq t_n} g_i(x_n,\vartheta_i)\right) - \left(\sum_{i\leq j:h_i(x_n)=t_n} g_i(x_n,\vartheta_i)\right)} = \sum_{n=1}^{m} C_{j,n}^{\mathrm{Dyna}}, \tag{4}$$

where we have defined the short-hand $C_{j,n}^{\mathrm{Dyna}}$ for the error of base hypothesis j on pattern n. The error function that has to be minimized on round $j+1$ can then be written as the sum of the old $C_{j,n}^{\mathrm{Dyna}}$ multiplied by an exponential factor:

$$C_{j+1}^{\mathrm{Dyna}} = \sum_{n=1}^{m} C_{j,n}^{\mathrm{Dyna}} e^{-g_{j+1}(x_n,\vartheta_i) I(h_{j+1}(x_n)=t_n)}, \tag{5}$$

where $I(\text{true}) = 1$ and $I(\text{false}) = -1$. If we choose the $\alpha_{j+1}(x) = g_{j+1}(x, \vartheta_{j+1})$ to be differentiable with respect to its parameters ϑ_{j+1}, gradient-based optimization algorithms can be used for minimizing (5).

In the experiments described in section 3, we choose a perceptron with a logistic output function for each weak learner determining the input-dependent hypothesis weights:

$$g_j(x, \vartheta_j) = \sigma(\theta_j \cdot x).$$

The choice of the activation function is motivated by the fact that the fixed hypothesis weights α_j in AdaBoost are strictly positive. The lower asymptote of the logistic function guarantees that this is also the case for the $\alpha_j(x)$ in DynaBoost. The higher asymptote of the logistic function has no counterpart in AdaBoost: α_j as defined in Algorithm 1 can grow arbitrarily large when $\varepsilon_j \to 1/2$. The higher asymptote may nevertheless be useful since allowing large values for $\alpha_j(x)$ could correspond to overly confident predictions and a certain risk of overfitting. The main motivation for using a perceptron is that is an easy to learn rather weak learner that can be trained with gradient-based methods. Another reason is that we are also interested in links between DynaBoost and mixtures of experts: the combined DynaBoost hypothesis resembles a mixture of experts with a gating network dynamically combining the base hypotheses. The combined hypothesis obtained by DynaBoost could, for example, be retrained in the framework of mixtures of experts as in [7].

Thus, the DynaBoost algorithm is similar to Algorithm 1 but with all α_j replaced by $\alpha_j(x)$ and the update of $\alpha_j(x)$ consists of a gradient-based optimization of a margin cost function (5). This cost function is minimized by putting more weight on correct predictions and $C_{j,n}^{\mathrm{Dyna}}$ plays a role similar to $D_j(n)$: most weight is placed on the difficult examples.[2]

[2] Actually $D_j(n)$ is obtained by normalizing $C_{j,n}^{\mathrm{Dyna}}$.

3 Experiments

The experiments were performed on 9 data sets from the UCI repository [13] and the banana toy data set.[3] On all data sets, 5x2 cross-validation [15] has been used and the results presented in this section are the average over the 10 outcomes.

We compare AdaBoost and DynaBoost using two different base hypotheses: perceptrons and MLPs with 10 hidden neurons. In all cases, we performed boosting by sampling where examples are drawn with replacement from the training set with a probability proportional to the distribution D. The output function for the base hypotheses was either a logistic function for two-class problems or a soft-max for multi-class problems. The error function minimized by the base hypotheses was cross-entropy. The base hypotheses and the perceptrons determining the input-dependent hypothesis weights (for DynaBoost) were trained for 30 iterations of the scaled conjugate gradient algorithm [16]. The number of rounds of boosting varies with the datasets and was chosen to be 100, 200, or 500.

These experiments are the very first executions of DynaBoost and many more have to be done in order to allow more robust conclusions on the performance of the method. Nevertheless, some general comments can be safely made on the base of these experiments.

Figure 1 clearly illustrates that the behavior of DynaBoost is quite similar to the one of AdaBoost when strong learners are used as hypotheses (e.g. MLPs, see the 10 plots on the right-hand half of the figure). On the contrary, important differences are observed between the two methods when weak learners are used.

Due to space constraints, the results on the training set are not presented in this paper. The training error for DynaBoost always converged to 0, usually much faster than the one for AdaBoost. This is not surprising since the use of dynamic weights considerably increases the power of the learning model. The drawback of a powerful model is the increased risk of overfitting. Interestingly enough, none of the ten experiments with MLP base hypotheses show any overfitting behavior.

The analysis is more complex in the case of weak learners. An impressive improvement is obtained with the use of dynamic weights in four cases, without any overfitting: banana, vowel, glass and segmentation. In two other cases, there are no significant differences between DynaBoost and AdaBoost (sonar and breast-cancer-wisconsin). In the case of soybean-large, ionosphere and optical-recognition, the initial behavior of DynaBoost is good but after some rounds the model starts to overfit. In the first two cases, the error of DynaBoost after a few rounds is smaller than the smallest one achieved by AdaBoost and after a large number of rounds the two errors are comparable. This means, in particular, that the use of a validation set to stop the boosting process adequately would have given a small and well performing learning model with dynamic weights. Finally, in the case of pima-indians-diabetes, which is known to have a high level of noise, DynaBoost overfits dramatically, while AdaBoost overfits only slightly.

4 Conclusions and Discussion

The DynaBoost algorithm seems an interesting extension of Freund and Schapire's AdaBoost that can improve upon AdaBoost when the base hypotheses are relatively weak. The use of dynamic weights considerably increases the power of the learning model and the training error for DynaBoost usually converges to zero much faster than for AdaBoost. The drawback of a powerful model is the increased risk of overfitting, but the experimental results indicate that in many cases DynaBoost does not overfit.

There is still much research to be done to fully understand the possible benefits and drawbacks of the DynaBoost approach. Directions for future research include the investigation of the convergence properties of DynaBoost: can it be shown that the training error of the combined hypothesis goes to zero as is the case for AdaBoost? Can something be said about the generalization error of DynaBoost? On the experimental side, more experiments have to be done in order to allow more robust conclusions on the performance of the method. This can include the use of other base hypotheses, such as stumps

[3] A script for generating the banana data has been made available by Gunnar Rätsch: www.first.gmd.de/raetsch/data/

and general decision trees. It would also be interesting to pursue other choices for the weak learners used for dynamically combining the base hypotheses. Finally, the combined hypothesis obtained by DynaBoost could also be retrained in the framework of mixtures of experts to improve performance even further.

Acknowledgments

The authors gratefully acknowledge the Swiss National Science Foundation (FN:21-45621.95) for their support of this research.

References

[1] Leo Breiman. Bagging predictors. *Machine Learning*, 26(2):123–140, 1996.

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

[3] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156, San Francisco, CA, 1996. Morgan Kaufmann.

[4] J. R. Quinlan. Bagging, boosting and C4.5. In *Proceedings of the Thirteenth National on Artificial Intelligence*, pages 725–730, Cambridge, MA, 1996. AAAI Press/MIT Press.

[5] H. Schwenk and Y. Bengio. Training methods for adaptive boosting of neural networks. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10, pages 647–653, Cambridge MA, 1998. MIT Press.

[6] R. Avnimelech and N. Intrator. Boosted mixture of experts: an ensemble learning scheme. *Neural Computation*, 11(2):483–498, February 1999.

[7] S. R. Waterhouse and G. D. Cook. Ensemble methods for phoneme classification. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, pages 800–806, Cambridge MA, 1997. MIT Press.

[8] L. Breiman. Arcing the edge. Technical Report 486, Statistics Department, University of California, Berkeley, 1997. Available from: `ftp://ftp.stat.berkeley.edu/pub/users/breiman/arcing-the-edge.ps.Z`.

[9] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. Technical report, Department of Statistics, Stanford University, 1998.
Available from: `ftp://utstat.toronto.edu/pub/tibs/boost.ps.Z`.

[10] G. Rätsch, T. Onoda, and K. R. Müller. Soft margins for AdaBoost. Technical Report NC-TR-1998-012, NeuroColt2 Technical Report Series, 1998.
Available from: `www.first.gmd.de/~raetsch/Neurocolt_Margin.ps.gz`.

[11] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent in function space. Technical report, Australian National University, Canberra, Australia, 1999.
Available from: `wwwsyseng.anu.edu.au/~jon/papers/doom2.ps.gz`.

[12] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26(5):1651–1686, 1998.

[13] C. Blake, E. Keogh, and C. J. Merz. UCI repository of machine learning databases. Irvine: University of California, Department of Information and Computer Sciences. `www.ics.uci.edu/~mlearn/`, 1998.

[14] A. Grove and D. Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 692–699, 1998.

[15] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.

[16] M. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525–533, 1993.

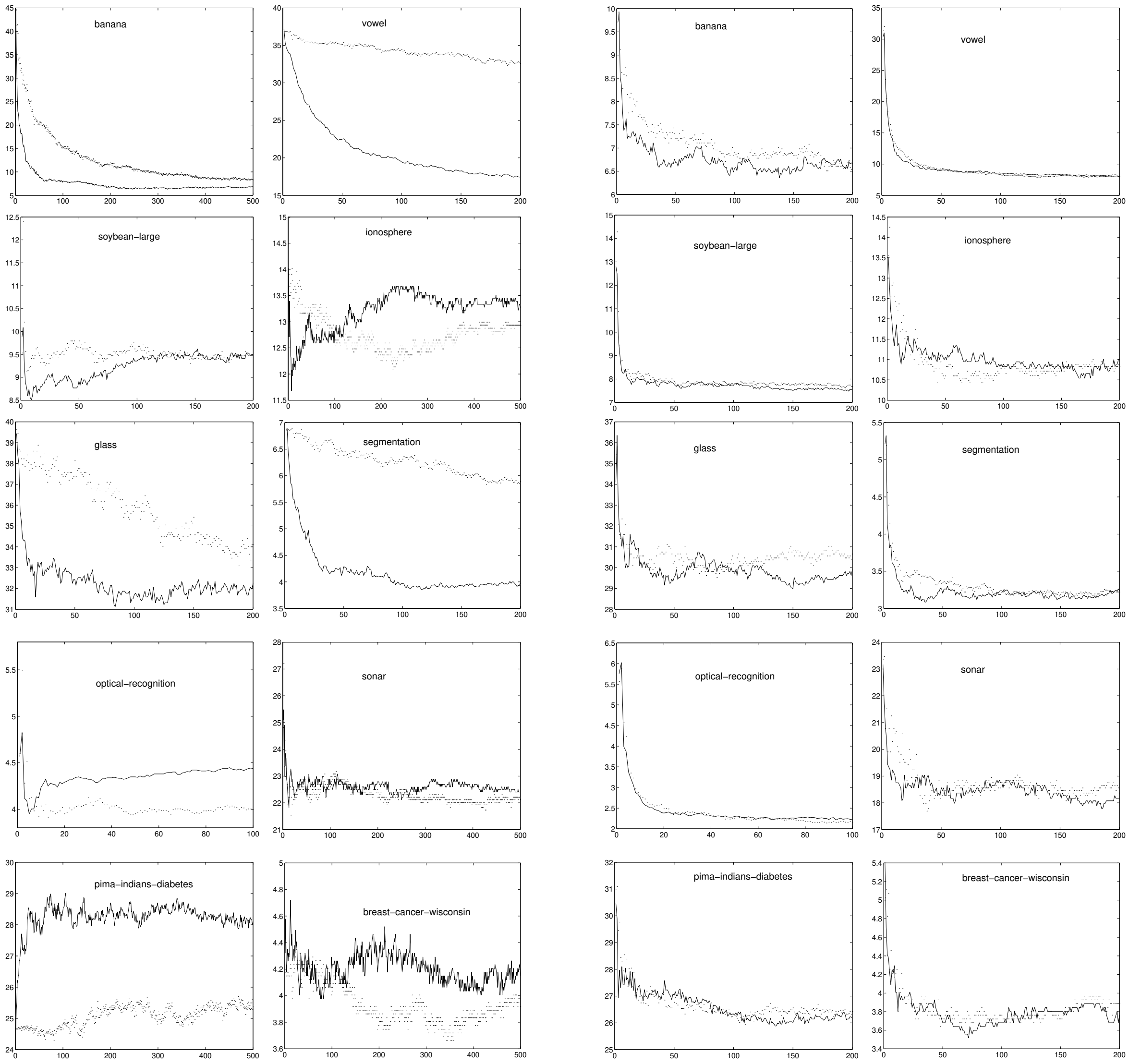

Figure 1: Comparison of AdaBoost (dotted) and DynaBoost (solid) on 10 classification problems: test errors (in percentage of misclassified examples) are displayed as a function of the number of rounds of boosting. The left-hand half gives the results with perceptron base hypotheses and the right-hand half with MLP base hypotheses.