Online Adaptive Policies for Ensemble Classifiers ⋆

Christos Dimitrakakis and Samy Bengio

IDIAP, P.O. Box 592, CH-1920 Martigny, Switzerland

Abstract

Ensemble algorithms can improve the performance of a given learning algorithm through the combination of multiple base classifiers into an ensemble. In this paper we attempt to train and combine the base classifiers using an adaptive policy. This policy is learnt through a Q-learning inspired technique. Its effectiveness for an essentially supervised task is demonstrated by experimental results on several UCI benchmark databases.

Key words: Neural networks, supervised learning, reinforcement learning, ensembles, mixture of experts, boosting, Q-learning

1 Introduction

The problem of pattern classification has been addressed in the past using supervised learning methods. In this context, a set of N example patterns $\hat{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$ is presented to the learning machine, which adapts its parameter vector so that when input vector x_i is presented to it the machine outputs the corresponding class $y_i \in \{1, 2, \ldots, c\}$, where $c \in \mathbb{N}$ is the number of classes. Let us denote the output of a learning machine for a particular vector x_i as $h(x_i)$. The classification error for that particular example can be designated as $l_i = 1$ if $h(x_i) \neq y_i$ and 0 otherwise. Thus, the

⋆ This work is supported by the Swiss National Science Foundation through the National Centre of Competence in Research on "Interactive Multimodal Information Management" and the european project PASCAL. A shorter version of this paper was presented in ESANN 2004.[1]

Email addresses: dimitrak@idiap.ch (Christos Dimitrakakis), bengio@idiap.ch (Samy Bengio).

classification error for the set of examples $\hat{D}$ can be summarised as the empirical error $\hat{L} = \sum_i l_i/N$, which is simply the zero-one loss. If $\hat{D}$ is a sufficiently large representative sample taken from a distribution D, then $\hat{L}$ should be close to the generalization error, $L = \int p_D(x)l(x)$. In practice, however, the training set provides limited sampling of the distribution D, leading to problems such as overfitting. Adding the effects of the classifier's inherent bias and variance, we will have $L > \hat{L}$.

Since the generalization error cannot be directly observed, it has been common to use a part of the training data for validation in order to estimate it. This has led to the development of techniques mainly aimed at reducing the over-fitting caused by limited sampling, such as early stopping and K-fold cross-validation.

Another possible solution is offered by ensemble methods, such as the mixture of experts (MOE) architecture [2], bagging [3] and boosting [4]. The boosting algorithm AdaBoost has been shown to significantly outperform other ensemble techniques. While the good performance of MOE and bagging is related to the independence of experts and the reduction of classifier variance, theoretical results explaining the effectiveness of AdaBoost relate it to the *margin of classification* [5]. See Appendix A for a description of margins.

In this work, which is an extended version of a paper presented at ESANN 2004 [1], the possibility of using an adaptive rather than a fixed policy for training and combining base classifiers is investigated. The field of reinforcement learning (RL) [6] provides natural candidates for use in adaptive policies. In particular, the policy is adapted using Q-learning [7], a method that improves a policy through the iterative approximation of an evaluation function Q. Previously Q-learning had been used in a similar mixture model applied to a control task [8]. An Expectation Maximisation based mixtures of experts (MOE) algorithm for supervised learning was presented in [9]. In this paper, we attempt to solve the same task as in the standard MOE model, but through the use of reinforcement learning rather than expectation maximization techniques. A description of the similarities between reinforcement learning and expectation maximisation methods for multi-expert architectures was presented in [10].

The rest of the paper is organised as follows. The framework of reinforcement learning is introduced in Section 2. Section 2.2 outlines how the RL methods are employed in this work and describes how the system is implemented. Experiments are described in Section 3, followed by conclusions and suggestions for future research.

2 General Architecture

The objective in classification tasks is to reduce the expected value of the error, $E\{l\}$. The empirical loss $\hat{L}$ provides an unbiased estimate of this error in the mean-square sense. The suggested classifier ensemble consists of a set of n base classifiers, or experts, $\mathcal{E} = \{e_1, e_2, ..., e_n\}$ and a controlling agent that selects the experts to make classification decisions and to train on particular examples. The controlling agent must learn to make decisions so that $E\{l\}$ is minimised. We employ reinforcement learning for the purpose of finding an appropriate behaviour for the agent.

The following section will give a brief introduction to the field of reinforcement learning and in particular Q-learning. Subsequent sections detail how Q-learning can be employed in classification tasks and potential problems with the technique are discussed. On the whole, however, it is estimated that reinforcement learning can provide an interesting alternative to supervised learning techniques even for supervised-learning tasks.

2.1 Reinforcement Learning

Reinforcement learning is concerned with the interaction between an agent and the environment it is embedded in. The reinforcement learning problem consists of finding an optimal way for the agent to behave in the environment. It differs from the supervised learning in that there is no direct guidance as to what the agent should learn, i.e. there is no explicit teacher. However some supervision is supplied to the agent in the form of a reward signal that is informative as to how well the agent is performing.

More formally, for the agent we define a set of states $s \in \mathcal{S}$, which can be thought of as corresponding to environmental observations, and a set of actions $a \in \mathcal{A}$ which can be applied by the agent to the environment. At each time step t the agent is at state $s_t = s$ and chooses action $a_t = a$. After the action is taken, the agent receives a reward r_t and it enters a new state $s_{t+1} = s'$, both of which are generated by the environment. A policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ is defined as a set of probabilities:

$$\pi = \left\{ p(a|s) \middle| (s, a) \in \mathcal{S} \times \mathcal{A} \right\}$$

for selecting an action a given the state s. The *objective* is to find the policy that maximises the expected value of the cumulative discounted future reward of the agent, starting at time t. This quantity, otherwise called the *return*, is

defined as:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{1}$$

where $\gamma \in [0, 1)$ is a scalar *discount factor*. Accordingly, the optimal policy can be defined as the policy which maximises the expected value of the return:

$$\pi^* = \arg\max_{\pi} E\{R_t|\pi\}.$$

One can think of γ as a mechanism for weighing the importance of rewards in the distant future relative to immediate rewards. When $\gamma \to 0$, the optimal policy is the policy that maximises the expected value of the immediate reward only. When $\gamma \to 1$, the optimal policy is that which maximises the expected value of all future rewards.

If the expected return of all policies had been known *a priori* we would have been able to trivially select the optimal one. However in general this is not the case and we need to perform the search through the iterative application of two steps. The first step involves estimating the return of the current policy. In the second step we generate an improved policy using our new and improved estimates. The policy evaluation step does not need to be complete, meaning that our evaluation is merely adjusted towards a more accurate value after having had some experience with the current policy π. Indeed, in Q-learning, which is the algorithm that is used in this paper, both the evaluation and the policy updates occur at every time step of experience. Q-learning employs mapping from state-action pairs to expected returns in order to maintain both the evaluation and a representation of the current policy. We briefly explain how this mapping is used in Q-learning below. For more information and an introduction to reinforcement one could turn to the book by Sutton and Barto [6]. A more in-depth look of the algorithm is offered in the book by Bertsekas and Tsitsiklis [11].

The algorithm's first step involves estimating the return of actions under the current policy π. More specifically, we define $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ as the expected return of taking action a when being at state s at time t and following π thereafter:

$$Q_t^\pi(s, a) = E\Big\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \Big| s_t = s, a_t = a, \pi \Big\}.$$

Q_t^π itself is unknown and we maintain instead an estimate Q_t for each state action pair. Herein we employ the Q-learning update when action a_j is selected in state s:

$$Q_{t+1}(s, a_j) = Q_t(s, a_j) + \eta(r + \gamma \max_i Q_t(s', a_i) - Q_t(s, a_j)), \qquad \eta > 0. \tag{2}$$

The second step involves deriving a policy π from the updated estimates Q. This can be derived from the evaluations $Q(s, a)$ either deterministically, by

always selecting the action a_j with the largest estimate of expected return, or stochastically. There are two commonly used stochastic selection mechanisms. The first, ϵ-greedy action selection, selects the highest evaluated action with probability $(1-\epsilon)$, with $\epsilon \in [0,1]$, otherwise it selects a random action. Softmax action selection selects action a_j with probability $e^{Q(s,a_j)} / \sum_i e^{Q(s,a_i)}$. Stochastic action selection is in general necessary so that all state-action pairs are sampled frequently enough to have accurate estimates of the expected return.

2.2 Implementation

We employ an architecture with n experts, implemented as multi-layer perceptrons (MLPs), and a further MLP with n outputs and parameters θ which acts as the controlling agent. All the MLPs have a single hidden layer with hyperbolic tangent units and are trained using steepest gradient descent. The expert MLPs use a softmax output and a cross-entropy criterion, which are suitable for a maximum likelihood classification training. In this setting we attempt to minimise

$$E_{\hat{D}}\{y \log h(x)\}.$$

The state space of the controlling agent is $\mathcal{S} \equiv \mathcal{X}$, the same as the classifiers' input space and its outputs approximate $Q(s, a_j)$. Thus, it is implemented with an MLP which has the same number of inputs as the expert MLPs and with a number of outputs equal to the number of possible actions.

At each time step t a new example x is presented to the ensemble and each expert e_i emits a classification decision $h_i : \mathcal{X} \to \{0,1\}^c$. The ensemble makes a classification decision of the form $f(x) = \sum_i w_i h_i(x)$, with $\sum_i w_i = 1$. We examine the case where the number of actions is equal to the number of experts and in which taking action a_j corresponds to setting $w_i = 1$ for $i = j$ and $w_i = 0$ otherwise. Thus, taking action a_j results in expert e_j making the classification decision. We also chose to use the action a_j to select the expert to be trained on the particular example. As an aside, note that under a given policy, the expected value of w_i given x corresponds to $E\{w_i|x\} = p(a_i|x)$, the probability of action a_i given x. In this manner one could write, for the softmax action selection method,

$$E\{w_i|x\} = p(a_i|x) = \frac{e^{Q(x,a_i)}}{\sum_j e^{Q(x,a_j)}}. \tag{3}$$

The classification decision at time t results in a reward $r_{t+1} \in \{0,1\}$, which is 1 if the example is classified correctly and 0 otherwise. As noted before, we use the gradient form of the Q-learning update (2). The derivative of the cost function with respect to the network outputs is $\delta = r_{t+1} + \gamma \max_i Q(s', a_i) - Q(s, a_j)$.

We use steepest gradient descent with learning rate $\eta > 0$. Note also that when $\gamma = 0$, the Q-learning update is indistinguishable from other state-action value temporal difference updates such as Sarsa(c.f. [6] for a description). The algorithm is implemented as follows:

(1) Select example x_t randomly from $\mathcal{X}$.
(2) Given $s = x_t$, choose $a_j \in \mathcal{A}$ according to a policy derived from Q (for example using ϵ-greedy action selection) .
(3) Take action a_j, observe r_{t+1} and the next state $s' = x_{t+1}$, chosen randomly from $\mathcal{X}$.
(4) $\delta = r_{t+1} + \gamma \max_i Q_t(s', a_i) - Q_t(s, a_j)$.
(5) $\theta_{t+1} = \theta_t + \eta\delta\nabla_\theta Q_t(s, a_j)$.
(6) $s = s'$.
(7) Loop to 2, unless termination condition is met.

2.2.1 Choice of γ

In the algorithm we have described, the state is completely determined by the example x_t. Since this example is selected randomly (steps 1,3), we have $p(s_{t+1} = s'|s_t = s, a_t = a) = p(s_{t+1} = s')$, leading to

$$E\{R_t|s_t = s, a_t = a\} = E\{r_{t+1}|s_t = s, a_t = a\} + \sum_{k=1} \gamma^k E\{r_{t+k+1}|s_t = s, a_t = a\}$$
$$= E\{r_{t+1}|s_t = s, a_t = a\} + \sum_{k=1} \gamma^k \sum_{s'} E\{r_{t+k+1}|s_{t+k} = s'\} p(s_{t+k} = s'|s_t = s, a_t = a)$$
$$= E\{r_{t+1}|s_t = s, a_t = a\} + \sum_{k=1} \gamma^k \sum_{s'} E\{r_{t+k+1}|s_{t+k} = s'\} p(s_{t+k} = s')$$
$$= E\{r_{t+1}|s_t = s, a_t = a\} + E\{r\} \sum_{k=1} \gamma^k$$

where there is an implicit dependency on the policy π. Thus, there is no temporal structure to be exploited by the full reinforcement learning framework, at least in the visible part of the state. In other words, the classification task is similar to an n-armed bandit problem [1] since the next state is not influenced by the agent's actions. For the above reasons, we have set the value of γ to zero. Maximisation of the expected value of equation (1), when $\gamma = 0$ amounts to maximising

$$E\{R_t\} = E\{r_{t+1}\}.$$

[1] In the n-armed bandit problem the objective is to choose an optimal action among n. The reward at each time step only depends upon the action taken and a state s, but the state s does not depend upon the action taken. Thus, the optimal policy is the same no matter what the value of γ is, as the action taken at time t only influences r_{t+1} and not any later rewards.

Since the optimal policy π^* is the policy that maximises this value, we have

$$\pi^* = \arg\max_{\pi} E\{r_{t+1}|\pi\}.$$

Because of our definition of the reward, this is equivalent to finding the policy that minimises the empirical error.

This loss of temporal structure might be considered unfortunate. Indeed, the task is more accurately described as a partially observable process since the parameters of the classifiers constitute a state which changes depending on the agent's actions. This would formally necessitate the need for $\gamma > 0$, and potentially the need to approximate the hidden state with some kind of model. Nevertheless, it seems reasonable to argue that the part of the system state which can be expressed as a function of the classifiers' parameters will change rapidly at the initial stages of learning and then stabilise when each local expert approaches its region of convergence. If this is true, then the problem is similar to a *semi-stationary* bandit problem and a value of $\gamma = 0$ is still appropriate, i.e. there is nothing to be gained by adding temporal structure since old states can never be revisited, at least not with the particular set of actions we have defined.

However there exist some sequence classification applications for which this is not so. These include event detection tasks, such as the detection of the onset of failures in dynamical systems. In particular, if we are defining a state model for the state that defines a joint distribution for actions, observations and state, then the state may no longer be degenerate. This is so in the case where each expert is a hidden Markov model, and where we use the action to switch between models. We, however, concentrate on the simple semi-stationary case, for which interesting parallels with the mixture of experts algorithm can be drawn.

2.2.2 Comparison with Mixture of Experts

The mixture of experts algorithm shares a number of similarities with the one presented here. A comparison between a mixture of experts using a modified version of the EM algorithm and the Q-learning algorithm was presented in [10]. We refrain from introducing new symbols whenever possible in this section, in order to emphasise the relations between algorithms.

In the mixtures of experts framework each expert e_i makes a classification decision $h_i : \mathcal{X} \to \mathbb{R}^c$, with $|h_i|_1 = 1$, where $|\cdot|_1$ denotes the l_1 norm. Thus $h_i(x)$ can be described as probability distribution over the classes given the data x. We use $p(y|x, i)$ to note the probability that expert e_i outputs class y, given x. Similarly, the gating mechanism is used to create a probability distribution over the experts given the data, $p(i|x)$, commonly referred to as

the *prior* of each expert. Thus in order to find the probability of each class given the data we simply use $p(y|x) = \sum_i p(y|x,i)p(i)$.

In order to adjust the parameters of the gating mechanism, both in the gradient and the EM versions of the algorithm, we estimate the corresponding *posterior* as $p(i|x,y) = \frac{p(y|x,i)p(i|x)}{p(y|x)}$. This is the main contrast with the reinforcement learning method we are employing, since the action selection mechanism only considers binary decisions made by the classifiers. Instead of actually calculating $p(i|x,y)$ we are treating the reinforcement r_t as a stochastic variable that depends on the action a_i and for which $E\{r|x,y,\pi,a_i\} = p(i|x,y)$.

3 Experimental Results

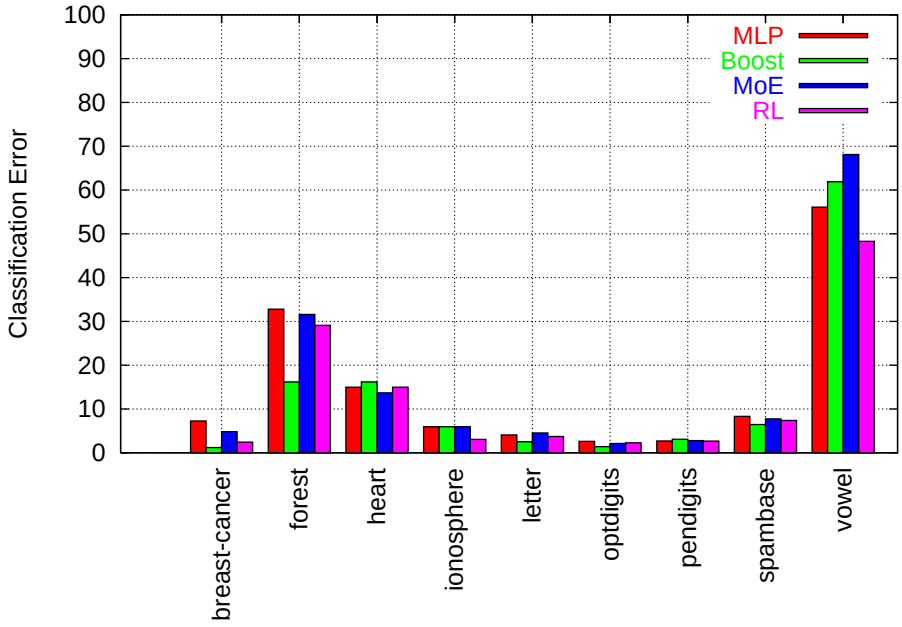

Fig. 1. Test classification error using on 9 UCI benchmark datasets. Results are shown for a single MLP (MLP), and mixtures of 32 experts that have been trained with boosting (Boost), mixture of experts (MoE), and Q-learning (RL).

In order to evaluate the effectiveness of this approach we have performed a set of experiments on 9 datasets that are available from the UCI Machine Learning Repository [12]. For each dataset there was a separate training and test set. We used cross-validation on the training set in order to select the number of hidden units for the base classifier. Each classifier was then trained on the whole training set for 100 iterations and a learning rate $\eta = 0.01$ was used. This was selected by fixing the number of iterations a priori and then choosing the learning rate so that the temporal difference error could converge by the end of 100 iterations. The discount parameter γ for the controlling agent was

set to 0, for the reasons explained in Section 2.2.1. The results reported here are for ϵ-greedy actions selection, with $\epsilon = 0.1$. Results with softmax action selection do not appear significantly different. [2]

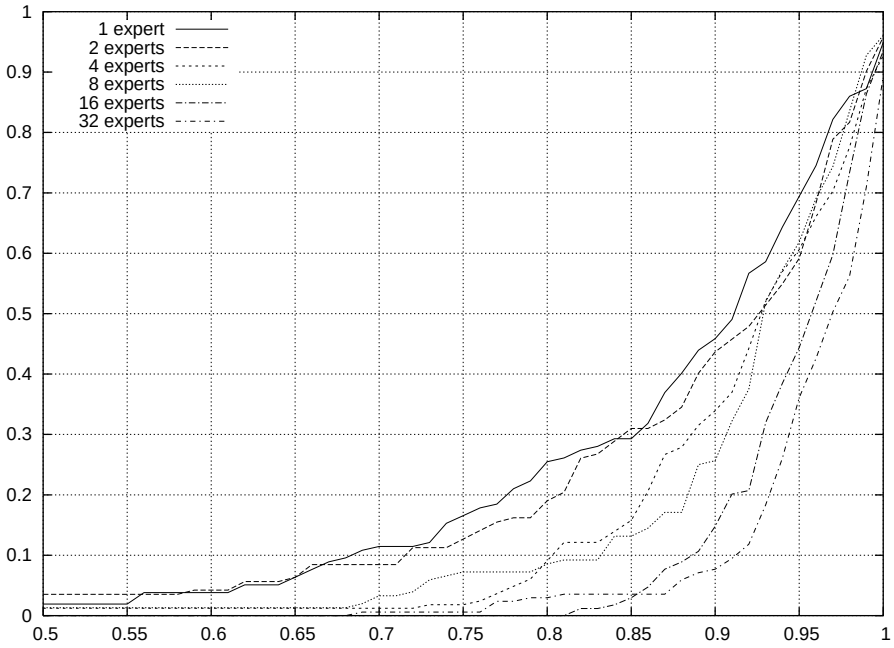

Fig. 2. Cumulative margin distribution for RL on the ionosphere dataset, with an increasing number of experts. See Appendix A for an explanation of margins.

A comparison was made between the RL-controlled mixture, a single MLP, the Mixture of Experts and AdaBoost using MLPs. As Figure 1 summarises, the ensembles generally manage to improve test performance compared to that of the base classifier. The RL mixture outperforms AdaBoost and MOE 4 and 7 times out of 9 respectively. For each dataset we have also calculated the cumulative margin distribution resulting from equation (A.1). For the RL mixture there was a constant improvement in the distribution in most datasets when the number of experts was increased (c.f. Figure 2), though this did not always result in an improvement in generalisation performance.

[2] For this particular problem, and with $\gamma = 0$, the expected return of the best action can be at most 1 while that of the worst action can be at 0. The probability of the greedy action in ϵ-greedy methods, given n actions, is $n + 1/n - \epsilon$. For the softmax method, we would have a similarly flat distribution if all other experts have a similar evaluation, which is to be expected for this particular problem.

4 Conclusions and Future Research

The aim of this work was to demonstrate the feasibility of using adaptive policies to train and combine a set of base classifiers. While this purpose has arguably been reached, there still remain some questions to be answered, such as under what conditions the margin of classification is increased when using this approach.

An interesting aspect of this problem is the state space of the agents. As has been noted in Section 2.2, the initial parameters of the experts constitute a part of the (in our case, unobservable) state space which is only briefly visited by the agent. As learning progresses, the parameters of each expert converge to a steady state. For the case where information about the expert parameters is not included in the state, the problem becomes a slowly changing n-armed bandit task, which in the end becomes stationary. If we include such information in the state, then we are faced with a slightly different reinforcement learning problem than the one commonly encountered. This occurs because there exist a subspace of the state vector (related to the data) which is sampled frequently and another subspace (related to the state of the experts) where only a single trajectory is sampled. The question is firstly what techniques, short of resetting the experts to an initial state, can be applied to sample more trajectories and secondly how can knowledge from more trajectories be used to aid in the search for a better stationary point.

Enlarging the space of actions poses another interesting problem. Suppose for example that the best decision that we can make for a particular input is to combine the outputs of two experts, rather than use a single expert's output. In order to generalise for this case, we define a set of possible weight combinations; each possible combination constitutes a different action. In (3) we defined the expectation of expert weights for a particular input under a softmax policy. In general, however, it is possible to maintain a probability distribution for the weights, rather than a simple expectation. After assuming a joint distribution for the weights we can estimate the conditional density of the return given the weights. Action selection could be done by sampling from the joint distribution of weights, or else importance sampling techniques could be used. This is part of our current work in the field of action selection.

An alternative to action value methods for such enlarged spaces is provided by direct gradient descent in policy space [13]. These have also been theoretically proven to converge in the case of multiple agents and could be much more suitable for problems in partially observable environments and with large state-action spaces.

References

[1] C. Dimitrakakis, S. Bengio, Online policy adaptation for ensemble classifiers, in: 12th European Symposium on Artificial Neural Networks, ESANN 04, 2004.

[2] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, G. E. Hinton, Adaptive mixtures of local experts, Neural Computation 3 (1) (1991) 79–87.

[3] L. Breiman, Bagging predictors, Machine Learning 24 (2) (1996) 123–140.
URL `citeseer.nj.nec.com/breiman96bagging.html`

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

[5] R. E. Schapire, Y. Freund, P. Bartlett, W. S. Lee, Boosting the margin: a new explanation for the effectiveness of voting methods, in: Proc. 14th International Conference on Machine Learning, Morgan Kaufmann, 1997, pp. 322–330.
URL `citeseer.nj.nec.com/schapire97boosting.html`

[6] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.

[7] C. J. Watkins, P. Dayan, Technical note Q-learning, Machine Learning 8 (1992) 279.

[8] C. Anderson, Z. Hong, Reinforcement learning with modular neural networks for control (1994).
URL `citeseer.nj.nec.com/anderson94reinforcement.html`

[9] M. I. Jordan, R. A. Jacobs, Hierarchical mixtures of experts and the EM algorithm, Neural Computation 6 (2) (1994) 181–214.

[10] M. Toussaint, A neural model for multi-expert architectures, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN 2002), 2002.

[11] J. N. T. Dimitri P. Bertsekas, Neuro-Dynamic Programming, Athena Scientific, 1996.

[12] C. Blake, C. Merz, UCI repository of machine learning databases, http://www.ics.uci.edu/∼mlearn/MLRepository.html (1998).

[13] J. Baxter, P. L. Bartlett, Reinforcement learning in POMDP's via direct gradient ascent, in: Proc. 17th International Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA, 2000, pp. 41–48.
URL `citeseer.nj.nec.com/baxter00reinforcement.html`

[14] L. Breiman, Arcing the edge, Tech. rep., Department of Statistics, University of California, Berkeley, CA. (1997).
URL `citeseer.nj.nec.com/breiman97arcing.html`

[15] L. Mason, P. L. Bartlett, J. Baxter, Improved generalization through explicit optimization of margins, Machine Learning 38 (3) (2000) 243.

[16] Y. Li, P. M. Long, The relaxed online maximum margin algorithm, Machine Learning 46 (1/3) (2002) 361.

A Classification Margin

The margin distribution for the two class case can be defined as:

$$\underset{f}{\text{margin}}(x, y) = yf(x),$$

where $x \in \mathcal{X}$, $y \in \{-1, 1\}$ and $f : \mathcal{X} \to [-1, 1]$. In general, the hypothesis $h(x)$ can be derived from $f(x)$ by setting $h(x) = \text{sign}(f(x))$. In this case, $|f(x)|$ can be interpreted as the confidence in the label prediction. For the multi-class case, let $f_y(x)$ be the model's estimate of the probability of class y given input x. In this case the margin is defined as:

$$\underset{f}{\text{margin}}(x, y) = f_y(x) - \max_{y' \neq y} f_{y'}(x). \tag{A.1}$$

Thus the margin can serve as a measure of how far away from the threshold classification decisions are made. A particular measure is the minimum margin over the set $\hat{D}$, i.e.:

$$\underset{\text{min}}{\text{margin}}(\hat{D}) = \min_{(x,y)\in\hat{D}} \text{margin}_f(x, y).$$

It is argued [5] that AdaBoost is indirectly maximising this margin, leading to more robust performance. Although there exist counterexamples for which the minimum margin is not an adequate predictor of generalisation [14], attempts to apply algorithms that directly maximise the margin have obtained some success [15,16].

B About the authors

Christos Dimitrakakis has been a PhD student in machine learning at the IDIAP Research Institue since 2001. He has obtained a BEng in Electronic Systems Engineering in 1997 from the University of Manchester and an MSc in Telecommunication and Information Systems in 1998. Before joining IDIAP, he was working in the multimedia and communications branch of ATMEL, an integrated circuit company.

His main research interest is sequence learning tasks, including sequence classification, reinforcement learning and sequence clustering.

Samy Bengio is a senior researcher in statistical machine learning at IDIAP Research Institute since 1999, where he supervises PhD students and postdoctoral fellows working on many areas of machine learning such as support vector machines, time series prediction, mixture models, large-scale problems, speech recognition, multi-modal (face and voice) person authentication, (asynchronous) sequence processing, brain computer interfaces, text mining, and many more.

He has obtained his PhD in computer science from Université de Montreal (1993), and spent three post-doctoral years at CNET, the research center of France Telecom, and INRS-Telecommunications (Montreal). He then worked as a researcher for CIRANO, an economic and financial academic research center, applying learning algorithms to finance. Before joining IDIAP, he was also research director at Microcell Labs, a private research center in mobile telecommunications. His current interests include all theoretical and applied aspects of learning algorithms.