Scaling Large Learning Problems with Hard Parallel Mixtures

Ronan Collobert[1]*, Yoshua Bengio[1], and Samy Bengio[2]

[1] Université de Montréal, DIRO, CP 6128, Succ. Centre-Ville, Montréal, Canada
{collober, bengioy}@iro.umontreal.ca
[2] IDIAP, CP 592, rue du Simplon 4, 1920 Martigny, Switzerland
bengio@idiap.ch

Abstract. A challenge for *statistical learning* is to deal with large data sets, e.g. in *data mining*. Popular learning algorithms such as *Support Vector Machines* have training time at least *quadratic* in the number of examples: they are hopeless to solve problems with a million examples. We propose a "hard parallelizable mixture" methodology which yields significantly reduced training time through modularization and parallelization: the training data is iteratively partitioned by a "gater" model in such a way that it becomes easy to learn an "expert" model separately in each region of the partition. A probabilistic extension and the use of a set of generative models allows representing the gater so that all pieces of the model are locally trained. For SVMs, time complexity appears empirically to locally grow *linearly* with the number of examples, while *generalization* performance can be enhanced. For the probabilistic version of the algorithm, the iterative algorithm provably goes down in a cost function that is an upper bound on the negative log-likelihood.

1 Introduction

As organizations collect more and more data, the interest in extracting useful information from these data sets with *data mining* algorithms is pushing much research effort toward the challenges that these data sets bring to statistical learning methods. One of these challenges is the sheer size of the data sets: many learning algorithms require training time that grows too fast with respect to the number of training examples. This is for example the case with Support Vector Machines [11] (SVM) and Gaussian processes [12], both being non-parametric learning methods that can be applied to classification, regression, and conditional probability estimation. Both require $O(T^3)$ training time (for T examples) in the worst case or with a poor implementation. Empirical computation time measurements on state-of-the-art SVM implementations show that training time grows much closer to $O(T^2)$ than $O(T^3)$ [2]. It has also been

* Part of this work has been done while Ronan Collobert was at IDIAP, CP 592, rue du Simplon 4, 1920 Martigny, Switzerland.

conjectured [3] that training of Multi-Layer Perceptrons (MLP) might also scale between quadratic and cubic with the number of examples[3]

It would therefore be extremely useful to have general-purpose algorithms which allow to decompose the learning problem in such a way as to drastically reduce the training time, so that it grows closer to $O(T)$.

Another motivation for our work is the availability of cheap parallelism with PC clusters (e.g. Linux clusters). If a decomposition algorithm could separate the work in tasks involving little or rare communication between tasks, then training time could be reduced by one or two orders of magnitude with such loosely-coupled clusters.

The basic idea of this paper is to use an iterative divide-and-conquer strategy to learn a partition of the data such that, ideally (1) the partition is "simple", i.e. it can be learned with good generalization by a classifier with a limited capacity, which we will call the *gater*, and (2) the learning task in each region of the partition is "simple", i.e. it can be learned with good generalization by an *expert* model trained only on the examples of that region. In the end, the prediction on a test point can be obtained by mixing the predictions of the different experts, weighting their predictions with the output of the gater. One therefore obtains a *mixture of experts* [6], but it will not have been trained in the usual ways (maximum likelihood, mean squared error, etc...).

The idea of an SVM mixture is not new, although previous attempts such as Kwok's paper on Support Vector Mixtures [7] trained each SVM on the whole data set. We instead advocate SVM mixtures in which each SVM is trained only on part of the data set, to overcome the time complexity problem for large data sets. We propose here *simple methods* to train such mixtures, and we will show that *in practice* these methods are *much faster* than training only one SVM, and have experimentally lead to results that are *at least as good as one SVM*. We conjecture that the training time complexity of the proposed approach with respect to the number of examples is sub-quadratic for large data sets. Moreover this mixture can be easily parallelized, which could improve again *significantly* the training time.

The organization of the paper goes as follows: in the next section, we briefly introduce the SVM model for classification. In section 4 we present two versions of the hard mixture (a non-probabilistic and a probabilistic one), followed in section 5 by some comparisons to related models. In section 6 we show experimental results on two large real-life data sets. One of the drawbacks of the first version of the algorithm is that it is tied to the mean squared error loss function. Another possible drawback is that the gater must be trained on the whole data set, and this operation could be the bottleneck of the whole procedure. To address these two issues, we present a probabilistic version of the hard mixture model in section 4.2. One advantage of the probabilistic formulation is that it

[3] This is more debatable and may strongly depend on the data distribution. Although we did not formally test this hypothesis, we conjecture that *on very large data sets*, with properly tuned stochastic gradient descent, training time of MLPs is much closer to linear than to quadratic in the number of examples.

generalizes the approach to other tasks (such as conditional probability estimation). The other is that it can eliminate the bottleneck by splitting the task of the gater into multiple local gaters, one per expert. Each of these local gaters is actually a generative model that gives a high score to input vectors that belong to the region of the associated expert model, and this local expert need only be trained with the examples from that region. This probabilistic decomposition is similar to the MOSAIC [4] model but it is used to form a hard partition and not trained by maximum likelihood. We show that the iterative partitioning algorithm actually minimizes an upper bound on the negative log-likelihood (which corresponds to the loss occurring when having to pick a single expert to make the prediction). Experimental results with the probabilistic version of the hard mixture model are presented in section 7.

2 Introduction to Support Vector Machines

Support Vector Machines (SVMs) [11] have been applied to many classification problems, generally yielding good performance compared to other algorithms. For classification tasks, the decision function is of the form

$$y = \text{sign}\left(\sum_{i=1}^{T} y_i \alpha_i K(x, x_i) + b\right) \tag{1}$$

where $x \in \mathbb{R}^d$ is the d-dimensional input vector of a test example, $y \in \{-1, 1\}$ is a class label, x_i is the input vector for the i^{th} training example, y_i is its associated class label, T is the number of training examples, $K(x, x_i)$ is a positive definite kernel function, and $\alpha = \{\alpha_1, \dots, \alpha_T\}$ and b are the parameters of the model. Training an SVM consists in finding α that minimizes the objective function

$$Q(\alpha) = -\sum_{i=1}^{T} \alpha_i + \frac{1}{2}\sum_{i=1}^{T}\sum_{j=1}^{T} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \tag{2}$$

subject to the constraints

$$\sum_{i=1}^{T} \alpha_i y_i = 0 \tag{3}$$

and

$$0 \le \alpha_i \le C \quad \forall i \ . \tag{4}$$

The kernel $K(x, x_i)$ can have different forms, such as the Radial Basis Function (RBF):

$$K(x_i, x_j) = \exp\left(\frac{-\|x_i - x_j\|^2}{\sigma^2}\right) \tag{5}$$

with parameter σ.

Therefore, to train an SVM, one must solve a quadratic optimization problem, where the number of parameters is T. This makes the use of SVMs for large data sets difficult: computing $K(x_i, x_j)$ for every training pair would require $O(T^2)$ computation, and solving may take up to $O(T^3)$. Note however that current state-of-the-art algorithms appear to have training time complexity scaling much closer to $O(T^2)$ than $O(T^3)$ [2].

3 Standard Mixture of Experts

3.1 Probabilistic framework

The idea of *Mixtures of Experts* [5] is simple to explain in a probabilistic framework: given two randoms variables $X \in \mathbb{R}^n$ and $Y \in \mathbb{R}^d$, one would like to represent a *conditional distribution* $P(Y|X)$ as a decomposition of several *simpler* conditional distributions called *experts*. For that, first consider a *discrete* variable E, the identity of an *expert* to be most appropriate for (X, Y). Thus the conditional distribution is rewritten:

$$P(Y|X) = \sum_{i=1}^{N} P(E = i|X)P_i(Y|X)$$

where N is the number of experts and $P_i(Y|X) = P(Y|X, E = i)$ is the conditional distribution for the expert i. The distribution $P(E|X)$ is called the *gater*, because it probabilistically assigns each example to an expert. Usually this kind of mixture is trained using a *log-likelihood maximization* technique, that is, by minimizing $-\sum_{t=1}^{T} logP(y_t|x_t)$ over a training set $D = \{(x_t, y_t)_{t=1..T}\}$.

3.2 Non-Probabilistic Framework

Here, one would like to represent a *function* $y = f(x)$ (instead of the conditional distribution $P(Y|X)$) as a combination of simpler *functions* which are called again "experts". More formally, given a training example $(x, y) \in D$, the following decomposition is built:

$$f(x) = \sum_{i=1}^{N} w_i(x)s_i(x) \tag{6}$$

where $s_i(.)$ is the output function for expert i, and $w(.)$ is the gater, which gives a weight for each expert, given an input x. In general one would like to find the gater $w(.)$ and the experts $s_i(.)$ that minimize the expected value of a loss $L(f, (x, y))$.

The probabilistic and non-probabilistic versions are quite similar, and both could be used in many applications.

4 A New Conditional Mixture

A *standard* mixture of experts represents a *soft decomposition* of the data into subsets, thus both the gater and each expert must be trained on the *whole* data set. Because we want to train complex models on large data sets, we would like instead to take advantage of such a decomposition to *split up the training task into small pieces*. That's the key point of the new models.

The kind of mixture of experts that is presented here could be applied with any kind of expert learner, but, as our first goal was to apply it with SVMs, let us begin with a non-probabilistic framework, where SVMs fit more easily.

4.1 Hard Non-Probabilistic Mixture

The output prediction associated with an input vector x for the hard non-probabilistic mixture that we propose is similar to that in (6) and is computed as follows:

$$f(x) = h\left(\sum_{i=1}^{N} w_i(x) s_i(x)\right) \quad (7)$$

where one just added a transformation of the output with a transfer function h, for example the hyperbolic tangent for classification tasks (which we have found to improve results). In the proposed model, the mixture is trained to minimize the cost function which is the sum of squared losses:

$$C = \sum_{t=1}^{T} \left[f(x_t) - y_t\right]^2 \quad . \quad (8)$$

To train this model, we propose a very simple algorithm:

Algorithm 1 Hard non-probabilistic mixture

1. *Divide* the training set D into N random subsets D_i of size near T/N.
2. Train each expert s_i *separately* over one of these subsets.
3. Keeping the experts fixed, train the gater w to minimize (8) on the whole training set.
4. *Reconstruct* N subsets: for each example (x_t, y_t),
   - sort the experts in descending order according to the values $w_i(x_t)$,
   - assign the example to the first expert in the list which has less than $(T/N + 1)$ examples in order to ensure a balance between the experts.
5. If a termination criterion is not fulfilled (such as a given number of iterations or a validation error going up), go to step 2.

Note that step 2 of this algorithm can be easily implemented in *parallel* as each expert can be trained separately on a different computer. Note also that step 3 can be an approximate minimization (as usually done when training MLPs), that can continue from the solution (parameters) found at the end of the previous outer loop iteration.

The idea of this mixture is intuitively obvious: one *iterates to discover a good partition* of the training set, which ideally could represent in a better way the structure of the training set. As this mixture is non-probabilistic, one can apply it directly to SVMs for experts. In the experiments, we have chosen a MLP for the gater, as for usual non-probabilistic mixture-of-experts.

4.2 Hard Probabilistic Mixture

One possible drawback of the previous model is that the gater must be trained over the whole data set, and this could be the training time bottleneck of the whole procedure. Thus, the second idea that we propose here, is that in a probabilistic context, one can *break up the gater itself* into sub-models, one per expert, that can be trained separately. The idea is similar to that exposed for example in MOSAIC [4]: each expert is associated with a *generative model* $P(X|E=i)$ that can be *trained solely* on the subset D_i. But unlike MOSAIC, the proposed algorithm forms a *hard partition* of the data to train the experts. With this new idea in mind, one can easily adapt the previous algorithm as follows:

Algorithm 2 Hard probabilistic mixture

1. *Divide* the training set into N random subsets D_i of size near T/N.
2. Train each expert $P_i(Y|X)$ *separately* over D_i.
3. Train each local gater $P(X|E=i)$ *separately* over D_i.
4. Estimate the priors $P(E=i)$ by normalizing $|D_i|$, and combine the generative models to obtain the function $P(E=i|X) = \frac{P(X|E=i)P(E=i)}{\sum_{j=1}^{N} P(X|E=j)P(E=j)}$
5. *Reconstruct* N subsets: for each example (x_t, y_t),
   - sort the experts in descending order according to the posterior $P(E=i|x_t, y_t) = \frac{P_i(y_t|x_t)P(E=i|x_t)}{\sum_{j=1}^{N} P_j(y_t|x_t)P(E=j|x_t)}$,
   - assign the example to the first expert in the list which has less than $(T/N+1)$ examples in order to ensure a balance between the experts.
6. If a termination criterion is not fulfilled, go to step 2.

This algorithm is very nice in the sense that it's a *hard version* of the standard mixture of experts model. Unfortunately, standard SVMs don't output probabilities. In the case of a classification problem with several classes, we decided to train one SVM per class (one class against the others) and then to apply a logistic regression on the outputs of the SVMs to obtain probabilities, following [8].

4.3 What Criterion is Minimized?

The above algorithm iteratively modifies parameters θ to go down on a criterion which is an upper bound on the negative joint log-likelihood:

$$J(\theta) = -\max_e J(\theta, e) \tag{9}$$

where

$$J(\theta, e) = \sum_t \sum_i e_{ti} \log P_\theta(y_t, x_t|E=i) P_\theta(E=i) \tag{10}$$

where $e_{ti} \in \{0,1\}$ is a binary variable that selects the i-th expert for example t, with the *selection constraints* $\forall t, \sum_i e_{ti} = 1$ and *balancing constraints* $\forall i, \sum_t e_{ti} \approx T/N$. Note that the joint likelihood for expert i is $P_\theta(y_t, x_t|E=i) = P_\theta(y_t|x_t, E=i)P_\theta(x_t|E=i)$ (i.e. the product of the expert output probability and the local gater likelihood). To relate this to Algorithm 2, note that we are trying to perform the double maximization

$$\max_\theta \max_e \sum_t J(\theta, e)$$

The idea is to perform a "coordinate descent" on $J(\theta, e)$, in which at the first stage of each iteration e is fixed and θ is modified to increase $J(\theta, e)$, and at the second stage θ is fixed and e (the assignment of examples to experts) is modified to increase $J(\theta, e)$. Note that when e is fixed, the above two probabilities (for the expert and local gater) decouple, so they can be maximized separately, as in steps 2 and 3 of the algorithm. In a second stage of each iteration, θ is fixed, and e is modified with step 5 in order to increase $J(\theta, e)$ (here it is an approximate heuristic optimization, to save computations).

Furthermore, the criterion $J(\theta)$ is an upper bound on the joint negative log-likelihood:

$$C(\theta) = -\sum_t \log P_\theta(y_t, x_t) = \sum_t \log(\sum_i P_\theta(y_t, x_t|E=i)P_\theta(E=i)) \tag{11}$$

since, with the constraints on e

$$\begin{aligned}\log(\sum_i P_\theta(y_t, x_t|E=i)P_\theta(E=i)) &\geq \log(\sum_i e_{ti}P_\theta(y_t, x_t|E=i)P_\theta(E=i)) \\ &= \sum_i e_{ti}\log(P_\theta(y_t, x_t|E=i)P_\theta(E=i)) \ .\end{aligned}$$

The idea of minimizing an upper bound on a more desirable cost function is already found in variational learning methods. Note that both cost functions (the negative log-likelihood and $J(\theta)$) will take close values when the gater manages to compute a harder partition.

5 Other Mixtures of SVMs

The idea of mixture models is quite old and has given rise to very popular algorithms, such as the well-known *Mixture of Experts* [6] where the cost function is similar to equation (8) but where the gater and the experts are trained, using gradient descent or EM, on the whole data set (and not subsets) and their parameters are trained simultaneously. Hence such an algorithm is quite demanding in terms of resources when the data set is large, if training time scales like $O(T^p)$ with $p > 1$.

In the more recent *Support Vector Mixture* model [7], the author shows how to replace the experts (typically MLPs) by SVMs and gives a learning algorithm for this model. Once again the resulting mixture is trained jointly on the whole data set, and hence does not solve the quadratic barrier when the data set is large.

In another *divide-and-conquer* approach [9], the authors propose to first divide the training set using an unsupervised algorithm to cluster the data (typically a mixture of Gaussians), then train an expert (such as an SVM) on each subset of the data corresponding to a cluster, and finally recombine the outputs of the experts. Here, the algorithm does indeed train separately the experts on small data sets, like the present algorithm, but there is no notion of an iterative re-assignment of the examples to experts according to the prediction made by the gater of how well each expert performs on each example. Our experiments suggest that this element is essential to the success of the algorithm.

Finally, the *Bayesian Committee Machine* [10] is a technique to partition the data into several subsets, train SVMs or Gaussian Processes on the individual subsets and then use a specific combination scheme based on the covariance of the test data to combine the predictions. This method scales linearly in the number of training data, but is in fact a *transductive* method as it cannot operate on a single test example. Again, this algorithm assigns the examples randomly to the experts (but the Bayesian framework would in principle allow to find better assignments).

6 Experiments: Hard Non-Probabilistic Mixture

In this section are presented two sets of experiments comparing the new non-probabilistic mixtures of SVMs to other machine learning algorithms. Note that all these experiments have been with the *Torch* library.[4] The computers that were used had Athlon 1.2Ghz CPUs.

6.1 A Large-Scale Realistic Problem: Forest

We did a series of experiments on part of the *UCI Forest* data set[5]. We modified the 7-class classification problem into a binary classification problem where the goal was to separate class 2 (the most numerous) from the other 6 classes. Each example was described by 54 input features, each normalized by dividing by the maximum found on the training set. The data set had more than 500,000 examples and this allowed us to prepare a series of experiments as follows:

- A separate test set of 50,000 examples was used compare algorithms.
- A validation set of 10,000 examples was used to select among SVM hyper-parameters, number of experts, of gater hidden units, and gater training epochs.
- Training set size varied from 100,000 to 400,000.
- The hard non-probabilistic mixtures had from 10 to 50 expert SVMs with Gaussian kernel; the MLP gater had between 25 and 500 hidden units.

Since the number of examples was quite large, the same hyper-parameters were selected for all iterations of the algorithm and for all the SVM experts.

We compared our models to

- a single MLP trained with a mean-squared error criterion, and where the number of hidden units was selected on the validation set (from 25 to 250 units),
- a single SVM, where the parameter of the kernel was also selected on the validation set,
- a mixture of SVMs where the gater was replaced by a constant vector, assigning the same weight value to every expert.

[4] available at `http://www.torch.ch`.

[5] The Forest data set is available on the UCI website at the following address: `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/covtype/covtype.info`.

Table 1 gives the results of a first series of experiments with a fixed training set of 100,000 examples. To select among the variants of the hard SVM mixture we considered performance over the validation set as well as training time. All the SVMs used $\sigma = 1.7$. The selected model had 50 experts and a gater with 150 hidden units. A model with 500 hidden units would have given a performance of 8.1% over the test set but would have taken 310 minutes on one machine (and 194 minutes on 50 machines).

Table 1. Comparison of performance between an MLP (100 hidden units), a single SVM, a uniform SVM mixture where the gater always output the same value $1/N$ for each expert, and finally the hard non-probabilistic mixture of SVMs (Algorithm 1).

Model used	Train Error (%)	Test Error (%)	Time (minutes) (1 CPU)	Time (minutes) (50 CPUs)	Iteration
single MLP	17.56	18.15	6		25
single SVM	16.03	16.76	1616		–
uniform SVM mixture	19.69	20.31	43	1	1
hard mixture of SVMs	5.91	9.28	119	37	5

The hard SVM mixture outperformed all models in terms of training and test error. Note that the training error of the single SVM is high because its hyperparameters were selected to minimize error on the validation set (other values could yield to much lower training error but larger test error). It was also much faster, even on one machine, than the single SVM and since the mixture could easily be parallelized (each expert can be trained separately), we also reported the time it took to train on 50 machines. In a first attempt to understand these results, one can at least say that the power of the model does not lie only in the MLP gater, since a single MLP was pretty bad, it is neither only because we used SVMs, since a single SVM was not as good as the hard mixture, and it was not only because we divided the problem into many sub-problems since the uniform mixture also performed badly. It seems to be a combination of all these elements.

In order to find how the algorithm scaled with respect to the number of examples, we then compared the same mixture of experts (50 experts, 150 hidden units in the gater) on different training set sizes. Figure 1 shows the validation error of the mixture of SVMs with training set sizes from 100,000 to 400,000. It seems that, *at least in this range and for this particular data set, the mixture of SVMs scales linearly with respect to the number of examples,* and not quadratically as a classical SVM. It is interesting to see for instance that the mixture of SVMs was able to solve a problem of 400,000 examples in less than 4 hours (on 50 computers) while it would have taken more than one month to solve the same problem with a single SVM.

Finally, figure 2 shows the evolution of the training and validation errors of a hard mixture of 50 SVMs gated by an MLP with 150 hidden units, during 5 iterations of the algorithm. This should convince that the iterative partitionning is essential in order to obtain good performance. It is also clear that the empirical convergence of the outer loop is extremely rapid.

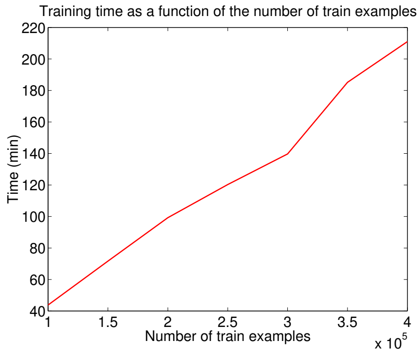

Fig. 1. Comparison of the training time of the same mixture of SVMs (50 experts, 150 hidden units in the gater) trained on different training set sizes, from 100,000 to 400,000.

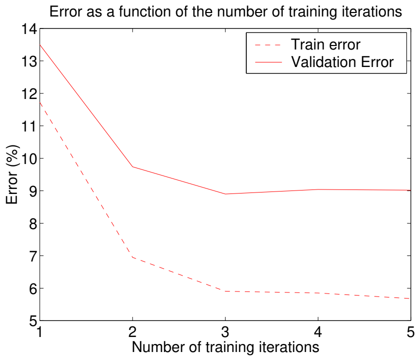

Fig. 2. Comparison of the training and validation errors of the mixture of SVMs as a function of the number of training iterations

6.2 Verification on Another Large-Scale Problem

To verify that the results obtained on *Forest* were replicable on other large-scale problems, we tested the SVM mixture on a speech task, the *Numbers95* data set [1], turned it into a binary classification problem (separate silence frames from non-silence frames, from a total of 540,000 frames). The training set contains 100,000 randomly chosen frames out of the first 400,000 frames. The disjoint validation set contains 10,000 randomly chosen frames out of the first 400,000. The test set contains 50,000 randomly chosen frames out of the last 140,000. The validation set was used to select the number of experts, the number of hidden units in the gater, and σ. Each frame was parameterized using standard methods (j-rasta coefficients, with first and second temporal derivatives) yielding 45 coefficients times 3 frames (= 135 inputs).

Table 2 shows a comparison between a single SVM and a non-probabilistic hard mixture of SVMs, with 50 experts, 50 hidden units in the gater, and $\sigma = 3$. The mixture of SVMs was again many times faster than the single SVM (even on a single CPU) but yielded similar generalization performance.

Table 2. Comparison of performance between a single SVM and a mixture of SVMs on the speech data set

Model used	Train Error (%)	Test	Time (minutes) (1 CPU)	(50 CPUs)
one SVM	0.98	7.57	3395	
hard non-prob. mixture of SVMs	4.41	7.32	426	33

7 Experiments: Hard Probabilistic Mixture

The second set of experiments concerns the probabilistic version of the algorithm. As standard SVMs don't output probabilities, we first present in the first subsection results with Multi-Layered-Perceptrons (MLP) as experts, to confirm that the approach works well with gradient-based learning algorithms. Then we present some results with SVMs as experts, with the SVM outputs being fed to a logistic regressor in order to obtain conditional probabilities, as in [8].

7.1 MLP Experts

The experiments described here are again with the *Forest* data set described earlier. The setup is the same as previously for the non-probabilistic mixture. Thus, we just have to specify the probabilistic model architecture: we used *Gaussians mixtures* for the generative models and *one-hidden-layer MLP* trained with a *log-likelihood maximization criterion* for the experts (i.e. maximizing $\sum_t \log P_\theta(y_t|x_t, E = i)$ where the output of the MLP has softmax units which sum to 1 to represent these probabilities). We compared the hard probabilistic mixture with a standard (not hard) probabilistic mixture (with MLPs as experts and an MLP as gater), trained by stochastic gradient ascent on the log-likelihood $\sum_t \log P_\theta(y_t|x_t)$. We also compared with a single MLP (also trained to maximize the log-likelihood). Note that with this training criterion, the single MLP gives better results than those obtained with a mean-squared error criterion (which was used in the experiments previously reported in section 6.1).

The results are summarized in Table 3. For MLPs and standard mixtures, the *iteration* column indicates the number of training epochs, whereas for hard mixtures it is the number of outer loop iterations. Note that for hard mixtures, the number of inner loop epochs to train MLP experts was fixed to a maximum of 100 (This number was chosen according to the validation set. Moreover, training was stopped earlier if training error did not decrease significantly.)

The hard probabilistic mixture appears to work very well. On this data set, we can obtain better generalization than an MLP, in a reasonable time if we use sequential training (on only one computer), and *impressively* short time if we use parallelization. If we take the time to do more iterations, the generalization can be impressive too, as shown on a training set size of 400,000 examples. Figure 3 shows the importance of the iterative process of our model for the training as well as generalization error, as previously shown for the non-probabilistic model.

We did one more experiment to compare the hard *non*-probabilistic and probabilistic mixtures, in terms of training time. The experiment is performed with 100,000 training examples (obtaining similar generalization results in both cases). The hard probabilistic mixture has 20 experts, 25 hidden units per expert and 20 Gaussians. The hard non-probabilistic mixture has 20 experts, and an MLP gater with 150 hidden units. The hard non-probabilistic mixture took *more than 30 minutes to train*, whereas the hard probabilistic mixture took *only* 1.3 *minutes*! It seems that *the training time bottleneck due to the gater has been broken with the hard probabilistic mixture.*

Table 3. Comparison of performance on the *Forest* data set between one MLP, a standard mixture, and the hard probabilistic mixture proposed in this paper

Model used	Error (%)			Time (minutes)		Iteration
	Train	Valid	Test	one CPU	parallel	
100,000 training examples						
single MLP (500 hidden units)	9.50	11.09	10.96	121	–	150
standard mixture (10 experts, 50 hidden units per expert and 150 units for the gater)	10.57	11.05	11.56	124	–	65
	7.01	9.30	9.10	290	–	150
hard prob. mixture (20 experts, 25 hidden units per expert and 20 Gaussians per $P(X\|E=i)$)	7.89	10.76	10.90	21	**1.3**	15
400,000 training examples						
single MLP (500 hidden units)	8.39	8.38	8.69	461	–	150
hard prob. mixture (40 experts, 25 units per expert and 10 Gaussians per $P(X\|E=i)$)	6.90	7.74	8.09	126	**3.6**	7
	4.63	5.64	6.24	344	10	20
hard prob. mixture (40 experts, 50 units per expert and 10 Gaussians per $P(X\|E=i)$)	6.68	7.54	8.05	195	5.3	6
	3.37	**5.60**	**5.61**	624	17	20

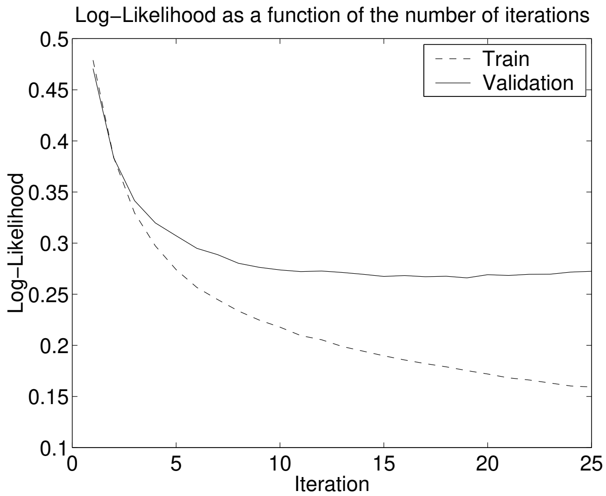

Fig. 3. Evolution of the log-likelihood with the number of iteration for the hard probabilistic mixture, on 100,000 training examples. The mixture had 20 experts (25 hidden units, 20 Gaussians)

7.2 Dimensionality Reduction for the Gaussian Mixture Models

We used Gaussians Mixture Models (GMM) to estimate $P(X|E=i)$ in the hard probabilistic mixture, and one might think that *GMM don't work well with high* *dimensional data*. Thus, we compare results obtained with and without reducing the dimensionality of the GMM observations, as a preprocessing before applying the mixture.

To reduce the dimensionality, we trained as a classifier (with conditional maximum likelihood) an MLP with a *small tanh hidden layer* and softmax outputs. The hidden layer learns a transformation that has low dimension and is useful to predict the output classes. A *single training epoch* is performed, on only a part of the training set if this one is very large (100,000 examples was sufficient on *Forest* in any case). This is *quick*, and surprisingly, sufficient to obtain good results. Finally, the hidden layer outputs of the MLP (for each input vector x_t) are given as observations for the GMM.

As shown in Table 4, it appears that the dimensionality reduction improves the generalization error, *as well as the training error*. The dimensionality reduction reduces capacity, but we suspect that the GMMs are so poor in high dimensional spaces that the dimensionality reduction improves results even on the training set, by making it easier to carve the input space in ways that lead to easy training of the experts.

Table 4. The effect of dimensionality reduction for GMMs in the hard probabilistic mixture, on 400,000 examples with 40 experts, 50 hidden units for experts and 10 Gaussians for each $P(X|E=i)$

Model used	Error (%)		
	Train	Valid	Test
Without Dim. Reduction	**4.45**	5.95	6.25
With Dim. Reduction	**3.37**	5.60	5.61

7.3 SVM Experts

Similar experiments were performed on the Forest database with the hard probabilistic mixture, but using SVMs plus logistic as probabilistic experts, rather than MLPs.

Table 5 shows the results obtained on the 100,000 examples training set, with different numbers of experts and different choices of gaters. The first experiment uses the methodology already introduced and used with MLP experts, but with 20 SVM experts. Note that training time is much larger than with MLP experts (Table 3, 1.3 min. in parallel), and much larger than with the hard non-probabilistic mixture (Table 1, 37 min. in parallel). One explanation is that convergence is much slower, but we do not understand why.

One clue is that when replacing the GMMs by a single MLP gater,[6] (with the two other experiments in Table 5), much faster convergence is obtained (down to 21 min. in parallel, i.e. faster than the hard non-probabilistic mixture), but still slower than with MLP experts.

[6] Here, the number of inner loop epochs for training the gater was chosen using the validation set, and fixed to 3.

Table 5. Comparison of performance of the hard *probabilistic* mixture, for several setups, on the *Forest* data set with 100,000 training examples

Model used	Error (%)			Time (minutes)		Iteration
	Train	Valid	Test	one CPU	parallel	
20 SVM experts and 10 Gaussians per $P(X\|E=i)$	5.39	10.93	10.70	2240	157	16
20 SVM experts and a MLP gater with 150 hidden units	2.63	8.86	8.93	291	30	9
50 SVM experts and a MLP gater with 150 hidden units	3.22	8.92	9.15	118	21	9

8 A Note on Training Complexity

For both the probabilistic and non-probabilistic mixtures, suppose that we choose the number of experts N such that the number of examples per expert $M = T/N$ is a *fixed* fraction of the total number of examples. Then if we suppose that the training time for one expert is polynomial of order p with the number of examples T, then the training time for training the experts in one outer-loop iteration of the hard mixtures is:

$$NM^p = TM^{p-1} = \mathbf{O}(\mathbf{T}) \ .$$

If the gater is not localized (e.g. as in the hard non-probabilistic mixture when using a single model as gater, and in the hard probabilistic mixture), then it may be a bottleneck of the algorithm. In the case of the non-probabilistic mixture, we don't know exactly the cost of training the gater. As it's a MLP, it's probably more than $O(T)$. But *for the probabilistic mixture*, it appears empirically that $O(T)$ *training time is sufficient for the gater, at each iteration of Algorithm 2*!

9 Conclusion

In this paper we have presented a new divide-and-conquer parallelizable hard mixture algorithms to reduce the training time of algorithms such as SVMs. Very good results were obtained compared to classical SVMs either in terms of training time or generalization performance on two large scale difficult databases. Moreover, the algorithms appears to scale linearly with the number of examples, at least between 100,000 and 400,000 examples. Both a probabilistic and a non-probabilistic version of the algorithm were presented, with a demonstration that the probabilistic version actually minimizes a well-defined criterion (that corresponds to the error made by a single chosen expert of the mixture).

These results are extremely encouraging and suggest that the proposed method could allow training SVM-like models for very large multi-million data sets in a reasonable time. Two types of "gater" models were proposed, one based on a single MLP, and one based on local Gaussian Mixture Models. The latter have the advantage of being trained very quickly and locally to each expert, thereby guaranteeing linear training time for the whole system (per iteration). However, the best results (even in training time) are often obtained with the MLP gater (which needs few epochs and yields in less iterations to a good partition). Surprisingly, even faster results (with as good generalization) are obtained if the SVM experts are altogether replaced by MLP experts. If training of the MLP gater with stochastic gradient takes time that grows much less than quadratically, as we conjecture it to be the case for very large data sets (to reach a "good enough" solution), then the whole method is clearly sub-quadratic in training time with respect to the number of training examples.

Acknowledgments

The authors thank Léon Bottou for stimulating discussions. RC would like to thank the Swiss NSF for financial support (project FN2100-061234.00). YB would like to thank NSERC, MITACS and IRIS for funding and support.

References

1. R.A. Cole, M. Noel, T. Lander, and T. Durham. New telephone speech corpora at CSLU. *Proceedings of the European Conference on Speech Communication and Technology, EUROSPEECH*, 1:821–824, 1995.
2. R. Collobert and S. Bengio. SVMTorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1:143–160, 2001.
3. S.E. Fahlman. Fast-learning variations on back-propagation: An empirical study. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 38–51, Pittsburg 1988, 1989. Morgan Kaufmann, San Mateo.
4. M. Haruno, DM. Wolpert, and M. Kawato. Mosaic model for sensorimotor learning and control. *Neural Computation*, 13(10):2201–2220, 2001.
5. R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixture of local experts. *Neural Computation*, 3:79–87, 1991.
6. Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.
7. J. T. Kwok. Support vector mixture for classification and regression problems. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, pages 255–258, Brisbane, Queensland, Australia, 1998.
8. J. Platt. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In Smola, Bartlett, Schölkopf, and Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 61–73. MIT Press, 1999.
9. A. Rida, A. Labbi, and C. Pellegrini. Local experts combination through density decomposition. In *Proceedings of UAI'99*. Morgan Kaufmann, 1999.
10. V. Tresp. A bayesian committee machine. *Neural Comp.*, 12:2719–2741, 2000.
11. V. N. Vapnik. *The nature of statistical learning theory*. Springer, 2nd edition, 1995.
12. C.K.I Williams and C.E. Rasmussen. Gaussian processes for regression. In D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 514–520. MIT Press, 1996.