THÈSE DE DOCTORAT DE L'UNIVERSITÉ PARIS VI

Spécialité
Informatique

Présentée par
Ronan Collobert

Pour obtenir le grade de
DOCTEUR de l'UNIVERSITÉ PARIS VI

Large Scale Machine Learning

Soutenue le
28 juin 2004

Devant le jury composé de

Directeur de thèse :	Patrick Gallinari
Rapporteurs :	Léon Bottou
	Hélène Paugam-Moisy
Examinateurs :	Samy Bengio
	Yves Grandvalet

Résumé

Cette thèse aborde de façon générale les algorithmes d'apprentissage, avec un intérêt tout particulier pour les grandes bases de données. Après avoir formulé le problème de l'apprentissage de manière mathématique, nous présentons plusieurs algorithmes d'apprentissage importants, en particulier les *Multi Layer Perceptrons*, les *Mixture d'Experts* ainsi que les *Support Vector Machines*. Nous considérons ensuite une méthode d'entraînement pour les *Support Vector Machines*, adaptée aux ensembles de données de tailles raisonnables. Cependant, l'entraînement d'un tel modèle reste irréalisable sur de très grande bases de données. Inspirés par la stratégie "diviser pour régner", nous proposons alors un modèle de la famille des *Mixture d'Experts*, permettant de séparer le problème d'apprentissage en sous-problèmes plus simples, tout en gardant de bonnes performances en généralisation. Malgré de très bonnes performances en pratique, cet algorithme n'en reste pas moins difficile à utiliser, à cause de son nombre important d'hyper-paramètres. Pour cette raison, nous préférons nous intéresser ensuite à l'amélioration de l'entraînement des *Multi Layer Perceptrons*, bien plus faciles à utiliser, et plus adaptés aux grandes bases de données que les *Support Vector Machines*. Enfin, nous montrons que l'idée de la marge qui fait la force des *Support Vector Machines* peut être appliquée à une certaine classe de *Multi Layer Perceptrons*, ce qui nous mène à un algorithme très rapide et ayant de très bonnes performances en généralisation.

Mots-Clefs : Algorithmes d'Apprentissage, Grandes Bases de Données, *Multi Layer Perceptrons*, *Mixture d'Experts*, *Support Vector Machines*, Diviser pour Régner, Gradient Stochastique, Optimisation, Marge.

Summary

This thesis aims to address machine learning in general, with a particular focus on large models and large databases. After introducing the learning problem in a formal way, we first review several important machine learning algorithms, particularly Multi Layer Perceptrons, Mixture of Experts and Support Vector Machines. We then present a training method for Support Vector Machines, adapted to reasonably large datasets. However the training of such a model is still intractable on very large databases. We thus propose a divide and conquer approach based on a kind of Mixture of Experts in order to break up the training problem into small pieces, while keeping good generalization performance. This mixture model can be applied to any kind of existing machine learning algorithm. Even though it performs well in practice the major drawback of this algorithm is the number of hyper-parameters to tune, which makes it difficult to use. We thus prefer afterward to focus on training improvements for Multi Layer Perceptrons, which are easier to tune, and more suitable than Support Vector Machines for large databases. We finally show that the margin idea introduced with Support Vector Machines can be applied to a certain class of Multi Layer Perceptrons, which leads to a fast algorithm with powerful generalization performance.

Keywords: Machine Learning, Large Databases, Multi Layer Perceptrons, Mixture of Experts, Support Vector Machines, Divide and Conquer, Stochastic Gradient Descent, Optimization, Margin.

Acknowledgements

I am indebted to several people who, over the course of my studies, I have come to rely upon for help and guidance. Without these people, this thesis would not have been the same.

I would like to thank Samy Bengio who first gave me the opportunity to do my research project, at IDIAP, in Switzerland. I must say it was great to work with him. He is the kind of rare supervisor who is always available, knows his domain, makes constructive remarks, and is also fun. Working with him was really motivating. Also, I would like to thank IDIAP for funding my research.

My thesis could be compared to a random walk process. I went twice to the University of Montréal, to work with Yoshua Bengio. Yoshua is Samy's brother – almost his twin. When I arrived in Montréal, I remember he said to me, "All you saw in machine learning until now is just the visible part of an iceberg." He was right. Many thanks to him for helping me to see the rest of this pretty impressive iceberg.

I would also like to thank Patrick Gallinari, for accepting me as a PhD candidate. Thanks also go to Léon Bottou, Yves Grandvalet and Hélène Paugam-Moisy for being on my jury and for their interest in my work.

Also, special thanks to Léon Bottou. Without some discussions we had at the yearly Snowbird conference, I would have missed some ideas that are developed in this thesis.

This thesis has a history. It started when I was 16 and I coded my first neural network. Since my father works in machine learning and astrophysics and my mother is a physicist, it seems I was bound to do science. My sister chose astrophysics, I chose machine learning. Thanks to all of them for all they gave me.

I would never have survived in Montréal without many friends. I would like to address my best thoughts to them, especially to Charles Dugas, Francis Pieraut, Myriam St-Jean and Pascal Vincent with whom I shared great moments.

Life in Switzerland was a paradise thanks to all my friends who were always there for me when I needed help with my work and with my life. Thanks to all of them, and especially Johnny Mariéthoz who shared more than the same office with me for three years. Thanks to Norbert Crettol and Anne-Danièle Wanner who showed me deep and beautiful parts of Switzerland. Also, thanks to my roommates Michael McGreevy, Pedro Quelhas and Kevin Smith who supported me in many ways in the last year (and minutes!) of my thesis.

Contents

1 Introduction

Under its somewhat pompous title "Large Scale Machine Learning", this thesis aims to address machine learning in general, with a particular focus on large models and large databases. An alien, a normal human or more simply a reader not aware of this field, would have already stopped and asked...

What is machine learning?
Machine learning is the study of algorithms which can learn. According to our favorite dictionary, an algorithm is "a precise rule or set of rules specifying how to solve some problem". Learning is "a behavioral modification especially through experience or conditioning". Note the important word *experience*. We are interested here in algorithms which can modify their behavior according to some *experience*. As algorithms are generally processed by a computer, and as we will define later learning in a mathematical way, machine learning is in the end a combination of mathematics and computer science.

What is a model?
The set of rules which defines an algorithm is fixed, but has some modifiable parameters. Changing these parameters changes the behavior of the algorithm. The set of these parameters defines what we call a model.

What is a database?
A database is the experience that we give to an algorithm. A database also represents a task to achieve. We show this task to the algorithm, and hopefully the algorithm changes its behavior in order to be good at this task. Then, we say that the algorithm learns if it is able to perform well at a similar task *that it has never seen before*. That is, it is able to *generalize*. Of course, if the task

is complex, then the algorithm will be complex as well. In other words, the larger the database, the larger the model defined by the algorithm will be.

And anyway, why this thesis?
Machine learning techniques are relevant to several areas such as data mining, computer vision and speech recognition, which are dealing with an increasing amount of data. It can be billions of lines of text, Terabytes of images or hours of speech. Machine learning techniques need to be adapted and improved for these applications. A mathematician who proposes a new algorithm is in general not interested in knowing if it is *tractable on large databases*. An engineer will try in general to patch an *existing* machine learning technique to suit his needs. We do not pretend to solve every large practical problem here. But we would like to address the problem of *learning on large databases* in a *general way* (it will not be specific to text, images or speech) always keeping in mind that what we really want is to *generalize*. We would like to *improve* and *extend existing machine learning techniques* in this way.

Fundamentals

This thesis aims to be self-explanatory, supposing that the reader has a good knowledge of mathematics and in particular probability theory. Thus, in the first chapters we define formally our framework. Chapter 2 defines the concept of learning in a mathematical way as well as fundamental concepts used in machine learning. It also gives an overview of the theory which explains why machine learning works. Chapter 3 defines several classical and important algorithms used in this thesis, and in particular Multi Layer Perceptrons, Mixture of Experts, and Support Vector Machines. Chapter 4 gives the experimental framework used to illustrate our following theoretical developments.

Contributions

The next chapters are dedicated to the improvement and extension of previously introduced machine learning algorithms. Each chapter corresponds to one or several reviewed and published papers, that we rewrote here in a unified manner. We also improved them, extended them, and tried to give a better understanding and analysis of our theoretical assertions, taking advantage of our global view and experience gained while carrying out this thesis. The chapters appear in chronological order, preserving the ideas which drove our research.

Optimization for Support Vector Machines

First, in Chapter 5 we present a training method for Support Vector Machines, adapted to reasonably large databases. We had to start with this training method, because Support Vector Machines are a recent important machine learning algorithm, but which was not very suitable for large databases at the beginning of its history. Even with the state-of-the-art algorithm we propose, Support Vector Machines are still intractable for very large scale databases.

Divide and Conquer

We thus propose a "divide and conquer" approach based on a kind of Mixture of Experts in Chapter 6, in order to try to break up the training problem into small pieces, while keeping the good generalization properties of Support Vector Machines. We also apply this divide and conquer technique on Multi Layer Perceptrons, which appears to work fairly well. Unfortunately, Mixture of Experts have plenty of "free" parameters which have to be set by their user. In practice a lot of trial-and-error steps are needed to obtain the perfect Mixture. Even if the "training" is fast, the "tuning" takes time.

Optimization for Multi Layer Perceptrons

We focus on Multi Layer Perceptrons in Chapter 7, which are much easier to tune and which also appeared more suitable for large databases than Support Vector Machines. In particular, we propose to study some training problems raised in the previous chapter, with the hope to find methods for faster Multi Layer Perceptron training on large databases.

Margin for Everyone

Finally, we wanted to perform a deep comparison between Support Vector Machines and Multi Layer Perceptrons, in order to answer the question "is it possible to modify the old Multi Layer Perceptron algorithm with some improvements for better generalization supplied by the new Support Vector Machine algorithm?". We answer this question in Chapter 8.

We give a general conclusion and personal thoughts in Chapter 9, as well as possible future research directions.

2 Statistical Learning Theory

Learning can be defined as *a modification of a behavioral tendency by experience.* The *machine learning* field aims to study ways to make machines learn. In fact, Turing (1950) who first introduced the concept of a "learning machine", pointed out that one can separate the hardware and programming aspects of the machine. Following this idea, *machine learning* focuses on *algorithms* which can learn, as defined in Definition 2.1, and disregards the hardware only necessary to implement the algorithms. It is also important to notice that our interest is not "learning by heart", but "learning to generalize".

Definition 2.1 *An algorithm is said to learn from experience with respect to some class of tasks, if its performance on this class of tasks increases with experience, given a measure of performance.*

In this chapter we will introduce the notion of a machine learning algorithm in a statistical mathematical framework. We first define the learning process as a process of choosing an appropriate function from a given set of functions. We then present the statistical learning theory introduced by Vapnik (1995) which is the foundation of modern machine learning.

Learning With a Finite Number of Examples

We first consider a distribution represented by a density $P : \mathcal{Z} \to \mathbb{R}$ (where $\mathcal{Z}$ is a real vector space), *that is unknown* and which represents the "class of tasks" to learn, as suggested in Definition 2.1. We then consider L observations

REF A. M. Turing. Computing machinery and intelligence. *Mind: A Quarterly Review of Psychology and Philosophy*, 59(236):433–461, 1950.

REF V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, second edition, 1995.

(the *training examples*)

$$\boldsymbol{z}_1,\, \boldsymbol{z}_2, \ldots,\, \boldsymbol{z}_L \tag{2.1}$$

drawn independently and identically from this distribution. These observations represent the "experience" that we will give to the learning algorithm, and is usually called the *training set*, or the *empirical data*. We then choose a set of functions $\mathcal{F}$ and we define a loss function (the "measure of performance")

$$Q : \mathcal{Z} \times \mathcal{F} \to \mathbb{R}\,. \tag{2.2}$$

With this notation, the learning process can be described as the search of a function f in $\mathcal{F}$ which minimizes the cost function Q over $\mathcal{Z}$. In other words, a machine learning algorithm is an algorithm which finds a way to minimizes the *expected risk* (or *generalization error*)

$$R : f \in \mathcal{F} \longmapsto \mathbf{E}(Q(\boldsymbol{z},\, f)) = \int_{\mathcal{Z}} Q(\boldsymbol{z},\, f)\, P(\boldsymbol{z})\, d\,\boldsymbol{z}\,, \tag{2.3}$$

with the knowledge of the training set (2.1) only.

In practice, we usually distinguish three main machine learning tasks; first of all, if $f \in \mathcal{F}$ is of the form $f : \mathcal{X} \to \mathcal{Y}$, and if the training examples can be rewritten as $\boldsymbol{z}_l = (\boldsymbol{x}_l,\, \boldsymbol{y}_l) \in \mathcal{X} \times \mathcal{Y}$, we have the following categories:

- *classification* : $\mathcal{Y}$ is a finite set, and $\boldsymbol{y} \in \mathcal{Y}$ represents a category (a class). The aim is to find a function $f \in \mathcal{F}$ which for all examples $(\boldsymbol{x},\, \boldsymbol{y}) \in \mathcal{X}\times\mathcal{Y}$ assigns input vector $\boldsymbol{x}$ to its corresponding class $\boldsymbol{y}$. Thus, the loss function could be

$$Q(\boldsymbol{z},\, f) = \begin{cases} 0 & \text{if } f(\boldsymbol{x}) = \boldsymbol{y} \\ 1 & \text{otherwise.} \end{cases}$$

- *regression*: in this case, $\mathcal{Y}$ is a real vector space, and we would like to find the function $f \in \mathcal{F}$ such that $f(\boldsymbol{x})$ is as close as possible to $\boldsymbol{y}$, with respect to an arbitrarily chosen distance, for all examples $(\boldsymbol{x},\, \boldsymbol{y}) \in \mathcal{X}\times\mathcal{Y}$. If considering the Mean Squared Error (MSE) distance, the loss function appears to be

$$Q(\boldsymbol{z},\, f) = \|f(\boldsymbol{x}) - \boldsymbol{y}\|^2\,.$$

  In that case, a minimum of the expected risk is then reached for the expectation of $\boldsymbol{y}$ knowing $\boldsymbol{x}$, that is for the function $f^\star(\boldsymbol{x}) = \mathbf{E}(\boldsymbol{y}|\boldsymbol{x})$. Indeed, for all functions f we can derive

$$\begin{aligned} R(f^\star) &= \mathbf{E}[\, \|\mathbf{E}(\boldsymbol{y}|\boldsymbol{x}) - \boldsymbol{y}\|^2\,] \\ &= \mathbf{E}[\, \|f(\boldsymbol{x}) - \boldsymbol{y}\|^2\,] + \mathbf{E}[\, \|\mathbf{E}(\boldsymbol{y}|\boldsymbol{x}) - f(\boldsymbol{x})\|^2\,] \\ &\quad + 2\mathbf{E}[\, (\mathbf{E}(\boldsymbol{y}|\boldsymbol{x}) - f(\boldsymbol{x})) \cdot (f(\boldsymbol{x}) - \boldsymbol{y})\,]\,. \end{aligned}$$

Using the classical equality $\mathbf{E}(\cdot) = \mathbf{E}(\mathbf{E}(\cdot|\boldsymbol{x}))$ in the last term of the equation leads to

$$R(f^\star) = R(f) - \mathbf{E}[\,\|\mathbf{E}(\boldsymbol{y}|\boldsymbol{x}) - f(\boldsymbol{x})\|^2\,] \leq R(f).$$

The conditional expectation $\mathbf{E}(\boldsymbol{y}|\boldsymbol{x})$ is thus a minimum of the expected risk, which motivates the usage of the MSE distance.

The third main task is *density estimation*, where the goal is to estimate the unknown density P. As $f \in \mathcal{F}$ has to be a density, f takes values in $\mathbb{R}^+_\star$, and must satisfy the constraint $\int_{\mathcal{Z}} f(\boldsymbol{z})\, d\,\boldsymbol{z} = 1$. Usually we then try to minimize the Kullback-Leibler (see Cover and Thomas, 1991) distance between f and P. Hence,

$$Q(\boldsymbol{z},\, f) = -\log f(\boldsymbol{z})\,.$$

Empirical Risk Minimization

As already mentioned, machine learning aims to minimize the expected risk (2.3) using only the information of the training set (2.1). Unfortunately, it is not a simple optimization problem, because the distribution P is unknown in (2.3). We thus consider instead the minimization of the *empirical risk*

$$R_L : f \in \mathcal{F} \longmapsto \frac{1}{L} \sum_{l=1}^{L} Q(\boldsymbol{z}_l,\, f)\,. \tag{2.4}$$

The use of this empirical risk is strongly motivated by the weak law of large numbers which guarantees the convergence in probability of the empirical risk to the expected risk, when increasing the number of training examples, for any f independent from the training set:

$$\forall \epsilon > 0 \;\; \mathbf{P}[\,|R_L(f) - R(f)| > \epsilon\,] \underset{L \to \infty}{\longrightarrow} 0\,. \tag{2.5}$$

There is also a finer result obtained with the Hoeffding (1963) inequality, when an upper bound τ of $(\sup Q - \inf Q)$ exists:

$$\forall \epsilon > 0 \;\; \mathbf{P}[\,|R_L(f) - R(f)| > \tau\,\epsilon\,] \leq 2\,\exp\left(-2\,\epsilon^2\,L\right). \tag{2.6}$$

Unfortunately, these results do not guarantee the convergence of the *minimum* of the empirical risk to the *minimum* of the expected risk. One main

REF T. M. Cover and J. A. Thomas. *Elements of Information Theory.* John Wiley and Sons, 1991.

REF W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

concern of statistical learning theory is the study of this convergence, that is, the *non-trivial consistence* (as stated in Definition 2.2) of the Empirical Risk Minimization principle.

Definition 2.2 *The Empirical Risk Minimization principle is said "non-trivially consistent" if for any subset* $\mathcal{F}_c,\ c \in \mathbb{R}$

$$\mathcal{F}_c = \{\, f \ :\ R(f) \geq c \,\}$$

the following convergence in probability holds:

$$\forall \epsilon > 0 \quad \mathbf{P}[\, |\inf_{f \in \mathcal{F}_c} R_L(f) - \inf_{f \in \mathcal{F}_c} R(f)| > \epsilon \,] \underset{L \to \infty}{\longrightarrow} 0\,.$$

A Uniform Convergence Problem

It is interesting to note that if we strengthen the convergence in probability (2.5) to *uniform* convergence in probability

$$\forall \epsilon > 0 \quad \mathbf{P}[\, \sup_{f \in \mathcal{F}} |R_L(f) - R(f)| > \epsilon \,] \underset{L \to \infty}{\longrightarrow} 0\,, \tag{2.7}$$

then the non-trivial consistence of the Empirical Risk Minimization principle is ensured. Indeed, let us first note f_L, the optimal function found with the minimization of the empirical risk R_L, and $f^\star$, the function which minimizes the expected risk. Then given an arbitrary $\epsilon > 0$ and $\eta > 0$ the uniform convergence (2.7) implies that there exists a value $L(\epsilon, \eta)$ such that for all $L \geq L(\epsilon, \eta)$ with probability $1 - \eta$ the following inequalities hold:

$$\begin{aligned} |R_L(f_L) - R(f_L)| &\leq \epsilon \\ |R_L(f^\star) - R(f^\star)| &\leq \epsilon\,. \end{aligned} \tag{2.8}$$

Thus, for all $L \geq L(\epsilon, \eta)$, the difference between the minimum of the empirical risk and the expected risk can be bounded as the following:

$$\begin{aligned} |\inf_{f \in \mathcal{F}} R_L(f) - \inf_{f \in \mathcal{F}} R(f)| &= |R_L(f_L) - R(f^\star)| \\ &\leq |R_L(f_L) - R(f_L)| \\ &\qquad + |R(f_L) - R(f^\star)| \\ &\leq \epsilon + \underbrace{R(f_L) - R(f^\star)}_{\geq 0}\,. \end{aligned} \tag{2.9}$$

Moreover, we have

$$\begin{aligned} R(f_L) - R(f^\star) &= (R(f_L) - R_L(f_L)) \\ &\quad + \underbrace{R_L(f_L) - R_L(f^\star)}_{\leq 0} + (R_L(f^\star) - R(f^\star)) \\ &\leq 2\,\epsilon\,. \end{aligned} \tag{2.10}$$

Substituting (2.10) into (2.9) gives

$$|\inf_{f\in\mathcal{F}} R_L(f) - \inf_{f\in\mathcal{F}} R(f)| \leq 3\,\epsilon\,,$$

which proves the non-trivial consistence of the principle, as stated in Definition 2.2. Uniform convergence (2.7) is thus a *sufficient* condition for non-trivial consistence. In fact, with more complex derivations it is possible to demonstrate (see Vapnik, 1995) that only one-sided uniform convergence (see Theorem 2.1) is sufficient, and even *necessary*.

Theorem 2.1 *Let constants a and b be such that for all functions f in the set $\{\,Q(z,\,f),\;\, f\in\mathcal{F}\,\}$ the following inequality is satisfied:*

$$a \leq R(f) \leq b\,.$$

Then a necessary and sufficient condition for the non-trivial consistence of the Empirical Risk Minimization principle is that there is a one-sided uniform convergence of the empirical risk R_L to the expected risk R:

$$\forall\epsilon>0\;\;\mathbf{P}[\,\sup_{f\in\mathcal{F}}(R_L(f)-R(f))>\epsilon\,] \underset{L\to\infty}{\longrightarrow} 0\,.$$

Unfortunately, it is not possible to apply this result in practice, because it depends on the expected risk R, which depends itself on the unknown distribution P. However, as we will show in the next section, Vapnik and Chervonenkis (1971) developed some *sufficient* conditions which do not suffer from this problem.

From Glivenko-Cantelli to Vapnik-Chervonenkis

The Glivenko-Cantelli (1933) theorem is a classical theorem in probability theory, which states the uniform convergence of the empirical repartition function to the true repartition function, when increasing the number of samples used for the empirical estimation. It is possible to reformulate this theorem as in Theorem 2.2.

REF V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, second edition, 1995.

REF V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2): 264–280, 1971.

REF F. P. Cantelli. Sulla determinazione empirica della leggi di probilita. *Giornale dell'Instituto Italiano degli Attuari*, 4, 1933.

Theorem 2.2 (Glivenko-Cantelli) *If we consider the set of functions* $\mathcal{F}^\star = \{\, z \mapsto x - z,\ \forall x \in \mathbb{R}\,\}$ *and the loss function*

$$Q(z,\, f) = \begin{cases} 1 & \text{if } f(z) > 0 \\ 0 & \text{otherwise.} \end{cases}$$

then the following relation always holds, for any distribution P*:*

$$\forall \epsilon > 0 \quad \mathbf{P}[\sup_{f \in \mathcal{F}^\star} |R_L(f) - R(f)| > \epsilon\,] \underset{L \to \infty}{\longrightarrow} 0\,.$$

Hence, at least for the particular set of functions $\mathcal{F}^\star$, the uniform convergence (2.7) holds, and thus the Empirical Risk Minimization principle is non-trivially consistent. Vapnik and Chervonenkis (1971) proposed a generalization of this theorem, which depends on the complexity of the set of functions $\mathcal{F}$. The measure of this complexity is called *capacity* or *Vapnik-Chervonenkis dimension* (VC-dimension). We give below the definition of the capacity in the case of classification.

Definition 2.3 *The VC-dimension (capacity) of a set of indicator functions* $\{\, Q(\boldsymbol{z},\, f),\ f \in \mathcal{F}\,\}$ *is equal to the largest number of vectors* $z_1,\, \ldots,\, z_L$ *that can be separated into two different classes in all the* 2^L *possible ways using this set of functions.*

If we denote the VC-dimension as h, then the generalized Glivenko-Cantelli theorem can be stated in the following theorem.

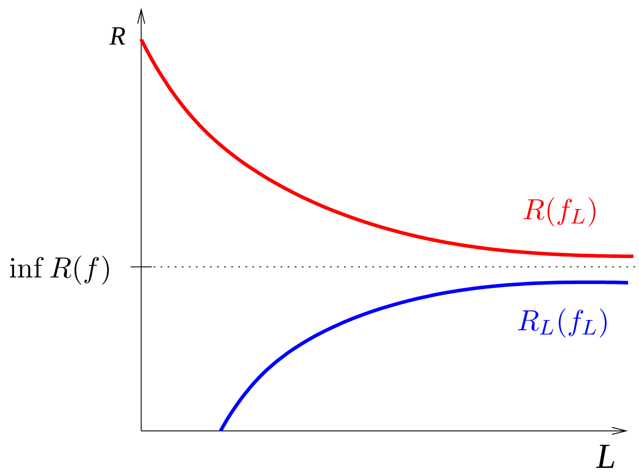
Theorem 2.3 (Vapnik-Chervonenkis) *Consider a set of functions* $\mathcal{F}$ *with a finite VC-dimension* h*, and a loss function* Q *such that there exists a constant* $\tau = \sup Q - \inf Q$*. Then for every distribution* P *and for all* $\epsilon > 0$*, the following inequality holds:*

$$\mathbf{P}[\sup_{f \in \mathcal{F}^\star} |R_L(f) - R(f)| > \epsilon\,] \leq 4 \exp\left(h\left(1 + \log(\frac{2\,L}{h})\right) - \left(\epsilon - \frac{1}{L}\right)^2 \frac{L}{\tau^2}\right).$$

Combining the result of Theorem 2.3 applied to the function f_L which minimizes the empirical risk R_L and the Hoeffding inequality (2.6) applied to the function $f^\star$ which minimizes the expected risk R, leads easily to Theorem 2.4.

Theorem 2.4 *Consider a set of functions* $\mathcal{F}$ *with a finite VC-dimension* h*, and a loss function* Q *such that there exists a constant* $\tau = \sup Q - \inf Q$*.*

REF V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2): 264–280, 1971.



▲ **Figure 2.1.** *Evolution of the empirical risk R_L and the expected risk R (applied to the function f_L found by the minimization of the empirical risk), with respect to the number of training examples. In this case, the Empirical Risk Minimization principle is non-trivially consistent.*

Let $f^\star$ be the minimum of the expected risk R, and f_L the minimum of the empirical risk R_L. Then for every distribution P and for all $\eta > 0$, the following inequality holds with at least probability $1 - 2\,\eta$:

$$R(f_L) \leq \inf_{f \in \mathcal{F}} R(f) + \frac{1}{L} + \tau \left(\sqrt{\frac{h\,\left(1 + \log(\frac{2\,L}{h})\right) - \log\left(\eta/4\right)}{L}} + \sqrt{\frac{-\log \eta}{2\,L}} \right) . \quad (2.11)$$

Note that if the Empirical Risk Minimization principle is applied to a set of functions $\mathcal{F}$ with a finite VC-dimension, then Theorem 2.3 guarantees the uniform convergence of the empirical risk R_L to the expected risk R when increasing the number of training examples. Hence, *if $\mathcal{F}$ has a finite VC-dimension the Empirical Risk Minimization principle is non-trivially consistent.* In practice, Theorem 2.3 and Theorem 2.4 allow us to give the global shape of the empirical risk and the expected risk with respect to the number of training examples, as depicted in Figure 2.1. It is also important to retain from (2.11) that *the more training examples we have, the better in generalization will the solution found by the Empirical Risk Minimization principle be.*

Model Selection

In practice, we usually have a fixed number of training examples. Thus, according to (2.11), we have to decrease the VC-dimension h of the set of functions $\mathcal{F}$ we consider, if we want to guarantee that the minimum of the expected risk is approached with the function which minimizes the empirical risk. However, if the VC-dimension of $\mathcal{F}$ is too small, then the empirical risk may not be well minimized, and as a consequence the generalization may be poor. This follows Occam's razor principle which states that the simplest solution which fits a problem should be selected: we have to choose a trade-off between the quality of the Empirical Risk Minimization and the complexity of the set of functions we consider. A formal analysis of this problem has been proposed by Vapnik and Chervonenkis (1974) and is known as *Structural Risk Minimization*.

Structural Risk Minimization

The minimum of the expected risk being unknown, the bound (2.11) of the expected risk for the function f_L which minimizes the empirical risk is intractable. Hence, we consider instead the following result obtained by applying Theorem 2.3 to the function f_L. For all $\eta > 0$, with at least probability $1-\eta$,

$$|R(f_L) - R_L(f_L)| \leq \frac{1}{L} + \tau \sqrt{\frac{h\,\left(1 + \log(\frac{2L}{h})\right) - \log(\eta/4)}{L}}\,. \tag{2.12}$$

By considering this equation, Vapnik and Chervonenkis (1974) proposed a new principle called *Structural Risk Minimization*. The idea is to minimize simultaneously the empirical risk, and the confidence interval given in (2.12). More formally, let us suppose that the set of functions $\mathcal{F}$ has a *structure* such that there exists p subsets $(\mathcal{F}_i)_{1 \leq i \leq p}$ such that

$$\mathcal{F}_1 \subset \mathcal{F}_2 \subset \cdots \subset \mathcal{F}_p = \mathcal{F}\,,$$

and where their capacity h_i are finite and ordered

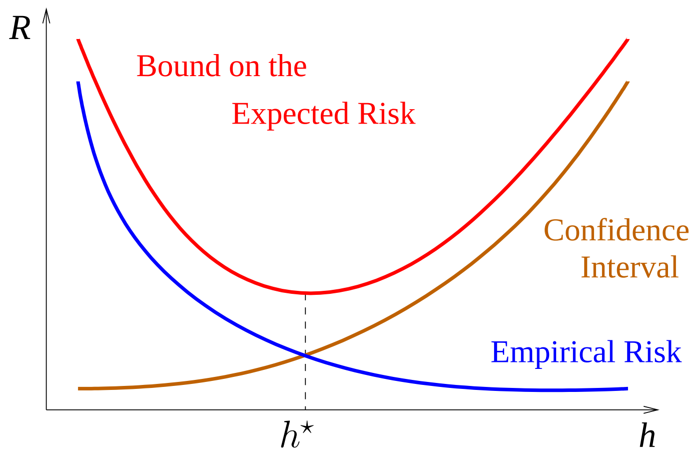
$$h_1 \leq h_2 \leq \cdots \leq h_p\,.$$

Then, if we note f_L^i the function found by minimizing the empirical risk over the set of function $\mathcal{F}_i$, it appears obvious that

$$R_L(f_L^p) \leq R_L(f_L^{p-1}) \leq \cdots \leq R_L(f_L^2) \leq R_L(f_L^1)\,.$$

REF V. N. Vapnik and A. Y. Chervonenkis. *The Theory of Pattern Recognition*. Nauka, 1974.

REF V. N. Vapnik and A. Y. Chervonenkis. *The Theory of Pattern Recognition*. Nauka, 1974.



▲ **Figure 2.2.** *Evolution of the empirical and expected risks with respect to the VC-dimension h. The training set is fixed. The Structural Risk Minimization principle aims at finding the optimal VC-dimension $h^\star$.*

Moreover, if we apply equation (2.12) to each subset $\mathcal{F}_i$, we obtain that the confidence interval between the expected risk and the empirical risk increases with respect to the VC-dimension h_i. We thus obtain the situation summarized in Figure 2.2. On the first hand, if the VC-dimension h_i is small, then the empirical risk $R_L(f_L^i)$ is high, and we say that the function f_L^i *under-fits* (or that we under-train). In that case, as the training performance is poor, the generalization performance is poor as well. On the other hand, if the VC-dimension h_i is large, then the empirical risk $R_L(f_L^i)$ is small. We say that f_L^i *over-fits* (or that we over-train). Here, the expected risk bound is high, and we cannot guarantee a good generalization. Thus, the Structural Risk Minimization principle aims at finding the optimal trade-off between the quality of the approximation on the training set and the complexity of the set of functions $\mathcal{F}_i$. This is also related to the bias-variance dilemma (Geman et al., 1992).

Validation Method

Even if the Structural Risk Minimization principle gives a good idea of the evolution of the expected risk with respect to the VC-dimension, as shown in Figure 2.2, it is often intractable in practice. Indeed, two problems occur: first, the VC-dimension is usually hard to compute, and thus it is not always possible to evaluate the bound (2.12). Moreover, the VC-dimension is also often

REF S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

very large as compared to the number of training examples: the bound (2.12) is in practice really sub-optimal. To overcome these problems, it is very common to use instead a *validation* set (or hold-out set) to estimate the expected risk.

The validation method supposes the existence of another set of observations $(\tilde{z}_l)_{1 \leq l \leq \tilde{L}}$ (the *validation* set) drawn independently and identically from the unknown distribution P, and independent from the training set $(z_l)_{1 \leq l \leq L}$. Then we can compute an estimate $\tilde{R}_{\tilde{L}}$ of the expected risk for the function f_L which minimizes the empirical risk:

$$\tilde{R}_{\tilde{L}} = \frac{1}{\tilde{L}} \sum_{l=1}^{\tilde{L}} Q(\tilde{\boldsymbol{z}}_l,\, f_L)\,. \tag{2.13}$$

As f_L does not depend on the validation set, we can obtain from the Hoeffding inequality (2.6) that for all $\eta > 0$, with probability $1 - \eta$, we have

$$|R(f_L) - \tilde{R}_{\tilde{L}}(f_L)| \leq \tau \sqrt{\frac{-\log(\eta/2)}{2L}}\,,$$

where $\tau = (\sup Q - \inf Q)$. This bound is quite tight in practice: for example, with 15000 validation examples we are ensured to approximate the expected risk with less than 1% error, with a probability of 0.9 (in the classification case). However, we often need more precision. We are thus forced to make some hypothesis on the data distribution. For classification, we usually rely on standard statistical tests such that the McNemar test (see for example Siegel, 1956) or the proportion test (Snedecor and Cochran, 1989). For more information, a recent comparison of statistical tests for machine learning is given in Dietterich (1998).

Regularization Theory

Regularization theory, first introduced by Hadamard (1932) and Tikhonov and Arsenin (1977), highlights the fact that many mathematical problems are *ill-posed* (that is, not well-posed as proposed by Definition 2.4), and aims at fixing these problems.

REF S. Siegel. *Nonparametric Methods for the Behavioral Sciences.* McGraw-Hill, 1956.

REF G. W. Snedecor and W. G. Cochran. *Statistical Methods.* Iowa State University Press, eighth edition, 1989.

REF T. G. Dietterich. Approximate statistical test for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1924, 1998.

REF J. Hadamard. *Le problème de Cauchy et les équations aux dérivées partielles linéaires hyperboliques.* Hermann, 1932.

REF A. N. Tikhonov and V. Y. Arsenin. *Solution of ill-posed problems.* Winston & Sons, 1977.

Definition 2.4 *We say that a mathematical problem is well-posed (in the sense of Hadamard) if the solution exists, is unique, and smoothly depends on the data.*

For example, the problem of finding a linear operator f which satisfies $A\,f = b$ (where A is a matrix and b is a vector) is ill-posed if the matrix A is not very well conditioned: if we know an approximate version $\tilde{b}$ of b, then the solution $\tilde{f}$ of the problem $A\,\tilde{f} = \tilde{b}$ may not be close to the solution f of the original problem. The machine learning field has a strong interest in regularization theory, because the Empirical Risk Minimization problem is usually ill-posed: a slight change in the training set may considerably change the solution.

In order to transform an ill-posed problem into a well-posed problem, Tikhonov proposed to first consider a measure of regularity $\Omega(f)$ on the set of functions $\mathcal{F}$. Then, instead of considering the empirical risk, we consider the *regularized* empirical risk

$$R_L(f) + \mu\,\Omega(f)\,, \tag{2.14}$$

where $\mu \in \mathbb{R}^+$ is a hyper-parameter which controls the regularization. If we note f_μ the minimum of (2.14), then minimizing the regularized risk is equivalent to minimizing the empirical risk on the set of functions $\mathcal{F}_\mu = \{\, f \in \mathcal{F},\ \Omega(f) \leq \Omega(f_\mu)\,\}$. When doing regularization, we are thus reducing the set of functions we consider: in some cases, if we control the VC-dimension of $\mathcal{F}_\mu$, regularization can implement the Structural Risk Minimization principle (Evgeniou et al., 2000). Finally, note that regularization theory was introduced very early in machine learning (see Plaut et al., 1986), and has been studied in detail more recently by Girosi et al. (1995).

Summary

In this chapter, we introduced the notion of machine learning in a statistical framework. We defined learning as a process of choosing an appropriate function which minimizes an "expected risk". As the expected risk is not calculable, we defined the Empirical Risk Minimization principle, and we gave an intuition of when the usage of this principle is relevant for approaching

REF T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13(1):1–50, 2000.

REF D. C. Plaut, S. J. Nowlan, and G. E. Hinton. Experiments on learning by back propagation. Technical Report CMU-CS-86-126, Carnegie-Mellon University, Computer Science Department, 1986.

REF F. Girosi, M. Jones, and T. Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7(2):219–269, 1995.

the expected risk minimum. After the definition of the complexity of a set of functions, we also gave a generalization of the Glivenko-Cantelli theorem, which guarantees in a more precise way the relevance of the Empirical Risk Minimization for “not too complex” sets of functions. We then introduced the Structural Risk Minimization principle which helps in choosing the complexity of the set of functions we consider. As this principle is usually intractable in practice, we briefly introduced the technique of validation. Finally, we gave an overview of regularization theory, which has some links with learning theory. Now that we have introduced the basics of statistical machine learning theory, we will review in the next chapter the main machine learning algorithms that we will consider in this thesis.

3 Algorithms

During the last fifty years of machine learning research, several algorithms have been developed in the community. However, time has made its own selection, and only a few algorithms are really used in practice. We propose in this chapter an introduction to several algorithms which were among the most popular in their time: the Perceptron algorithm, Multi Layer Perceptrons (MLPs), Mixtures of Experts (MoEs), and finally Support Vector Machines (SVMs).

Framework

We consider in this chapter a training set $(\boldsymbol{x}_l, \boldsymbol{y}_l)_{1 \leq l \leq L}$, where $\boldsymbol{x}_l \in \mathbb{R}^d$ represents the input vector of the l^{th} example, and $\boldsymbol{y}_l$ represents its corresponding target. We will focus on two problems: a two-class classification problem and a regression problem. Note that we differentiate between *parameters* (also called *weights*, and which are trained by an algorithm), and *hyper-parameters* (which are parameters chosen by the user of an algorithm).

Classification

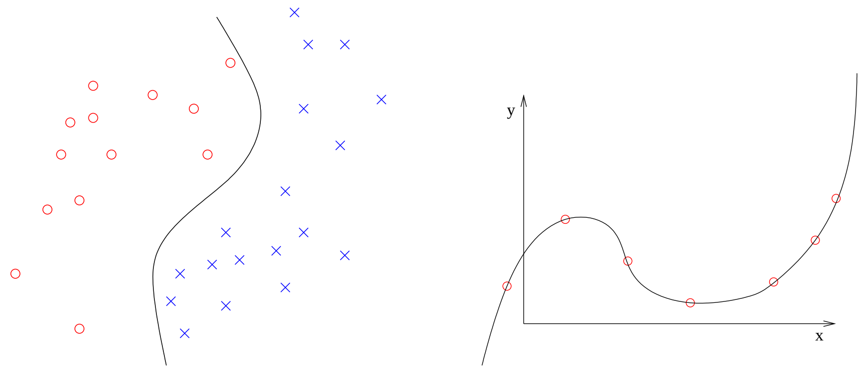
This classification problem supposes $y_l \in \{-1, 1\}$. The aim is to find a model

$$\begin{array}{ccc} \mathbb{R}^d & \longrightarrow & \mathbb{R} \\ \boldsymbol{x} & \mapsto & f_{\boldsymbol{\theta}}(\boldsymbol{x}) \end{array} \tag{3.1}$$

such that the *decision surface*

$$\{\, \boldsymbol{x} \in \mathbb{R}^d,\ f_{\boldsymbol{\theta}}(\boldsymbol{x}) = 0 \,\}$$

separates positive class examples and negative class examples. In other words, for any example $(\boldsymbol{x},\, y)$, the class assigned by the model is "sign $f_{\boldsymbol{\theta}}(\boldsymbol{x})$" and we



(a) (b)

▲ **Figure 3.1.** *(a) A classifier is a decision surface which separates two sets of examples, here represented by crosses (blue) and circles (red), separated by the black curve. (b) Regression aims to find a model which predicts a target $\boldsymbol{y}$ given an input example $\boldsymbol{x}$, for all examples $(\boldsymbol{x}, \boldsymbol{y})$. Here the black curve fits red circles.*

would like to have $y = \operatorname{sign} f_{\boldsymbol{\theta}}(\boldsymbol{x})$, where we define

$$\operatorname{sign}(z) = \begin{cases} -1 & \text{if } z \leq 0 \\ 1 & \text{if } z > 0\,. \end{cases}$$

The generic real vector $\boldsymbol{\theta}$ represents the parameters that we have to train. The situation is summarized in Figure 3.1a.

Regression

For regression problems, we have $\boldsymbol{y}_l \in \mathbb{R}^n$, and we would like to find a model

$$\begin{array}{ccc} \mathbb{R}^d & \longrightarrow & \mathbb{R}^n \\ \boldsymbol{x} & \longmapsto & f_{\boldsymbol{\theta}}(\boldsymbol{x}) \end{array} \tag{3.2}$$

such that for any example $(\boldsymbol{x}, \boldsymbol{y})$ the target $\boldsymbol{y}$ is predicted knowing the input vector $\boldsymbol{x}$, that is $\boldsymbol{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$. As for classification, $\boldsymbol{\theta}$ represents the parameters we have to train. The situation is summarized in Figure 3.1b.

Perceptrons

The original Perceptron algorithm introduced by Rosenblatt (1957), is usually considered one of the first machine learning algorithms. It is a *linear*

REF F. Rosenblatt. The perceptron: a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, N.Y., 1957.

classifier. Thus, we consider the model

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b\,, \tag{3.3}$$

where $\boldsymbol{\theta} = (\boldsymbol{w}\,, b)$ are the parameters to train. The training algorithm proposed by Rosenblatt (see Algorithm 3.1) considers each training example $(\boldsymbol{x}_l,\, y_l)$ successively. If one of them is misclassified, that is if $y_l\, f_{\boldsymbol{\theta}}(\boldsymbol{x}_l) \leq 0$, then the

Algorithm 3.1 Original Perceptron Algorithm

Initialize $\boldsymbol{w}$ and b to zero
repeat
  for $l \in \{1..L\}$ **do**
    if $y_l\,(\boldsymbol{w} \cdot \boldsymbol{x}_l + b) \leq 0$ **then**
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_l\,\boldsymbol{x}_l$
      $b \leftarrow b + y_l$
    end if
  end for
until termination criterion

parameters are updated such that $y_l\, f_{\boldsymbol{\theta}}(\boldsymbol{x}_l)$ is increased. Indeed, if we denote the parameters after t updates as $\boldsymbol{\theta}_t$, $\boldsymbol{w}_t$ and b_t and the index of the example updated at time t as l_t, we then have

$$y_{l_t}\, f_{\boldsymbol{\theta}_{t+1}}(\boldsymbol{x}_{l_t}) = y_{l_t}\, f_{\boldsymbol{\theta}_t}(\boldsymbol{x}_{l_t}) + \underbrace{\|\boldsymbol{x}_{l_t}\|^2 + 1}_{>0}\,.$$

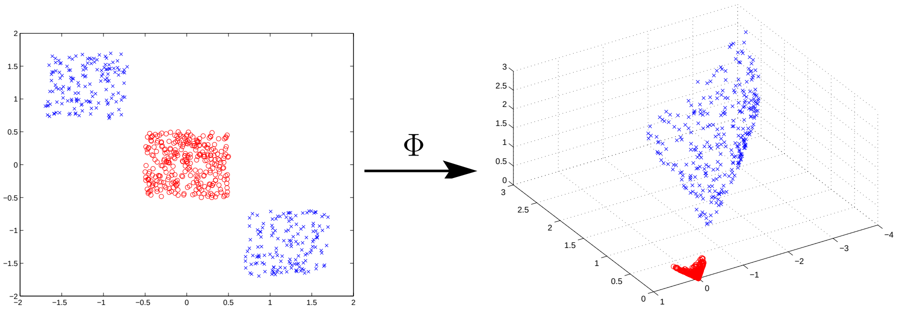
The training is stopped when all examples are well classified, or according to some other criterion (such as the error on a separate validation set) if the classes are not linearly separable. It has been shown by Novikoff (1962) that this algorithm converges toward a solution which separates classes, if the classes are linearly separable.

Φ-Machines

A linear classifier such as the Perceptron algorithm usually performs poorly in real life problems, where classes are rarely linearly separable with respect to available inputs. To overcome this problem, Φ-Machines were introduced very early in machine learning (see Nilsson, 1965). The idea of Φ-Machines, as

REF A. B. J. Novikoff. On convergence proofs on perceptrons. In Polytechnic Institute of Brooklyn, editor, *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622, 1962.

REF N. J. Nilsson. *Learning Machines.* McGraw-Hill, 1965.



▲ **Figure 3.2.** *Mapping the input space into a larger space may facilitate the separation of the classes. In this example, the mapping function from* $\mathbb{R}^2$ *into* $\mathbb{R}^3$ *was* $\Phi(\boldsymbol{x}) = \Phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1\, x_2\, , x_2^2)$. *With this mapping, the circles (red) and the crosses (blue), which are not linearly separable in* $\mathbb{R}^2$, *become linearly separable in* $\mathbb{R}^3$.

suggested in Figure 3.2, is to first send the input vectors into another space (the *feature space*), usually larger than the initial space. This is strongly motivated by Theorem 3.1, introduced by Cover (1965) which relates the capacity of a Perceptron to the dimension of the data. As the capacity in classification is (according to Definition 2.3) also related to the largest number of examples that a set of functions can separate, one might think that in a higher dimensional space the classes may be more linearly separable.

Theorem 3.1 (Cover) *The set of hyperplanes in dimension d*

$$\{\, \boldsymbol{x} \mapsto \boldsymbol{w} \cdot \boldsymbol{x} + b, \;\; \boldsymbol{w} \in \mathbb{R}^d,\, b \in \mathbb{R} \,\}$$

has a capacity of $d + 1$.

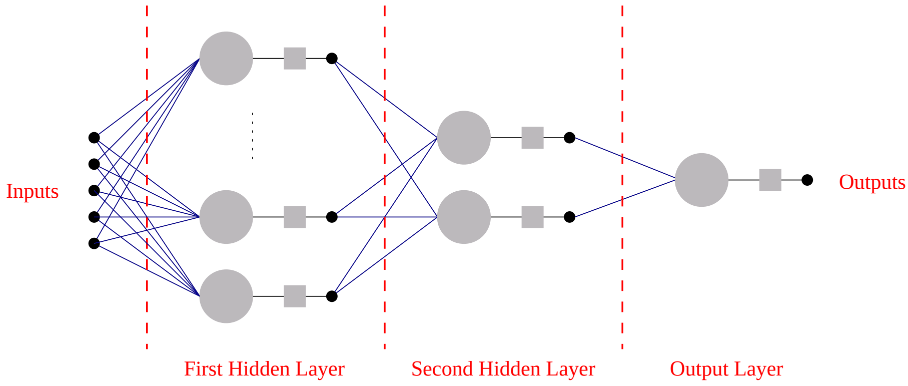
Thus, after choosing an arbitrary Φ function, we apply the Perceptron algorithm on examples $(\Phi(\boldsymbol{x}_l),\, y_l)$ instead of $(\boldsymbol{x}_l,\, y_l)$. Hence, the Perceptron model (3.3) becomes

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w} \cdot \Phi(\boldsymbol{x}) + b\,. \tag{3.4}$$

Multi Layer Perceptrons

Even with its Φ-Machine extension, the Perceptron algorithm is limited in practice, because the choice of Φ is completely arbitrary. In order to enlarge

REF T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, 14: 326–334, 1965.



▲ **Figure 3.3.** *Description of a Multi Layer Perceptron. Gray circles represent Perceptrons. Gray squares represent transfer functions.*

separation capabilities, one can combine several Perceptrons in a non-linear way. This is what a Multi Layer Perceptron (MLP) does.

Description

As described in Figure 3.3, an MLP is a kind of function structured in several layers, which are applied in a feed-forward manner. Each layer contains several Perceptrons (the *units*), and their output is passed through a non-linear function, usually called a *transfer* function. This is usually a sigmoid function

$$z \mapsto \frac{1}{1+\exp(-z)}\,,$$

or a hyperbolic tangent function. The last layer of the MLP is also called the *output layer*, and other layers are called *hidden layers*. In the case of our regression problem, the output layer contains as many Perceptrons as the dimension of the target vectors $\boldsymbol{y} \in \mathbb{R}^n$ to predict. The transfer function in the last layer is often removed, to be able to approximate any kind of function. In the case of our two-class classification problem, the output layer contains only one Perceptron. The transfer function in the output layer may be removed as well, but it is not necessary. For simplicity, we will consider only the case of MLPs with one hidden layer, because they are easier to tune than MLPs with more than one hidden layer, and also because it has been shown by Hornik et al. (1989) that they are universal approximators (which means that given a finite number of training examples $\boldsymbol{x}_i$ and a target function g, there exists

REF K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

an MLP $f_{\boldsymbol{\theta}}$ that can approximate g as close as desired, for all $\boldsymbol{x}_i$). Moreover, we will focus only on MLPs for classification in this thesis. The MLP that we consider can thus be rewritten as

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w} \cdot \Phi(\boldsymbol{x}) + b\,, \tag{3.5}$$

where $\Phi(\cdot) = (\Phi_1(\cdot),\, \Phi_2(\cdot),\, \ldots,\, \Phi_N(\cdot))$ represents the hidden layer composed of N hidden units. The n-th hidden unit is described as

$$\Phi_n(\boldsymbol{x}) = h(\boldsymbol{v}_n \cdot \boldsymbol{x} + a_n)\,, \tag{3.6}$$

where h is a transfer function, and $(\boldsymbol{v}_n,\, a_n) \in \mathbb{R}^d \times \mathbb{R}$ are the parameters of the hidden unit n to train. Note that with this notation, an MLP looks like a Φ-Machine (3.4) where the feature space (represented by Φ) is trained. If we consider Φ as fixed, we thus recover the Φ-Machine model. Moreover, if we consider $\Phi(\boldsymbol{x}) = \boldsymbol{x}$, we obtain the Perceptron model.

Training

The training of MLPs is usually achieved with the optimization of a given criterion $((\boldsymbol{x},\, \boldsymbol{y}),\, f_{\boldsymbol{\theta}}) \mapsto Q((\boldsymbol{x},\, \boldsymbol{y}),\, f_{\boldsymbol{\theta}})$ over the training set. It leads to the minimization of the empirical risk

$$R_L : f_{\boldsymbol{\theta}} \to \frac{1}{L} \sum_{l=1}^{L} Q((\boldsymbol{x}_l,\, \boldsymbol{y}_l),\, f_{\boldsymbol{\theta}})\,, \tag{3.7}$$

using a *gradient based learning method*.

Criteria

As gradient based methods are used for training MLPs, we need to use *derivable* criteria. The most often used criterion, which can be applied both for classification and regression, is probably the Mean Squared Error (MSE) criterion

$$Q((\boldsymbol{x},\, \boldsymbol{y}),\, f_{\boldsymbol{\theta}}) = \frac{1}{2}\, \|\boldsymbol{y} - f_{\boldsymbol{\theta}}(\boldsymbol{x})\|^2\,. \tag{3.8}$$

We already highlighted in Chapter 2 that the minimum of the expected risk using the MSE criterion is the condition expectation $\mathbf{E}(\boldsymbol{y}|\boldsymbol{x})$, which motivates its usage. It is also possible to show that from a likelihood perspective, the MSE criterion is equivalent to maximizing the likelihood under the hypothesis that the observations $\boldsymbol{y}_l$ are generated from a smooth function with added noise $\boldsymbol{\xi}$ following a Gaussian distribution $\mathcal{N}(0,\, 1)$:

$$\boldsymbol{y}_l \sim f_{\boldsymbol{\theta}}(\boldsymbol{x}_l) + \boldsymbol{\xi}\,. \tag{3.9}$$

Indeed, let us consider the log-likelihood over the training set

$$\log \mathcal{L}(\boldsymbol{\theta}) = \log \left(\prod_{l=1}^{L} \mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{y}_l | \boldsymbol{x}_l) \right) = \sum_{l=1}^{L} \log \mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{y}_l | \boldsymbol{x}_l)\,, \tag{3.10}$$

where $\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{y}_l | \boldsymbol{x}_l)$ is the conditional distribution of $\boldsymbol{y}_l$ knowing $\boldsymbol{x}_l$. Then, if $\boldsymbol{y}_l$ is generated according to the distribution (3.9), we get

$$\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{y}_l | \boldsymbol{x}_l) = \frac{1}{\sqrt{2\,\pi}} \exp(-\frac{1}{2} \, \|\boldsymbol{y}_l - f_{\boldsymbol{\theta}}(\boldsymbol{x}_l)\|^2)$$

and thus the log-likelihood can be rewritten as

$$\log \mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{2} \log(2\,\pi) - \frac{1}{2} \sum_{l=1}^{L} \|\boldsymbol{y}_l - f_{\boldsymbol{\theta}}(\boldsymbol{x}_l)\|^2\,. \tag{3.11}$$

Hence, maximizing the log-likelihood (3.11) is completely equivalent to the minimization of the empirical risk (3.7) using the MSE criterion (3.8). Therefore, for classification problems the MSE criterion may not be appropriate, since y_l is a binary variable, which does not fit the Gaussian model (3.9). One could consider instead (see Bishop, 1995) y_l coming from a Bernoulli distribution:

$$\mathbf{P}_{\boldsymbol{\theta}}(y_l | \boldsymbol{x}_l) = \begin{cases} \frac{1}{1+\exp(-f_{\boldsymbol{\theta}}(\boldsymbol{x}_l))} & \text{if} \quad y_l = 1 \\ 1 - \frac{1}{1+\exp(-f_{\boldsymbol{\theta}}(\boldsymbol{x}_l))} & \text{if} \quad y_l = -1\,, \end{cases} \tag{3.12}$$

where we add a sigmoid at the output $f_{\boldsymbol{\theta}}$ of the MLP to guarantee that output values lie between 0 and 1. By noticing that (3.12) can be rewritten as

$$\mathbf{P}_{\boldsymbol{\theta}}(y_l | \boldsymbol{x}_l) = \frac{1}{1 + \exp(-y_l\, f_{\boldsymbol{\theta}}(\boldsymbol{x}_l))}\,,$$

it becomes straightforward to compute the log-likelihood (3.10)

$$\log \mathcal{L}(\boldsymbol{\theta}) = -\sum_{l=1}^{L} \log(1 + \exp(-y_l\, f_{\boldsymbol{\theta}}(\boldsymbol{x}_l)))\,.$$

This leads to another criterion for classification problems

$$Q((\boldsymbol{x},\, y),\, f_{\boldsymbol{\theta}}) = \log(1 + \exp(-y\, f_{\boldsymbol{\theta}}(\boldsymbol{x})))\,, \tag{3.13}$$

that is often called the *Cross-Entropy* (CE) criterion, first introduced by Hopfield (1987).

REF C. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

REF J. J. Hopfield. Learning algorithms and probability distributions in feed-forward and feed-back networks. In *Proceedings of the National Academy of Sciences*, volume 84, pages 8429–8433, 1987.

Relevance of the MSE and CE Criteria for Classification

We have shown that the MSE and CE criteria are equivalent to maximizing a likelihood over the training set. Hence, following the theory developed by Vapnik (1995), if we had an infinite number of training examples the "empirical" posterior probability $\mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x})$ which maximizes the likelihood (3.10) would be equal to the true posterior probability $\mathbf{P}(y|\boldsymbol{x})$. Moreover, it is easy to show (as demonstrated by Duda and Hart, 1973) that taking the decision y which maximizes $\mathbf{P}(y|\boldsymbol{x})$ leads to the minimum possible classification error rate. The use of the MSE and CE criteria for classification is therefore justified.

Gradient Descent

A gradient-based learning method (first introduced by Cauchy, 1847) is used to minimize the empirical risk over the training set (3.7). Many gradient-based techniques exist (see Battiti, 1992), and usually rely on a local Taylor approximation of the empirical risk around the current vector of parameters $\boldsymbol{\theta}_t$,

$$R_L(f_{\boldsymbol{\theta}}) = R_L(f_{\boldsymbol{\theta}_t}) + (\boldsymbol{\theta} - \boldsymbol{\theta}_t) \cdot \frac{\partial R_L(f_{\boldsymbol{\theta}_t})}{\partial \boldsymbol{\theta}} + (\boldsymbol{\theta} - \boldsymbol{\theta}_t)^{\mathrm{T}} \frac{\partial^2 R_L(f_{\boldsymbol{\theta}_t})}{\partial \boldsymbol{\theta}^2} (\boldsymbol{\theta} - \boldsymbol{\theta}_t) + \dots \,. \tag{3.14}$$

The most popular technique remains the *gradient descent* also called *steepest gradient descent*. It was first applied to MLPs by LeCun (1985) and Rumelhart et al. (1986), and is known as the *back-propagation* technique, because the criterion error is efficiently "back-propagated" through the MLP, using chained derivatives. Gradient descent is a first order technique, which means it considers only the gradient term $\frac{\partial R_L}{\partial \boldsymbol{\theta}}$ in the Taylor approximation (3.14). In other words, the empirical risk is locally approximated as a line, with respect to the parameters $\boldsymbol{\theta}$. Under this assumption, the best we can do to reduce R_L, is to

---

REF V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, second edition, 1995.

REF R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley & Sons, New York, 1973.

REF A. Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. In *Compte Rendu Hebdomadaire des Séances de l'Académie des Sciences*, volume 25, pages 536–538, Paris, France, 1847.

REF R. Battiti. First and second-order methods for learning: Between steepest descent and Newton's method. *Neural Computation*, 4(2):141–166, 1992.

REF Y. LeCun. A learning scheme for asymmetric threshold networks. In *Proceedings of Cognitiva 85*, pages 599–604, Paris, France, 1985.

REF D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by back-propagating errors. In D.E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, 1986.

use the rule

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \lambda_t \frac{\partial R_L(\boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}} \tag{3.15}$$

to update the parameters, where λ_t is a *learning rate* (fixed or depending on the number of updates t) that has to be tuned with a trial-and-error process. Learning methods using the update rule (3.15) are referred as *batch* methods, since an entire "batch" of the training set must be considered before updating the parameters. However, when using large training sets, a lot of redundancies appear in the data. Hence, as highlighted by Bottou (1991b), it is in practice much more efficient to estimate the gradient after considering only one training example, and thus to use the update rule

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \lambda_t \frac{\partial Q((\boldsymbol{x}_l,\, \boldsymbol{y}_l),\, f_{\boldsymbol{\theta}_t})}{\partial \boldsymbol{\theta}}\,, \tag{3.16}$$

for all examples $(\boldsymbol{x}_l,\, \boldsymbol{y}_l)$. We refer as *iteration* or *epoch* the application of this rule over the whole training set. This method is called *stochastic* gradient descent, and was first introduced by Robbins and Monro (1951). This is the method we will consider, as we are considering large databases in this thesis.

Convergence of Stochastic Gradient Descent

The convergence of stochastic gradient descent has been well studied in a general way by Bottou (1991a), who also proposed extended results in (Bottou, 1998). The main theorem he expresses (see Theorem 3.2) proves the convergence of stochastic gradient descent under several assumptions while considering the general update

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \lambda_t\, G_{\boldsymbol{\theta}}(\boldsymbol{x}_l,\, \boldsymbol{y}_l)\,, \tag{3.17}$$

instead of the update (3.16).

Theorem 3.2 (Bottou) *Given any distribution $P(\boldsymbol{x},\, \boldsymbol{y})$, we define the risk over all training examples*

$$R(f) = \mathbf{E}\left[\, Q((\boldsymbol{x},\, \boldsymbol{y}),\, f)\,\right] = \int Q((\boldsymbol{x},\, \boldsymbol{y}),\, f)\, P(\boldsymbol{x},\, \boldsymbol{y})\, d(\boldsymbol{x},\, \boldsymbol{y})\,.$$

REF L. Bottou. *Une Approche Théorique de l'Apprentissage Connexioniste; Applications à la reconnaissance de la Parole*. PhD thesis, Université de Paris Sud, Orsay, 1991b.

REF H. Robbins and S. Monro. A stochastic approximation method. In *Annals of Mathematical Statistics*, volume 22, pages 400–407, 1951.

REF L. Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, Nimes, France, 1991a. EC2.

REF L. Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

We suppose $R(f_{\boldsymbol{\theta}})$ is differentiable with respect to $\boldsymbol{\theta}$ up to the third order. If the following assertions are true (where $\|\cdot\|_2$ is the Euclidean norm),

$$
\begin{aligned}
i)&\quad \exists R_{min},\ \forall \boldsymbol{\theta},\ R_{min} < R(f_{\boldsymbol{\theta}})\\
ii)&\quad \mathbf{E}\left[G_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{y})\right] = \frac{\partial R(f_{\boldsymbol{\theta}})}{\partial \boldsymbol{\theta}}\\
iii)&\quad \sum_{t=0}^{\infty} \lambda_t = \infty,\ \sum_{t=0}^{\infty} \lambda_t^2 < \infty\\
iv)&\quad \exists A,\, B,\ \forall \boldsymbol{\theta},\ \mathbf{E}\left[\|G_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{y})\|_2^2\right] < A + B\,\|\boldsymbol{\theta}\|_2^2\\
v)&\quad \exists D,\ \inf_{\|\boldsymbol{\theta}\|_2 > D} \boldsymbol{\theta}\cdot \frac{R(f_{\boldsymbol{\theta}})}{\partial \boldsymbol{\theta}} > 0\\
vi)&\quad \exists E > D,\, \exists K,\, \forall \boldsymbol{\theta},\ \sup_{\|\boldsymbol{\theta}\|_2 > E} \|G_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{y})\|_2 \le K
\end{aligned}
$$

then $R(f_{\boldsymbol{\theta}_t})$ converges with probability 1 and $\partial R(f_{\boldsymbol{\theta}_t})/\partial\boldsymbol{\theta}$ converges to 0 with probability 1 when $t \to \infty$.

Assertion $i)$ ensures the existence of a minimum. Assertion $ii)$ assumes we are performing gradient descent on the expected risk. Assertion $iii)$ is a common schedule of the learning rate. It imposes the choice of a decreasing learning rate. Assertion $iv)$ guarantees that the update term in (3.17) does not grow more than linearly with the size of the parameters. Assertion $v)$ prevents the gradient descent from getting stuck in a plateau, where the parameter vector $\boldsymbol{\theta}$ may grow indefinitely without escaping. Note that it is possible to show (see Bottou, 1998) that assertion $v)$ ensures a *global confinement* of the parameters. In other words, $\boldsymbol{\theta}$ will be confined to a *bounded* region containing the origin. Assertion $vi)$ ensures that for a small norm of $\boldsymbol{\theta}$, the gradient is bounded regardless of any example $(\boldsymbol{x},\boldsymbol{y})$.

When the criterion

$$\theta \mapsto Q((\boldsymbol{x}, y),\, f_{\boldsymbol{\theta}})$$

is differentiable and has an integrable gradient, it is straightforward to prove that assertion $ii)$ is satisfied for the update (3.16) using Lebesgue's dominated convergence theorem to swap the integration and differentiation operators. Assertion $iv)$ can be verified with few derivations, for all models using stochastic gradient descent presented in this thesis. However, assertion $v)$ is not true in general. In particular, it is not satisfied when using an MLP with a hyperbolic tangent transfer function in the hidden units (3.6). As pointed out by Bottou

REF L. Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

REF L. Bottou. *Une Approche Théorique de l'Apprentissage Connexioniste; Applications à la reconnaissance de la Parole.* PhD thesis, Université de Paris Sud, Orsay, 1991b.

(1991b), we could instead use the *slanted* hyperbolic tangent

$$z \mapsto \tanh(z) + \epsilon\, z\,,$$

to guarantee the validity of assertion v) (where ϵ is an arbitrarily chosen real value). Nevertheless, it seems that this does not have any impact in practice, as long as the parameters of the model are initialized correctly (see LeCun et al., 1998).

Note that Theorem 3.2 proves the convergence of the stochastic gradient descent to an *extremal point*. This includes local and global minima, but also local maxima, saddle points and plateaus. According to Bottou (1998) saddle points and local maxima are usually unstable solutions, and plateaus are prevented with assertion v).

Also note that this theorem proves the convergence of stochastic gradient descent using the *expected* risk in the case of an *infinite* number of training examples. However, if we consider a *finite* number of training examples L, we can apply the theorem with the particular "counting" distribution

$$dP(\boldsymbol{x},\, \boldsymbol{y}) = \frac{1}{L} \sum_{l=1}^{L} \delta_{(\boldsymbol{x},\, \boldsymbol{y}) - (\boldsymbol{x}_l,\, \boldsymbol{y}_l)}\,,$$

where δ is the Dirac distribution. Then, Theorem 3.2 proves the convergence of stochastic gradient descent to an extremal point of the *empirical* risk.

Mixtures of Experts

Mixtures of Experts (MoEs), first introduced by Jacobs et al. (1991), represent a function $f_{\boldsymbol{\theta}}$ as a combination of simpler functions which are called *experts*. More formally, given an input example $\boldsymbol{x}$, the following decomposition with K experts is built:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{k=1}^{K} g_{\boldsymbol{\theta},k}(\boldsymbol{x})\, f_{\boldsymbol{\theta},k}(\boldsymbol{x})\,, \tag{3.18}$$

with the conditions

$$\sum_{k=1}^{K} g_{\boldsymbol{\theta},k} = 1 \quad \text{and} \quad g_{\boldsymbol{\theta},k} \geq 0 \quad \forall k\,. \tag{3.19}$$

REF Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In G.B. Orr and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, pages 9–50. Springer, 1998.

REF L. Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

REF R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixture of local experts. *Neural Computation*, 3(1):79–87, 1991.

The function $f_{\boldsymbol{\theta},k}$ is the output of the k-expert, and $g_{\boldsymbol{\theta}}(\cdot) = \{g_{\boldsymbol{\theta},1}(\cdot), \ldots, g_{\boldsymbol{\theta},K}(\cdot)\}$ is the *gater*, which provides a real weight for each expert, given an input $\boldsymbol{x}$. The conditions (3.19) on the gater can be easily achieved using a *soft-max* function: given any outputs $\{\tilde{g}_{\boldsymbol{\theta},1}(\cdot), \ldots, \tilde{g}_{\boldsymbol{\theta},K}(\cdot)\}$, the outputs

$$g_{\boldsymbol{\theta},k}(\cdot) = \frac{\exp(\tilde{g}_{\boldsymbol{\theta},k}(\cdot))}{\sum_{j=1}^{K} \exp(\tilde{g}_{\boldsymbol{\theta},j}(\cdot))}$$

satisfy the required conditions.

The experts and the gater could be any machine learning algorithm. The underlying idea of MoEs is that if the function to approximate is complex but can be easily decomposed into several simpler functions, each acting on a different input subspace, then MoEs should be more appropriate, and the training of such models should be easier.

Training

The training of MoEs completely depends on the nature of the experts and the gater. For example, with MLP-experts and MLP-gater, we could use gradient descent, as for the training of MLPs. In the case of probabilistic models, it is also possible to use the Expectation-Maximization training algorithm (see for example Jordan and Jacobs, 1994).

Support Vector Machines

Support Vector Machines (SVMs) were first invoked by Vapnik and Lerner (1963) as a large margin linear classifier. Step by step, they were improved and extended to the non-linear case until the algorithm proposed in Cortes and Vapnik (1995). The same year, an extension for regression problems was proposed by Vapnik (1995). We introduce these algorithms in a chronological way in this section.

Classification

SVMs were originally a kind of linear classifier. We thus define the model as

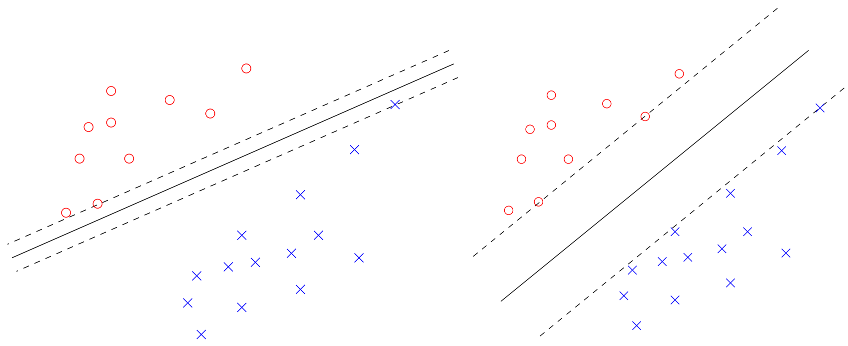
$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b \,. \tag{3.20}$$

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

REF V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24, 1963.

REF C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

REF V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, second edition, 1995.



(a) (b)

▲ **Figure 3.4.** *Two hyperplanes for separating crosses (blue) and circles (red). (a) Hyperplane with a small margin. (b) Hyperplane with a large margin. The margin is the distance between the two dashed hyperplanes.*

However, SVMs not only aim at separating two classes (as does the Perceptron algorithm, for example) but also at maximizing the margin between these two classes, as depicted in Figure 3.4. The intuitive idea is that a hyperplane with a large margin should be more resistant to noise than a hyperplane with a small margin. SVMs are thus often referred as *large margin classifiers*. More formally, we first define (strict) separating constraints of the classes as

$$\forall l \begin{cases} \boldsymbol{w} \cdot \boldsymbol{x}_l + b \geq \gamma & \text{if} \quad y_l = 1 \\ \boldsymbol{w} \cdot \boldsymbol{x}_l + b \leq -\gamma & \text{if} \quad y_l = -1 \, , \end{cases}$$

where $\gamma > 0$ is an arbitrary real, which can be chosen equal to 1, with a possible rescaling of $\boldsymbol{w}$ and b. For convenience sake, we thus rewrite the constraints as

$$\forall l, \ y_l \left(\boldsymbol{w} \cdot \boldsymbol{x}_l + b \right) \geq 1 \, . \tag{3.21}$$

We then define the margin ρ as the distance between the hyperplane $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = 1$ and the hyperplane $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = -1$. The margin can be easily computed as

$$\rho = \frac{2}{\|\boldsymbol{w}\|} \, . \tag{3.22}$$

Hence, the SVM algorithm has to maximize the margin (3.22) while respecting the constraints (3.21). Unfortunately this first version of SVMs (introduced initially by Vapnik and Lerner, 1963) cannot deal with non-linearly separable classes. Two ideas were introduced to overcome this problem. First, similarly to Φ-Machines, SVMs deal with examples $(\Phi(\boldsymbol{x}_l), y_l)$ (given an arbitrary Φ

REF V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24, 1963.

function) instead of dealing with examples $(\boldsymbol{x}_l,\, y_l)$. Hence, we redefine the SVM model as

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w} \cdot \Phi(\boldsymbol{x}) + b\,. \tag{3.23}$$

Moreover, using the formulation suggested by Smith (1968), the constraints (3.21) are relaxed as the following *soft* constraints

$$\begin{gathered} \forall l,\;\, y_l\,(\boldsymbol{w} \cdot \Phi(\boldsymbol{x}_l) + b) \geq 1 - \xi_l \\ \forall l,\;\, \xi_l \geq 0\,, \end{gathered} \tag{3.24}$$

where the *slack variables* ξ_l have to be minimized. This leads to the minimization of

$$J : (\boldsymbol{w},\, b,\, \boldsymbol{\xi}) \mapsto \frac{\mu}{2}\, \|\boldsymbol{w}\|^2 + \frac{1}{L} \sum_{l=1}^{L} \xi_l \tag{3.25}$$

under the constraints (3.24). Note that the *minimization* of the first term corresponds to the *maximization* of the margin (3.22) *in the feature space* generated by Φ. The hyper-parameter μ has to be tuned, and controls the trade-off between the size of the margin, and the sum of margin errors (corresponding to examples which do not respect constraints (3.24)).

About the Notation

Note that in this thesis we are not using the "standard" SVM notation which considers

$$J : (\boldsymbol{w},\, b,\, \boldsymbol{\xi}) \mapsto \frac{1}{2}\, \|\boldsymbol{w}\|^2 + C \sum_{l=1}^{L} \xi_l$$

instead of (3.25). The relation between these two notations is easily given with

$$C = \frac{1}{\mu\, L}\,.$$

Moreover, the Lagrange multipliers $\boldsymbol{\alpha}$ defined below have to be divided by μ to retrieve the standard ones. We prefer the "μ" notation instead of the "C" one, because it unifies MLP and SVM notations and facilitates the establishment of links between these models, as shown in Chapter 8.

REF F. W. Smith. Pattern classifier design by linear programming. *IEEE Transactions on Computers*, C-17(4):367–372, 1968.

Minimization of a Quadratic Function Under Constraints

The minimization of (3.25) under the constraints (3.24) is achieved by using a classical Lagrangian method. We first introduce the Lagrangian

$$L(\boldsymbol{w},\, b,\, \boldsymbol{\xi},\, \boldsymbol{\alpha},\, \boldsymbol{\nu}) = \\ J(\boldsymbol{w},\, b,\, \boldsymbol{\xi}) + \sum_{l=1}^{L} \alpha_l [1 - \xi_l - y_l\,(\boldsymbol{w} \cdot \Phi(\boldsymbol{x}_l) + b)] - \sum_{l=1}^{L} \nu_l\, \xi_l\,, \quad (3.26)$$

with

$$\forall l \qquad \begin{array}{c} \alpha_l \geq 0 \\ \nu_l \geq 0\,. \end{array} \quad (3.27)$$

The variables $\boldsymbol{\alpha}$ and $\boldsymbol{\nu}$ are called *Lagrange multipliers*. As (3.25) is a *convex* minimization problem with *convex* constraints (3.24), we know from Lagrangian theory (see Appendix A for an introduction, or Ciarlet, 1990) that if $(\boldsymbol{w}^\star,\, b^\star,\, \boldsymbol{\xi}^\star)$ is a minimum of (3.25) under the constraints (3.24) then there exists a *saddle point* $((\boldsymbol{w}^\star,\, b^\star,\, \boldsymbol{\xi}^\star),\, (\boldsymbol{\alpha}^\star,\, \boldsymbol{\nu}^\star))$ of the Lagrangian (3.26). In other words, $(\boldsymbol{w}^\star,\, b^\star,\, \boldsymbol{\xi}^\star)$ has to be a *minimum* of

$$(\boldsymbol{w},\, b,\, \boldsymbol{\xi}) \mapsto L(\boldsymbol{w},\, b,\, \boldsymbol{\xi},\, \boldsymbol{\alpha}^\star,\, \boldsymbol{\nu}^\star)\,,$$

and $(\boldsymbol{\alpha}^\star,\, \boldsymbol{\nu}^\star)$ has to be a *maximum* of

$$(\boldsymbol{\alpha},\, \boldsymbol{\nu}) \mapsto L(\boldsymbol{w}^\star,\, b^\star,\, \boldsymbol{\xi}^\star,\, \boldsymbol{\alpha},\, \boldsymbol{\nu})\,.$$

We thus first look for minima of the Lagrangian with respect to $(\boldsymbol{w},\, b,\, \boldsymbol{\xi})$:

$$\forall l \qquad \begin{cases} \dfrac{\partial L}{\partial \boldsymbol{w}} = 0 \;\; \Leftrightarrow \;\; \boldsymbol{w} = \dfrac{1}{\mu} \displaystyle\sum_{l=1}^{L} \alpha_l\, y_l\, \Phi(\boldsymbol{x}_l) \\ \dfrac{\partial L}{\partial b} = 0 \;\; \Leftrightarrow \;\; \displaystyle\sum_{l=1}^{L} \alpha_l\, y_l = 0 \\ \dfrac{\partial L}{\partial \boldsymbol{\xi}} = 0 \;\; \Leftrightarrow \;\; \dfrac{1}{L} - \alpha_l - \nu_l = 0\,. \end{cases} \quad (3.28)$$

Note also that looking for maxima with respect to $(\boldsymbol{\alpha},\, \boldsymbol{\nu})$ gives us

$$\forall l \qquad \begin{cases} \alpha_l\, [1 - \xi_l - y_l\,(\boldsymbol{w} \cdot \Phi(\boldsymbol{x}_l) + b)] \;=\; 0 \\ \nu_l\, \xi_l \;=\; 0\,. \end{cases} \quad (3.29)$$

REF P. G. Ciarlet. *Introduction à l'analyse numérique matricielle et à l'optimisation.* Masson, 1990.

Equations (3.27), (3.28) and (3.29) are often referred as the *Karush-Kuhn and Tucker* (KKT) *conditions* (Introduced by Karush, 1939 and Kuhn and Tucker, 1951. See Appendix A for details). Substituting equation (3.28) into the Lagrangian (3.26) leads to the *maximization* of

$$\boldsymbol{\alpha} \mapsto \sum_{l=1}^{L} \alpha_l - \frac{1}{2\,\mu} \sum_{l=1}^{L} \sum_{m=1}^{L} y_l\, y_m\, \alpha_l\, \alpha_m\, \Phi(\boldsymbol{x}_l) \cdot \Phi(\boldsymbol{x}_m)\,,$$

under the constraints (3.28).

The SVM Problem

The SVM problem is therefore equivalent to the *minimization* of

$$\boldsymbol{\alpha} \mapsto \frac{1}{2\,\mu} \sum_{l=1}^{L} \sum_{m=1}^{L} y_l\, y_m\, \alpha_l\, \alpha_m\, \Phi(\boldsymbol{x}_l) \cdot \Phi(\boldsymbol{x}_m) - \sum_{l=1}^{L} \alpha_l\,, \tag{3.30}$$

under the constraints

$$\begin{aligned} &\sum_{l=1}^{L} \alpha_l\, y_l = 0 \\ \forall l \qquad & 0 \leq \alpha_l \leq \frac{1}{L}\,. \end{aligned} \tag{3.31}$$

The weight $\boldsymbol{w}$ is then given by

$$\boldsymbol{w} = \frac{1}{\mu} \sum_{l=1}^{L} y_l\, \alpha_l\, \Phi(\boldsymbol{x}_l)\,, \tag{3.32}$$

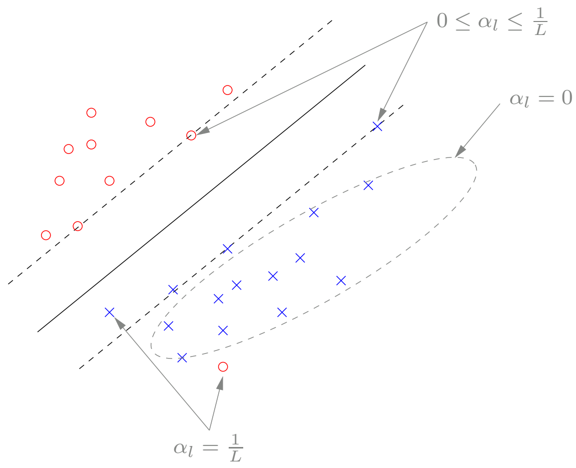
and b is given using conditions (3.29). Indeed, after considering all KKT conditions, it is easy to see that if $0 < \alpha_l < 1/L$, then $\xi_l = 0$ and thus

$$1 - y_l\,(\boldsymbol{w} \cdot \Phi(\boldsymbol{x}_l) + b) \;=\; 0\,.$$

Note that the examples $(\boldsymbol{x}_l,\, y_l)$ such that the corresponding α_l are non-zero are called *support vectors*, because they are the only training examples necessary to define the separating hyperplane in (3.32). Using KKT conditions, it is also possible to see when an example is a support vector, as shown in Figure 3.5. This happens only if the example is *on* the margin, or if it does not respect the separation conditions (3.21) (in this latter case, the support vector is said to be *at bound*, because in that case $\alpha_l = 1/L$ is the maximum allowed by constraints (3.31)). Hence SVMs are said to perform a *sparse representation* of the training data. Finally, it is worth mentioning that the SVM solution is

REF W. Karush. *Minima of functions of several variables with inequalities as side constraints*. Department of Mathematics. University of Chicago, 1939.

REF H. W. Kuhn and A. W. Tucker. Non-linear programming. In *Proceedings* 2^{nd} *Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, Berkeley, 1951. University of California Press.



▲ **Figure 3.5.** *Description of the separating hyperplane in the feature space, for classification SVMs. Support vectors are the examples corresponding to non-zero α_l.*

usually unique (due to the convex minimization problem), except in some rare pathological cases, described in Burges and Crisp (2001).

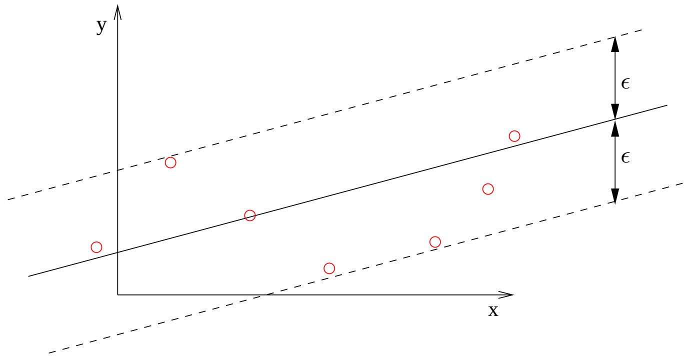
Optimization

The minimization of the quadratic problem (3.30) under constraints (3.31) is usually achieved with a kind of gradient descent technique under constraints, such as the algorithm proposed by Platt (1999a), or the improved algorithm proposed by Joachims (1999). We will talk about this optimization problem in Chapter 5.

REF C. Burges and D. Crisp. Uniqueness of the SVM solution. In S. A. Solla, T. K. Leen, and K. R. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12, pages 223–229. MIT Press, 2001.

REF J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. The MIT Press, 1999a.

REF T. Joachims. Making large-scale support vector machine learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. The MIT Press, 1999.



▲ **Figure 3.6.** *SVMs in regression try to fit the training set (red circles) into a tube of width $2\,\epsilon$.*

Regression

An extension of SVMs for regression problems has been proposed by Vapnik (1995). It considers real target values $y_l \in \mathbb{R}$. The idea is to find a hyperplane

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b\,,$$

such that the norm $\|\boldsymbol{w}\|$ is minimized while the training examples $(\boldsymbol{x}_l,\, y_l)$ lie inside a "tube" of width $2\,\epsilon$ around this hyperplane, for a given $\epsilon > 0$, as depicted in Figure 3.6. More formally, we want

$$\forall l \qquad \begin{array}{c} (\boldsymbol{w} \cdot \boldsymbol{x}_l + b) - y_l \leq \epsilon \\ y_l - (\boldsymbol{w} \cdot \boldsymbol{x}_l + b) \leq \epsilon\,. \end{array} \tag{3.33}$$

The minimization of the norm $\|\boldsymbol{w}\|$ is justified as a regularization parameter, following the idea of ridge regression introduced by Hoerl and Kennard (1970). The same techniques used for SVMs in classification are applied for the regression case: instead of dealing with examples $(\boldsymbol{x}_l,\, y_l)$, we consider examples $(\Phi(\boldsymbol{x}_l),\, y_l)$ for an arbitrarily chosen Φ function. Moreover, as it is usually not possible to respect the constraints (3.33) we relax them as

$$\forall l \qquad \begin{cases} (\boldsymbol{w} \cdot \Phi(\boldsymbol{x}_l) + b) - y_l \leq \epsilon + \xi_l \\ y_l - (\boldsymbol{w} \cdot \Phi(\boldsymbol{x}_l) + b) \leq \epsilon + \xi_l^*\,, \end{cases} \tag{3.34}$$

REF V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, second edition, 1995.

REF A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67, 1970.

where the non-negative slack variables $\boldsymbol{\xi} \in \mathbb{R}_+^L$ and $\boldsymbol{\xi}^* \in \mathbb{R}_+^L$ have to be minimized. This leads to the minimization of

$$(\boldsymbol{w},\, b,\, \boldsymbol{\xi},\, \boldsymbol{\xi}^*) \mapsto \frac{\mu}{2}\,\|\boldsymbol{w}\|^2 + \frac{1}{L}\sum_{l=1}^{L}(\xi_l + \xi_l^*)\,,$$

under the constraints (3.34).

The SVM Regression Problem

As for the classification case, introducing Lagrangian variables $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^*$ corresponding to constraints (3.34), leads us to the minimization of

$$\begin{aligned}(\boldsymbol{\alpha},\, \boldsymbol{\alpha}^*) \mapsto \frac{1}{2\,\mu}\sum_{l=1}^{L}\sum_{m=1}^{L}(\alpha_l^* - \alpha_l)\,(\alpha_m^* - \alpha_m)\,\Phi(\boldsymbol{x_l})\cdot\Phi(\boldsymbol{x_m})&\\ - \sum_{l=1}^{L} y_l\,(\alpha_l^* - \alpha_l) + \epsilon\sum_{l=1}^{L}(\alpha_l^* + \alpha_l)\,,&\end{aligned} \quad (3.35)$$

with $\boldsymbol{\alpha} \in \mathbb{R}^L$, $\boldsymbol{\alpha}^* \in \mathbb{R}^L$ and under the constraints

$$\begin{aligned}\sum_{l=1}^{L}(\alpha_l - \alpha_l^*) = 0&\\ \forall l \qquad 0 \le \alpha_l,\, \alpha_l^* \le \frac{1}{L}\,.&\end{aligned} \quad (3.36)$$

The weight $\boldsymbol{w}$ is then obtained with

$$\boldsymbol{w} = \frac{1}{\mu}\sum_{l=1}^{L}(\alpha_l^* - \alpha_l)\,\Phi(\boldsymbol{x}_l)\,,$$

and b is given using the equations

$$\begin{aligned}(\boldsymbol{w}\cdot\Phi(\boldsymbol{x}_l) + b) - y_l - \epsilon = 0 &\quad \text{when } 0 < \alpha_l < \frac{1}{L}\\ y_l - (\boldsymbol{w}\cdot\Phi(\boldsymbol{x}_l) + b) - \epsilon = 0 &\quad \text{when } 0 < \alpha_l^* < \frac{1}{L}\,.\end{aligned}$$

The SVM regression problem has similar properties to the classification one. In particular, it is easy to see with KKT conditions that for all examples $(\boldsymbol{x}_l,\, y_l)$ the corresponding Lagrange variables satisfy $\alpha_l\,\alpha_l^* = 0$. Therefore, all examples which have one non zero Lagrange variable are called "support vectors" as well. Once again, it appears that SVMs are performing a sparse representation of the training set. As for classification, the optimization of such a quadratic problem can be achieved by a gradient descent algorithm under constraints, such as the algorithm proposed by Smola and Schölkopf (1998) and the one proposed by Flake and Lawrence (2002). We will discuss this in Chapter 5.

Kernels

A *kernel* is function of two variables which defines an inner product in a (possibly unknown) feature space. More precisely, if $k(\cdot,\,\cdot)$ is a kernel, there exists a function Φ such that

$$\forall \boldsymbol{x}_1,\, \boldsymbol{x}_2 \qquad k(\boldsymbol{x}_1,\, \boldsymbol{x}_2) = \Phi(\boldsymbol{x}_1) \cdot \Phi(\boldsymbol{x}_2)\,. \tag{3.37}$$

Kernel functions were introduced a long time ago by Mercer (1909), who proposed Theorem 3.3, which characterizes functions which are kernels.

Theorem 3.3 (Mercer) *Let us consider a compact subset $\mathcal{C}$ of R^n, and a continuous symmetric function in $\boldsymbol{L}^2(\mathcal{C} \times \mathcal{C})$*

$$k : \mathcal{C} \times \mathcal{C} \to \mathbb{R}\,.$$

To guarantee that k has an expansion

$$k(\boldsymbol{x}_1,\, \boldsymbol{x}_2) = \sum_{k=1}^{\infty} a_k\, \Psi_k(\boldsymbol{x}_1) \cdot \Psi_k(\boldsymbol{x}_2)\,,$$

with positive coefficients $a_k > 0$ (which is equivalent, from the Hilbert-Schmidt theory to say that $k(\cdot,\,\cdot)$ describes an inner product in some feature space), it is necessary and sufficient that the condition

$$\int_{\mathcal{C}} \int_{\mathcal{C}} k(\boldsymbol{x}_1,\, \boldsymbol{x}_2)\, g(\boldsymbol{x}_1)\, g(\boldsymbol{x}_2)\, d\boldsymbol{x}_1\, d\boldsymbol{x}_2 \geq 0$$

is valid for all functions $g \in \boldsymbol{L}^2(\mathcal{C})$.

The main interest of kernels is that they *could* replace a computationally expensive inner product $\Phi(\boldsymbol{x}_1) \cdot \Phi(\boldsymbol{x}_2)$ (if Φ maps the input data in a very high dimensional space) by a simpler function evaluation $k(\boldsymbol{x}_1,\, \boldsymbol{x}_2)$.

REF A. Smola and B. Schölkopf. A tutorial on support vector regression. Technical Report NeuroCOLT NC-TR-98-030, Royal Holloway College, University of London, 1998.

REF G. W. Flake and S. Lawrence. Efficient SVM regression training with SMO. *Machine Learning*, 46(1–3):271–290, 2002.

REF J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society*, A 209:415–446, 1909.

Examples

The two most used kernels (see Vapnik, 1995) are the Gaussian (or Radial Basis Function, RBF) kernel

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp(-\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2/(2\,\sigma^2))\,, \tag{3.38}$$

where $\sigma \in \mathbb{R}_\star^+$, and the polynomial kernel

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = (1 + \boldsymbol{x}_1 \cdot \boldsymbol{x}_2)^p\,, \tag{3.39}$$

where p is a positive integer. Note that many more kernels exist. For example, techniques to "construct" kernels are proposed by Cristianini and Shawe-Taylor (2000). It is also worth mentioning that the function

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \tanh(a + \boldsymbol{x}_1 \cdot \boldsymbol{x}_2)\,,$$

with $a \in \mathbb{R}$, is often referred as a kernel, which is *wrong*, as demonstrated by Smola et al. (2001).

Kernel Algorithms

Since the very beginning of machine learning, kernels have been used in learning algorithms, as for example in the Potential Function Classifier (Aiserman et al., 1966). Kernels then gained a lot of popularity with their introduction into the SVM algorithm by Boser et al. (1992), who pointed out that inner products $\Phi(\boldsymbol{x}_l) \cdot \Phi(\boldsymbol{x}_m)$ in the SVM problem (3.30) could be replaced by kernels. Moreover, using (3.32) allows us to rewrite the SVM decision function (3.23) as

$$\begin{aligned} f_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \boldsymbol{w} \cdot \Phi(\boldsymbol{x}) + b \\ &= b + \sum_{l=1}^{L} \alpha_l\, y_l\, \Phi(\boldsymbol{x}_l) \cdot \Phi(\boldsymbol{x}) \\ &= b + \sum_{l=1}^{L} \alpha_l\, y_l\, k(\boldsymbol{x}_l, \boldsymbol{x})\,. \end{aligned}$$

REF V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, second edition, 1995.

REF N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines.* Cambridge University Press, 2000.

REF A. J. Smola, Z. L. Óvári, and R. C. Williamson. Regularization with dot-product kernels. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13, pages 308–314. MIT Press, 2001.

REF M. A. Aiserman, E. M. Braverman, and L. I. Rozonoer. Potential functions technique and extrapolation in learning system theory. In *Proceedings of I.F.A.C.*, 1966.

REF B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, 1992. ACM Press.

Thus, it is possible with SVMs to map input data in an arbitrary feature space using kernels, without knowing explicitly the mapping function Φ. After the success of kernels for SVMs, many algorithms were "kernelized", such as the Perceptron (Freund and Schapire, 1999), Principal Component Analysis (Schölkopf et al., 1998), Fisher Discriminant Analysis (Mika et al., 1999), Projection Pursuit (Vincent and Bengio, 2002) and Independent Component Analysis (Bach and Jordan, 2002).

Capacity Control and Regularization

In Chapter 2, we highlighted the importance of controlling the capacity (VC-dimension) of a machine learning algorithm, in order to be able to control its ability to generalize. We also mentioned the existence of ill-posed problems, which can be solved with the introduction of a regularization parameter. We thus present in this section techniques for capacity control and regularization, for the models we presented above.

Perceptrons and MLPs

Capacity

Cover (1965) presented Theorem 3.1 which proves that the capacity of a set of hyperplanes (and thus Perceptrons and their Φ-Machine extension) is related to the dimension of the input data. Thus, at a first glance, it is not possible to tune the capacity of Perceptrons, and if we are dealing with high dimensional data, good generalization cannot be guaranteed, according to Vapnik bounds introduced in Chapter 2. (This is often referred as the *curse of dimensionality*).

REF Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

REF B. Schölkopf, A. Smola, and K. R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

REF S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K. R. Müller. Fisher discriminant analysis with kernels. In Y. H. Hu, J. Larsen, E. Wilson, and S. Douglas, editors, *Neural Networks for Signal Processing IX*, pages 41–48. IEEE, 1999.

REF P. Vincent and Y. Bengio. Kernel matching pursuit. *Machine Learning*, 48(1):165–187, 2002.

REF F. R. Bach and M. I. Jordan. Kernel independent component analysis. *Journal of Machine Learning Research*, 3:1–48, 2002.

REF T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, 14: 326–334, 1965.

It was also demonstrated recently by Karpinski and Macintyre (1995) that the capacity of a set of MLPs with sigmoid transfer functions is approximately quadratic with respect to the number of weights of the model. One way to control the capacity of an MLP is thus tuning the number of hidden units. However, as for the Perceptron, the capacity of MLPs depends on the dimension of the data (because the number of weights depends on it as well). Thus, once again good generalization cannot be guaranteed when dealing with high dimensional data.

Regularization

Even with the control of the number of hidden units, regularization of MLPs (or Perceptrons) is also usually employed in order to reduce over-fitting problems. Two methods were introduced early in the history of MLPs by Plaut et al. (1986) and are still used: either early stopping is performed (that is, the training process is stopped before reaching a local optimum, in order to control the value of the parameters) or regularization terms over the parameters of the model are added in the empirical risk (3.7). Regularization terms (also called *weight decays*) are usually of the form of

$$\frac{\mu}{2} \|\boldsymbol{\theta}\|^2 ,$$

where $\boldsymbol{\theta}$ represents the parameters of the MLP, and $\mu \in \mathbb{R}^+$ is the weight decay parameter, which controls the trade-off between the regularization and the minimization of the original empirical risk.

SVMs

As SVMs are a kind a linear classifier in the feature space generated by the Φ function in (3.23), their capacity should be related to the dimension of the feature space, following Theorem 3.1. This is particularly annoying, because as we pointed out before, one idea of SVMs (as Φ-Machines) is to map data to a *higher* dimensional space. In particular, it has been shown (see for example

REF M. Karpinski and A. Macintyre. Polynomial bounds for vc dimension of sigmoidal neural networks. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 200–208, 1995.

REF D. C. Plaut, S. J. Nowlan, and G. E. Hinton. Experiments on learning by back propagation. Technical Report CMU-CS-86-126, Carnegie-Mellon University, Computer Science Department, 1986.

REF C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining Knowledge Discovery*, 2(2):121–167, 1998.

Burges, 1998) that a Gaussian kernel (3.38) maps the data in an *infinite* space and a polynomial kernel (3.39) of degree p maps the data to a space of about

$$C^p_{p+d-1}$$

dimensions (where d is the input dimension of the data). Even though the capacity is controlled by p, it quickly becomes high as p increases. In both cases, the capacity is thus high and the Vapnik bounds given in Chapter 2 cannot guarantee a good generalization. What saves the SVMs in classification is that instead of considering the set of "hyperplanes in the feature space", one should consider the set of "hyperplanes in the feature space *with large margins*", because SVMs also maximize the margin in the feature space. Hence, to justify the use of SVMs, Vapnik (1995) proposed considering the set of γ-*margin hyperplanes*

$$\{\, \boldsymbol{x} \mapsto \hat{\boldsymbol{w}} \cdot \boldsymbol{x} + \hat{b},\ \ \hat{\boldsymbol{w}} \in \mathbb{R}^d,\, \hat{b} \in \mathbb{R},\, \|\hat{\boldsymbol{w}}\| = 1 \,\}\,,$$

which satisfy

$$\forall l \qquad y_l\,(\hat{\boldsymbol{w}} \cdot \boldsymbol{x}_l + \hat{b}) \geq \gamma\,.$$

It is easy to see that every γ-margin hyperplane corresponds to one hyperplane (3.20) under the constraints (3.21), by taking $\gamma = \frac{1}{\|\boldsymbol{w}\|}$. Then Theorem 3.4 can be given.

Theorem 3.4 (Vapnik) *Suppose the input vectors belong to a sphere of radius r in $\mathbb{R}^d$. Then the set of γ-margin separating hyperplanes has a capacity h which satisfies the inequality*

$$h \leq \min\left(\frac{r^2}{\gamma^2},\, d\right) + 1\,.$$

Therefore, large margin separating hyperplanes ensure a low capacity even in a high dimensional space, and thus a good accuracy of the generalization bounds given in Chapter 2. Note that the capacity can be controlled with the hyper-parameter μ in (3.25) which controls the trade-off between the size of the margin, and the number of margin errors of the SVM.

About Other Algorithms

In this thesis we will focus only on the algorithms presented in this chapter. We will consider them under the light of large scale machine learning problems, and we will propose some analysis and some improvements in the framework

REF V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, second edition, 1995.

of this particular context. Obviously, in half a century of machine learning, many other algorithms have been proposed, but we will not consider them in this thesis for evident lack of time and space reasons. We will thus not talk about popular algorithms such as trees (see for instance CART (Breiman et al., 1984), ID3 (Quinlan, 1986), C4.5 (Quinlan and Rivest, 1989)), boosting (see for instance Adaboost (Freund and Schapire, 1996), Marginboost (Mason et al., 1999), Logitboost (Friedman et al., 2000)) which could also have an interest for dealing with large scale problems.

Summary

We presented in these chapter several algorithms which will be employed in this thesis. We introduced the original Perceptron algorithm and its Φ-Machine generalization which were among the first machine learning algorithms for classification problems. We then gave an overview of Multi Layer Perceptrons (MLPs), and the stochastic gradient descent algorithm. MLPs are powerful models which can approximate any function, and are thus able to solve classification and regression problems. We also briefly introduced Mixtures of Experts (MoEs), which combine several arbitrary models. Finally, we presented Support Vector Machines (SVMs) for both classification and regression problems. SVMs in classification add the idea of maximizing the margin between the classes it separates. We also showed how to control the capacity of all these models. It is interesting to highlight the fact that all these models map the training data into a feature space. In the case of classification, they then linearly separate classes in this feature space. In the case of regression, they linearly approximate the data in the feature space. Moreover, the feature space has to be chosen by the user for Φ-Machines and SVMs (but given this choice, the SVM solution is unique), whereas it is trained when using MLPs (however the MLP solution is not unique). Now that the models have been introduced,

---

REF L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA, 1984.

REF J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

REF J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.

REF Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156, 1996.

REF L. Mason, J. Baxter, P. L. Bartlett, and M. Frean. Functional gradient techniques for combining hypotheses. In Smola, Bartlett, Schölkopf, and Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 61–73. MIT Press, 1999.

REF J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 28(2):337–374, 2000.

we will focus in the next chapter on the experimental framework considered in this thesis.

4 Experimental Framework

Most of the work achieved in this thesis focuses on classification over large databases. Instead of introducing these databases each time we have to perform experiments, we describe them in this chapter. We also briefly describe the software we used to carry out these experiments.

Torch

We developed software to ease practical comparisons between machine learning algorithms. The *Torch software library* is publicly available to the scientific community at

`http://www.torch.ch`,

under a free BSD license. It implements most state-of-the-art machine learning algorithms in a unified framework (see Collobert et al., 2002c for more details). This library greatly simplifies the process of extending algorithms or even creating new ones. Moreover, it is written in C++ in an efficient way which allows us to deal with large databases and models. All experiments in this thesis have been performed using Torch.

Experiments Setup

All experiments in this thesis follow the same protocol. The databases are divided into three subsets: a *training* set, a *validation* set and a *test* set. The training set is used for training the parameters of the various machine learning algorithms we consider. The validation set is used to tune the hyper-parameters

REF R. Collobert, S. Bengio, and J. Mariéthoz. Torch: a modular machine learning software library. Technical Report IDIAP-RR 02-46, IDIAP, 2002c.

(that is, the parameters which are external to the algorithm, and which must be chosen by the user) of the algorithms when the goal is to maximize generalization performance. All the hyper-parameters are thus chosen according to the classification error rate found on this subset, following the theory introduced in Chapter 2. However, in some cases we desire the best training performance instead of generalization performance. In these particular cases, the validation set is not used and all hyper-parameters of the model are tuned according to the training performance. Finally, the test set is employed to *compare* generalization performance between several algorithms.

Pre-Processing

For all databases, we normalized each input vector $\boldsymbol{x}$ for all examples $(\boldsymbol{x}, y)$ by subtracting the means from the input variables, and then by normalizing to 1 the variance of each input variable. More precisely, we first computed the mean m_i of the i-th column of input vectors *over the training set*

$$m_i = \frac{1}{L} \sum_{l=1}^{L} x_{l,i} \,.$$

We also computed the standard deviation σ_i where

$$\sigma_i^2 = \frac{1}{L} \sum_{l=1}^{L} x_{l,i}^2 - m_i^2 \,.$$

We then applied the following normalization for each example $(\boldsymbol{x}, y)$ in the database

$$x_i \leftarrow \frac{x_i - m_i}{\sigma_i} \,.$$

This normalization forces the input variables to lie in the same range, and thus to have similar influence for the algorithms we will consider. In the particular case of Multi Layer Perceptrons, it also ensures better conditioning of the empirical risk minimization problem, as highlighted by LeCun et al. (1998).

Forest

The Forest database is currently the largest database available on the UCI website at

`http://kdd.ics.uci.edu/databases/covertype/covertype.data.html`,

REF Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In G.B. Orr and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, pages 9–50. Springer, 1998.

Class	1	2	3	4	5	6	7
# Obs.	211,840	283,301	35,754	2,747	9,493	17,367	20,510

▲ **Table 4.1.** *Number of observations for each of the* 7 *classes of the Forest database.*

Class	win	lost	draw
# Obs.	44473	16635	6449

▲ **Table 4.2.** *Number of observations for each of the* 3 *classes of the Connect-4 database.*

along with a very detailed description. The database contains more than 500,000 observations. Each observation in the database corresponds to a 30×30 square meter cell of forest, represented by a 54 dimensional input vector with 10 continuous variables and 44 binary variables. The task is to determine the type of forest in each cell. This leads to a classification task with 7 classes distributed as given in Table 4.1. In order to have a *balanced* two-class classification problem, we decided to consider instead the problem of classifying class 2 against all the other classes. After shuffling, we kept only 100,000 examples for training, 10,000 for validation and 50,000 for testing.

Connect-4

The Connect-4 database is the second largest database available on the UCI website at

`ftp://ftp.ics.uci.edu/pub/machine-learning-databases/connect-4`.

This database contains all legal positions after eight plays in the game of Connect-4 (a tic-tac-toe like game, in which players drop pieces onto columns and try to get four in a row), in which neither player has won yet, and in which the next move is not forced. The task is to predict if the first player will win or loose, or if the game will be a draw. This leads to a 3-class classification problem, where the distribution of the classes is given in Table 4.2. We transformed this problem into a two-class classification problem, where the goal is to classify class a "win" against the others. The observations in this database are represented by vectors containing 42 discrete attributes which correspond to the 42 playable positions available on the game board. Each of these attributes can take three different values, corresponding to the state of a particular position on the game board: either the first player has taken the position, the second player has taken the position, or the position is not taken yet. As these position states "taken by player 1", "taken by player 2" and "not taken" are not ordered values, it is not appropriate to represent them as integer or real values. We thus transformed each attribute as binary vectors with 3 sub-attributes:

$$\begin{array}{lcl} \text{taken by player 1} & \rightarrow & 0,0,1 \\ \text{taken by player 2} & \rightarrow & 0,1,0 \\ \text{not taken} & \rightarrow & 1,0,0\,. \end{array}$$

This leads to input vectors with 126 binary attributes. We then shuffled the data set and took 50,000 examples for training, 7,500 for validation, and 10,000 for testing.

Summary

In this chapter, we briefly introduced the two main classification databases we will use in this thesis: the Forest and Connect-4 UCI databases. We also introduced the Torch software library used to perform all our experiments. In the next chapter, we will focus on Support Vector Machines (SVMs), a recent and important machine learning algorithm. In particular, having shown in Chapter 3 that SVMs require the minimization of a quadratic problem under constraints, we will now introduce an algorithm able to perform this minimization on large databases.

5 Optimization for SVMs

In Chapter 3 we introduced Support Vector Machines (SVMs), a recent and important machine learning algorithm for classification and regression proposed by Vapnik (1995). We showed that SVM training requires the minimization of a quadratic problem under constraints. Solving this problem using a classical technique such as a conjugate gradient method with projection is known to cost on the order of $O(L^3)$ in time, with respect to the number of training examples L (see Ciarlet, 1990). With a naive implementation, it also has a memory cost on the order of $O(L^2)$ to store the matrix of the quadratic problem. These orders of complexity are not suited for large scale problems. Thus, we propose in this chapter a state-of-the-art technique to train SVMs in an efficient way. This technique is the same as the one proposed by Joachims (1999). Our contribution extends Joachims ideas to the regression SVM problem. Though it may seems obvious, curiously it was not the technique used to train regression SVMs at the time we proposed this extension.

General Form of the Minimization Problem

We already introduced SVMs in Chapter 3, both for classification and regression tasks. It is interesting to see that we can rewrite the SVM problem in a more general form, at least adapted for both cases. Using the same notations

---

REF V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, second edition, 1995.

REF P. G. Ciarlet. *Introduction à l'analyse numérique matricielle et à l'optimisation.* Masson, 1990.

REF T. Joachims. Making large-scale support vector machine learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods.* The MIT Press, 1999.

as in Chapter 3, the decision function in both cases is written as

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w} \cdot \Phi(\boldsymbol{x}) + b\,, \tag{5.1}$$

where b is a real number, $\boldsymbol{w}$ is real vector, and Φ is an arbitrarily chosen function. We can then consider the more general SVM minimization problem

$$(\boldsymbol{w},\, b,\, \boldsymbol{\xi}) \mapsto \frac{\mu}{2}\,\|\boldsymbol{w}\|^2 + \frac{1}{L}\sum_{l=1}^{M} \xi_l\,, \tag{5.2}$$

subject to the constraints

$$\forall l \in \{1 \dots M\}, \quad z_l\,[\,\boldsymbol{w} \cdot \boldsymbol{\phi}_l + b\,] + \gamma_l + \xi_l \geq 0 \tag{5.3}$$

and

$$\boldsymbol{\xi} \geq \mathbf{0}, \tag{5.4}$$

where L is the number of training examples, M is the number of constraints, μ is a regularization parameter as defined in Chapter 3. The constant real vectors $\boldsymbol{z}$, $\boldsymbol{\gamma}$, and $\boldsymbol{\phi}_l$ depend on the task and are defined later in this chapter. Adapting derivations made in Chapter 3, we introduce Lagrange multipliers $\boldsymbol{\alpha} \in \mathbb{R}^M$ and $\boldsymbol{\eta} \in \mathbb{R}^M$ corresponding to constraints (5.3) and (5.4) respectively. As the problem (5.2) is a convex problem with convex constraints, we know (see Appendix A) that finding a minimum $(\boldsymbol{w},\, b,\, \boldsymbol{\xi})$ of (5.2) is equivalent to finding $(\boldsymbol{w},\, b,\, \boldsymbol{\xi},\, \boldsymbol{\alpha},\, \boldsymbol{\eta})$ which satisfies the KKT conditions rewritten here for convenience:

$$\begin{aligned}
\boldsymbol{w} &= \frac{1}{\mu}\sum_{l=1}^{M} \alpha_l\, z_l\, \boldsymbol{\phi}_l \\
\boldsymbol{\alpha}^{\mathrm{T}}\, \boldsymbol{z} &= 0 \\
\alpha_l\,[z_l\,(\boldsymbol{w} \cdot \boldsymbol{\phi}_l + b) + \gamma_l + \xi_l] &= 0 \quad \forall l \\
\frac{1}{L} - \alpha_l - \eta_l &= 0 \quad \forall l \\
\eta_l\, \xi_l &= 0 \quad \forall l \\
\alpha_l &\geq 0 \quad \forall l \\
\eta_l &\geq 0 \quad \forall l\,.
\end{aligned} \tag{5.5}$$

Using Lagrangian convex theory (see Chapter 3 and Appendix A), we also know that $\boldsymbol{\alpha}$ has to be a minimum of

$$J(\boldsymbol{\alpha}) = \frac{1}{2\,\mu}\,\boldsymbol{\alpha}^{\mathrm{T}}\,\boldsymbol{K}\,\boldsymbol{\alpha} + \boldsymbol{\alpha}^{\mathrm{T}}\,\boldsymbol{\gamma}\,, \tag{5.6}$$

under the constraints

$$\begin{aligned}
\boldsymbol{\alpha}^{\mathrm{T}}\,\boldsymbol{z} &= 0 \\
\boldsymbol{\alpha}^{min} \leq \boldsymbol{\alpha} &\leq \boldsymbol{\alpha}^{max}\,,
\end{aligned} \tag{5.7}$$

where $\boldsymbol{\alpha}^{min} = \mathbf{0}$, $\boldsymbol{\alpha}^{max} = \frac{\mathbf{1}}{\boldsymbol{L}}$ and the matrix $\boldsymbol{K} \in \mathbb{R}^{M \times M}$ is defined with

$$K_{lm} = z_l \, z_m \, \boldsymbol{\phi}_l \cdot \boldsymbol{\phi}_m \,. \tag{5.8}$$

Thus, after finding the minimum $\boldsymbol{\alpha}$ of problem (5.6), *we know* that there exists a couple $(w\, b)$ satisfying the KKT conditions (5.5) with this $\boldsymbol{\alpha}$. In particular, the weight $\boldsymbol{w}$ of the model is directly given by KKT conditions. The bias b is found by noticing that for any l such that $\alpha_l^{min} < \alpha_l < \alpha_l^{max}$, $\xi_l = 0$ and the following equation holds

$$z_l \, (\boldsymbol{w} \cdot \boldsymbol{\phi}_l + b) + \gamma_l = 0 \,. \tag{5.9}$$

Our main concern in this chapter is to find a minimum of (5.6) under constraints (5.7) in an efficient way.

Classification and Regression Tasks

We now define the constants of the general problem (5.2) for the specific case of classification and regression tasks, according to derivations given in Chapter 3. For classification tasks, it is obvious that the number of constraints M is equal to the number of examples L. We have

$$\forall l \in \{1 \dots L\} \quad \boldsymbol{\phi}_l = \Phi(\boldsymbol{x}_l) \,,$$

where the function Φ is given by our model (5.1). Vectors $\boldsymbol{z}$ and $\boldsymbol{\gamma}$ belong to $\mathbb{R}^L$ and are defined as

$$\forall l \in \{1 \dots L\} \qquad \begin{cases} z_l = y_l \\ \gamma_l = -1 \,. \end{cases}$$

For regression tasks, the number of constraints M is equal to $2L$. If we first perform the substitution

$$\boldsymbol{\alpha} \leftarrow (\boldsymbol{\alpha}, \, \boldsymbol{\alpha}^*) \tag{5.10}$$

and if we define

$$\forall l \in \{1 \dots 2L\}, \quad \boldsymbol{\phi}_l = \begin{cases} \Phi(\boldsymbol{x}_l) & \text{if } l \leq L \\ \Phi(\boldsymbol{x}_{l-L}) & \text{if } l > L \,, \end{cases}$$

then the SVM regression problem is equivalent to the minimization of (5.6), where $\boldsymbol{z} \in \mathbb{R}^{2L}$ is defined as

$$\forall l \in \{1 \dots 2L\} \qquad z_l = \begin{cases} -1 & \text{if} \quad l \leq L \\ \;\;\,1 & \text{if} \quad l > L \,, \end{cases}$$

and $\boldsymbol{\gamma} \in \mathbb{R}^{2L}$ is given by

$$\forall l \in \{1 \dots 2L\} \qquad \gamma_l = \begin{cases} \epsilon + y_l & \text{if} \quad l \leq L \\ \epsilon - y_l & \text{if} \quad l > L \,. \end{cases}$$

A Computationally Expensive Problem

Solving the minimization problem (5.6) under constraints (5.7) needs resources on the order of $O(M^3)$ in time and on the order of $O(M^2)$ in memory with a classical technique such as a conjugate gradient method with projection (see Ciarlet, 1990) and a naive implementation. In this chapter we propose a method to efficiently solve any kind of problem which is derived from a minimization of form (5.6) (and thus classification and regression) using a *decomposition algorithm* very similar to the one proposed by Joachims (1999) in the context of classification problems.

The Decomposition Algorithm

As in the classification algorithm proposed by Joachims (1999), which was based on a idea from Osuna et al. (1997), our algorithm is based on a *loop* subdivided into the following four steps, which are detailed later in the following sub-sections:

1. Select q variables α_l as the new working set, defined with $l \in \mathcal{W}$.

2. Fix the other variables $\alpha_l,\ l \in \mathcal{F}$ to their current values and solve the problem (5.6) with respect to $\alpha_l,\ l \in \mathcal{W}$.

3. Compute the new bias b.

4. Search for variables α_l whose values have been at α_l^{min} or α_l^{max} (corresponding to constraints (5.7)) for a long time and that will probably not change anymore. This optional step is the *shrinking* phase, as these variables are removed from the problem.

5. Test whether the optimization is finished; if not, we continue the loop by returning to the first step.

REF P. G. Ciarlet. *Introduction à l'analyse numérique matricielle et à l'optimisation*. Masson, 1990.

REF T. Joachims. Making large-scale support vector machine learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. The MIT Press, 1999.

REF T. Joachims. Making large-scale support vector machine learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. The MIT Press, 1999.

REF E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In J. Principe, L. Giles, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII - Proceedings of the 1997 IEEE Workshop*, pages 276–285. IEEE Press, New York, 1997.

Selection of a New Working Set

We propose to select a new set of q variables such that the overall criterion will be optimized. In order to select such a working set, we use the same idea as Joachims (1999): we simply search for the optimal gradient descent direction $\boldsymbol{p}$ which is *feasible* (such that there exists a $\zeta \in \mathbb{R}^+$ such that $\boldsymbol{\alpha} + \zeta\, \boldsymbol{p}$ satisfies constraints (5.7)) and which has only q non-null components. The indices corresponding to these components are chosen to be the new working set $\mathcal{W}$. Thus we need to minimize:

$$\boldsymbol{p} \mapsto \left(\frac{\partial J}{\partial \boldsymbol{\alpha}}\right)^{\mathrm{T}} \boldsymbol{p}\,, \tag{5.11}$$

subject to the constraints

$$\begin{aligned} \boldsymbol{z}^{\mathrm{T}}\boldsymbol{p} &= 0 \\ p_l \geq 0 \ \ &\text{for } l \text{ such that } \alpha_l = \alpha_l^{min} \\ p_l \leq 0 \ \ &\text{for } l \text{ such that } \alpha_l = \alpha_l^{max}\,, \end{aligned} \tag{5.12}$$

and

$$-\mathbf{1} \leq \boldsymbol{p} \leq \mathbf{1} \tag{5.13}$$

$$\mathrm{card}\{\, p_l \ / \ p_l \neq 0 \,\} = q\,. \tag{5.14}$$

Since we are searching for an optimal descent direction (*i.e.* where the scalar product with the gradient is the smallest), we want to minimize (5.11), where the first derivative of J is easily given by

$$\frac{\partial J}{\partial \boldsymbol{\alpha}} = \frac{1}{\mu}\, \boldsymbol{K}\boldsymbol{\alpha} + \boldsymbol{\gamma}\,, \tag{5.15}$$

according to the definition of J given in (5.6). Conditions (5.12) are necessary to ensure the feasibility of the obtained direction. Condition (5.13) ensures that the problem has a solution (because a direction is still the same if we multiply it by an arbitrary scalar). Finally, condition (5.14) is imposed because we are searching for a direction with only q non-null components. Substituting

$$p_l \leftarrow p_l\, z_l\,,$$

and considering (5.15), the problem is equivalent to minimizing

$$\boldsymbol{p} \mapsto \sum_{l=1}^{L} \left[z_l \left(\frac{1}{\mu}\, \boldsymbol{K}\boldsymbol{\alpha} + \boldsymbol{\gamma} \right)_l \right] p_l \tag{5.16}$$

under constraints (5.13), (5.14) and

$$\mathbf{1}^{\mathrm{T}}\boldsymbol{p} = 0 \tag{5.17}$$

$$z_l\, p_l \geq 0 \ \ \text{for } l \text{ such that } \alpha_l = \alpha_l^{min} \tag{5.18}$$

$$z_l\, p_l \leq 0 \ \ \text{for } l \text{ such that } \alpha_l = \alpha_l^{max}\,. \tag{5.19}$$

This is a kind of linear programming problem, and it is known (see for example Ciarlet, 1990) that there exists a solution where the q non-zero coordinates of $\boldsymbol{p}$ are extreme points of the polyhedron defined by (5.17) and (5.13). If we consider only even values of q, then the non-zero coordinates are equal to ± 1 and (5.17) ensures that the number of positive direction coordinates $p_l = 1$ is equal to the number of negative direction coordinates $p_l = -1$. If we now introduce a bijection φ from $\{1 \dots L\}$ into itself, such that the terms

$$z_{\varphi(l)} \left[\frac{1}{\mu} \boldsymbol{K\alpha} + \boldsymbol{\gamma} \right]_{\varphi(l)}$$

in the sum (5.16) are sorted in decreasing order, then for the $q/2$ first indices $\varphi(l)$ satisfying

$$\begin{array}{l} \text{if } z_{\varphi(l)} \leq 0 \;\; \text{then } \alpha^{min}_{\varphi(l)} \leq \alpha_{\varphi(l)} < \alpha^{max}_{\varphi(l)} \\ \text{if } z_{\varphi(l)} \geq 0 \;\; \text{then } \alpha^{min}_{\varphi(l)} < \alpha_{\varphi(l)} \leq \alpha^{max}_{\varphi(l)} \,, \end{array}$$

the direction coordinates must be negative ($p_{\varphi(l)} = -1$), and for the $q/2$ last indices $\varphi(l)$ satisfying

$$\begin{array}{l} \text{if } z_{\varphi(l)} \geq 0 \;\; \text{then } \alpha^{min}_{\varphi(l)} \leq \alpha_{\varphi(l)} < \alpha^{max}_{\varphi(l)} \\ \text{if } z_{\varphi(l)} \leq 0 \;\; \text{then } \alpha^{min}_{\varphi(l)} < \alpha_{\varphi(l)} \leq \alpha^{max}_{\varphi(l)} \,, \end{array}$$

the direction coordinates must be positive ($p_{\varphi(l)} = 1$), according to (5.18) and (5.19). Note that a coordinate $\varphi(l)$ may be selected twice, in which case we would have to lower the value of q, as the coordinates have to be distinct. Other coordinates of $\boldsymbol{p}$ are set to zero. The resulting direction vector $\boldsymbol{p}$ is a solution of the optimal feasible direction problem (5.16). The indices l such that $p_l \neq 0$ obtained with this method correspond to the new working set $\mathcal{W}$.

Solving the Sub-Problem

We want now to solve problem (5.6) taking into account only the variables $\alpha_l, \; l \in \mathcal{W}$. For convenience, we consider the decomposition of all vectors in the following manner:

$$\boldsymbol{\alpha} = (\boldsymbol{\alpha}_{\mathcal{W}}, \, \boldsymbol{\alpha}_{\mathcal{F}}) \,,$$

where the first part corresponds to the variables belonging to the working set $\mathcal{W}$, and the second part corresponds to the variables belonging to the fixed set $\mathcal{F}$. We suppose that the variables are appropriately re-ordered. In the same way, we also decompose the matrix $\boldsymbol{K}$ as follows:

$$\boldsymbol{K} = \begin{pmatrix} \boldsymbol{K}_{\mathcal{W}\mathcal{W}} & \boldsymbol{K}_{\mathcal{W}\mathcal{F}} \\ \boldsymbol{K}_{\mathcal{F}\mathcal{W}} & \boldsymbol{K}_{\mathcal{F}\mathcal{F}} \end{pmatrix} \,.$$

REF P. G. Ciarlet. *Introduction à l'analyse numérique matricielle et à l'optimisation.* Masson, 1990.

Minimizing (5.6) under the constraints (5.7) with respect to variables belonging to the working set $\mathcal{W}$ can then be rewritten as minimizing

$$J_{\mathcal{W}}(\boldsymbol{\alpha}_{\mathcal{W}}) = \frac{1}{2\,\mu}\,\boldsymbol{\alpha}_{\mathcal{W}}^{\mathrm{T}}\,\boldsymbol{K}_{\mathcal{W}\mathcal{W}}\,\boldsymbol{\alpha}_{\mathcal{W}} + \boldsymbol{\alpha}_{\mathcal{W}}^{\mathrm{T}}\left[\frac{1}{\mu}\,\boldsymbol{K}_{\mathcal{W}\mathcal{F}}\,\boldsymbol{\alpha}_{\mathcal{F}} + \boldsymbol{\gamma}_{\mathcal{W}}\right], \tag{5.20}$$

under the constraints

$$\boldsymbol{\alpha}_{\mathcal{W}}^{\mathrm{T}}\,\boldsymbol{z}_{\mathcal{W}} = -\boldsymbol{\alpha}_{\mathcal{F}}^{\mathrm{T}}\,\boldsymbol{z}_{\mathcal{F}} \tag{5.21}$$

$$\boldsymbol{\alpha}_{\mathcal{W}}^{min} \leq \boldsymbol{\alpha}_{\mathcal{W}} \leq \boldsymbol{\alpha}_{\mathcal{W}}^{max}\,, \tag{5.22}$$

by noticing that $\boldsymbol{K}_{\mathcal{W}\mathcal{F}}^{\mathrm{T}} = \boldsymbol{K}_{\mathcal{F}\mathcal{W}}$ and by removing constant terms. This minimization can be achieved using a classical constrained quadratic optimizer, such as a conjugate gradient method with projection or an interior point method (see Fletcher, 1987). However, if the working set size q is equal to 2, the problem can also be solved analytically following the idea found in the Sequential Minimal Optimization (SMO) algorithm introduced by Platt (1999a).

Enhanced SMO Algorithm

The original SMO algorithm is a kind of decomposition algorithm with a working set size equal to 2, but where the working set is chosen randomly or according to some heuristics (see Platt, 1999a). Note that in its random version, the SMO algorithm can be viewed as a stochastic gradient under constraints, where the optimal learning rate is known at each step. Our algorithm is quite better, because we also select the steepest feasible direction at each step. Note however that Keerthi et al. (2001) also proposed some heuristic improvements to the SMO algorithm which is equivalent (in the classification case) to the algorithm proposed here for $q = 2$. Thus, we now consider the case of $q = 2$, and to simplify we index variables in $\mathcal{W}$ by 1 and 2. In particular, we rewrite $\boldsymbol{\alpha}_{\mathcal{W}}$ as

$$\boldsymbol{\alpha}_{\mathcal{W}} = (\alpha_1,\, \alpha_2)\,,$$

and with a similar notation, $\boldsymbol{K}_{\mathcal{W}\mathcal{W}}$ becomes

$$\boldsymbol{K}_{\mathcal{W}\mathcal{W}} = \begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix}. \tag{5.23}$$

REF R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, 1987.

REF J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. The MIT Press, 1999a.

REF S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. Murthy. Improvements to platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13(3):637–649, 2001.

We first want to find the minimum of (5.20) with respect to α_1 along the line defined by (5.21), while ignoring the constraint in (5.22). Note that (5.21) becomes now

$$z_1\,\alpha_1 + z_2\,\alpha_2 = -\boldsymbol{\alpha}_{\mathcal{F}}^{\mathrm{T}}\,\boldsymbol{z}_{\mathcal{F}}\,,$$

which is equivalent to

$$\alpha_2 = -z_2\,\boldsymbol{\alpha}_{\mathcal{F}}^{\mathrm{T}}\,\boldsymbol{z}_{\mathcal{F}} - z_1 z_2\,\alpha_1. \tag{5.24}$$

Thus, finding the minimum of $J_{\mathcal{W}}$ with respect to α_1 along the line of (5.24) corresponds to finding the minimum of

$$\tilde{J}(\alpha_1) = J_{\mathcal{W}}(\psi(\alpha_1))\,,$$

where ψ is defined as

$$\psi\,:\,\alpha_1 \mapsto \begin{pmatrix} \alpha_1 \\ -z_2\,\boldsymbol{\alpha}_{\mathcal{F}}^{\mathrm{T}}\,\boldsymbol{z}_{\mathcal{F}} - z_1 z_2\,\alpha_1 \end{pmatrix}.$$

As it is obvious that $\tilde{J}$ is quadratic with respect to α_1, we obtain with a second order Taylor expansion

$$\tilde{J}(\alpha_1) = \tilde{J}(\alpha_1^o) + \frac{\partial \tilde{J}(\alpha_1^o)}{\partial \alpha_1}\,(\alpha_1 - \alpha_1^o) + \frac{1}{2}\frac{\partial^2 \tilde{J}(\alpha_1^o)}{\partial \alpha_1^2}\,(\alpha_1 - \alpha_1^o)^2\,, \tag{5.25}$$

where $\boldsymbol{\alpha}^o$ is the current value of $\boldsymbol{\alpha}$. The first and second derivatives of $\tilde{J}$ are now found with a chained derivative rule. First we have

$$\frac{\partial \tilde{J}(\alpha_1^o)}{\partial \alpha_1} = \left(\frac{\psi(\alpha_1^o)}{\partial \alpha_1}\right)^{\mathrm{T}} \frac{\partial J_{\mathcal{W}}(\psi(\alpha_1^o))}{\partial \boldsymbol{\alpha}_{\mathcal{W}}}\,,$$

where

$$\frac{\partial \psi}{\partial \alpha_1} = \begin{pmatrix} 1 \\ -z_1 z_2 \end{pmatrix},$$

and

$$\begin{aligned}\frac{\partial J_{\mathcal{W}}}{\partial \boldsymbol{\alpha}_{\mathcal{W}}} &= \frac{1}{\mu}\,\boldsymbol{K}_{\mathcal{W}\mathcal{W}}\,\boldsymbol{\alpha}_{\mathcal{W}} + \frac{1}{\mu}\,\boldsymbol{K}_{\mathcal{W}\mathcal{F}}\,\boldsymbol{\alpha}_{\mathcal{F}} + \boldsymbol{\gamma}_{\mathcal{W}} \\ &= \left[\frac{1}{\mu}\,\boldsymbol{K}\boldsymbol{\alpha} + \boldsymbol{\gamma}\right]_{\mathcal{W}}.\end{aligned}$$

Moreover, as the second derivative of ψ is zero, we can state

$$\frac{\partial^2 \tilde{J}(\alpha_1^o)}{\partial \alpha_1^2} = \left(\frac{\psi(\alpha_1^o)}{\partial \alpha_1}\right)^{\mathrm{T}} \frac{\partial^2 J_{\mathcal{W}}(\psi(\alpha_1^o))}{\partial \boldsymbol{\alpha}_{\mathcal{W}}^2}\,\frac{\psi(\alpha_1^o)}{\partial \alpha_1}\,,$$

where the second derivative of $J_{\mathcal{W}}$ is equal to $\frac{1}{\mu}\,\boldsymbol{K}_{\mathcal{W}\mathcal{W}}$. A bit of algebra on this formula yields

$$\frac{\partial \tilde{J}(\alpha_1^o)}{\partial \alpha_1} = \left[\frac{1}{\mu}\,\boldsymbol{K}\boldsymbol{\alpha}^o + \boldsymbol{\gamma}\right]_1 - z_1 z_2\,\left[\frac{1}{\mu}\,\boldsymbol{K}\boldsymbol{\alpha}^o + \boldsymbol{\gamma}\right]_2\,, \tag{5.26}$$

and

$$\frac{\partial^2 \tilde{J}(\alpha_1^o)}{\partial \alpha_1^2} = \frac{1}{\mu}\,[\,k_{11} - 2z_1z_2\,k_{12} + k_{22}\,]\;.$$

Defining $\rho = \frac{1}{\mu}\,[\,k_{11} - 2z_1z_2\,k_{12} + k_{22}\,]$, (5.25) admits a unique minimum $\alpha_1^\star$ if $\rho > 0$, given by

$$\alpha_1^\star = \alpha_1^o + \frac{z_1z_2\,\left[\frac{1}{\mu}\,\boldsymbol{K}\boldsymbol{\alpha}^o + \boldsymbol{\gamma}\right]_2 - \left[\frac{1}{\mu}\,\boldsymbol{K}\boldsymbol{\alpha}^o + \boldsymbol{\gamma}\right]_1}{\rho}\;. \tag{5.27}$$

Consider once again equation (5.24) which can be rewritten as

$$\alpha_1 = (\alpha_1^o + z_1z_2\,\alpha_2^o) - z_1z_2\,\alpha_2. \tag{5.28}$$

Thus, if $z_1z_2 > 0$ then α_1 is allowed to take values between the bounds L_{α_1} and H_{α_1} where

$$\begin{aligned} L_{\alpha_1} &= \max[\,\alpha_1^{min},\,(\alpha_1^o + \alpha_2^o) - \alpha_2^{max}\,] \\ H_{\alpha_1} &= \min[\,\alpha_1^{max},\,(\alpha_1^o + \alpha_2^o) - \alpha_2^{min}\,]\,. \end{aligned}$$

If $z_1z_2 < 0$, then we have instead

$$\begin{aligned} L_{\alpha_1} &= \max[\,\alpha_1^{min},\,(\alpha_1^o - \alpha_2^o) + \alpha_2^{min}\,] \\ H_{\alpha_1} &= \min[\,\alpha_1^{max},\,(\alpha_1^o - \alpha_2^o) + \alpha_2^{max}\,]\,. \end{aligned}$$

Finally, the solution of our problem (5.20) for $q = 2$ is given as follows:

- If $\rho > 0$, then

  $$\begin{aligned} \alpha_1 &= \max(L_{\alpha_1},\,\min(\alpha_1^\star,\,H_{\alpha_1})) \\ \alpha_2 &= \alpha_2^o + z_1z_2\,(\alpha_1^o - \alpha_1)\,, \end{aligned}$$

  where $\alpha_1^\star$ is given by (5.27), and α_2 is given using the relation (5.28).

- If $\rho = 0$, then (5.25) is a line. The minimum is therefore determined according to the sign s of the slope (5.26):

  $$\alpha_1 = \begin{cases} L_{\alpha_1} & \text{if } s > 0 \\ H_{\alpha_1} & \text{if } s < 0 \end{cases}$$

  Again, α_2 is then given using (5.28) which yields

  $$\alpha_2 = \alpha_2^o + z_1z_2\,(\alpha_1^o - \alpha_1)\,.$$

Note that the case $\rho < 0$ is impossible, as the matrix $\boldsymbol{K}$ is always a matrix of inner-products, according to its definition (5.8). Thus, ρ corresponds to a squared distance between two vectors, and is non-negative.

Computing the Bias

As stated earlier, the bias b of the SVM is easily found with equation (5.9) for l such that

$$\alpha_l^{min} < \alpha_l < \alpha_l^{max} \,. \tag{5.29}$$

However, we can only assume that these equations are valid for the same b only if the training is finished (that is, when we found a minimum $\boldsymbol{\alpha}$ of (5.6) and we know that KKT conditions are verified with this $\boldsymbol{\alpha}$). Thus, in the middle of the training, for each l such that (5.29) is satisfied, one bias b_l exists such that

$$\begin{aligned} b_l &= -[\, z_l\, \gamma_l + \boldsymbol{w} \cdot \boldsymbol{\phi}_l \,] \\ &= -z_l \left[\gamma_l + \frac{1}{\mu} \sum_{m=1}^{M} z_m\, z_l\, \alpha_m\, \boldsymbol{\phi}_m \cdot \boldsymbol{\phi}_l \right] \\ &= -z_l \left[\frac{1}{\mu} \boldsymbol{K}\boldsymbol{\alpha} + \boldsymbol{\gamma} \right]_l \,. \end{aligned} \tag{5.30}$$

We thus could estimate the bias b by taking an average of b_l values found with (5.30), which will be equal to the correct value at the end of the training. More simply, we can just take the estimate

$$b = \frac{1}{2} \left[\max_l b_l + \min_l b_l \right] \,, \tag{5.31}$$

which will also be equal to the correct value at the end of the training.

Shrinking

The idea of *shrinking* is to fix some variables α_l whose values have been equal to the bounds α_l^{min} or α_l^{max} for a long time during training, and that will *hopefully* not change anymore. Note that at the end of training, according to KKT conditions (5.5), if $\alpha_l = \alpha_l^{min} = 0$, then $\xi_l = 0$ and the constraint (5.3) can be rewritten as

$$z_l\, [\, \boldsymbol{w} \cdot \boldsymbol{\phi}_l + b \,] + \gamma_l \geq 0 \,. \tag{5.32}$$

Furthermore, if the inequality (5.32) is *strictly* satisfied, then $\alpha_l = 0$. Moreover, if $\alpha_l = \alpha_l^{max} = \frac{1}{L}$ then $\xi_l \geq 0$ and

$$z_l\, [\, \boldsymbol{w} \cdot \boldsymbol{\phi}_l + b \,] + \gamma_l \leq 0 \,. \tag{5.33}$$

Also, if (5.33) is strictly satisfied, then $\alpha_l = \alpha_l^{max}$. Note that (5.32) and (5.33) can be easily verified by noticing that the following equality holds:

$$\begin{aligned} z_l\, [\, \boldsymbol{w} \cdot \boldsymbol{\phi}_l + b \,] + \gamma_l &= z_l\, b + \gamma_l + \frac{1}{\mu} \sum_{m=1}^{M} z_m\, z_l\, \alpha_m\, \boldsymbol{\phi}_m \cdot \boldsymbol{\phi}_l \\ &= \left[\frac{1}{\mu} \boldsymbol{K}\boldsymbol{\alpha} + \boldsymbol{\gamma} \right]_l + z_l\, b \,, \end{aligned} \tag{5.34}$$

where b is estimated using (5.31).

Therefore, if α_l is equal to α_l^{min} or α_l^{max}, and if (5.32) or (5.33) are strictly satisfied (by more than a positive threshold chosen by the user) during a certain amount of training iterations (again chosen by the user), we can expect that the corresponding variable α_l will stay at its value. We can fix such variables during the rest of the training, and continue to deal only with other variables after this shrinking phase. Note, however, that shrinking is only a heuristic. One can still verify at the end of the training that the KKT conditions are satisfied for all fixed variables, and re-train with them if needed (*unshrinking* phase).

Termination Criterion

We already highlighted that if $\boldsymbol{\alpha}$ is solution of the minimization problem (5.6) then the KKT conditions have to be verified for some $(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\eta})$ and with this $\boldsymbol{\alpha}$. We also pointed out that if the KKT conditions are satisfied for $(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\eta})$ then $\boldsymbol{\alpha}$ is a minimum of (5.6) and $(\boldsymbol{w}, b, \boldsymbol{\xi})$ is a minimum of (5.2). Thus, we could say that the training is finished if the KKT conditions are satisfied "well enough". Since KKT conditions are satisfied if all biases b_l computed with (5.30) are equal, we can consider the training finished when these biases are "sufficiently close". In other words, if

$$\max_l b_l - \min_l b_l$$

is sufficiently small (with respect to a threshold determined by the user), then the training is considered as finished.

Implementation Details

Note that in all computations needed by this decomposition algorithm, the only expensive part is the computation of the gradient of the cost:

$$\frac{\partial J}{\partial \boldsymbol{\alpha}} = \frac{1}{\mu} \boldsymbol{K} \boldsymbol{\alpha} + \boldsymbol{\gamma} \,.$$

Thus, we should keep track of this gradient in a memory table, which requires the storage of M real values. After a variable α_l is updated, we can update this gradient as

$$\forall m \in \{1 \dots M\}, \quad \left(\frac{\partial J}{\partial \boldsymbol{\alpha}}\right)_m \leftarrow \left(\frac{\partial J}{\partial \boldsymbol{\alpha}}\right)_m + \frac{1}{\mu} K_{ml} \left[\alpha_l - \alpha_l^o\right]$$

where α_l^o was the old value of the variable. Note that for this update, we only need the l-th column of the matrix $\boldsymbol{K}$. Thus, this algorithm does not require the full storage of $\boldsymbol{K}$ in memory. It is interesting to implement a *cache* that keeps in memory the columns of $\boldsymbol{K}$ that correspond to *the most used variables* α_l in order to avoid recomputing these columns at each iteration.

Convergence

Keerthi and Gilbert (2002) proved that the decomposition algorithm converges for a working set of size 2, if the shrinking is not used. Furthermore, Lin (2001) has shown the convergence of this algorithm for any value of working set size (but again without shrinking), under the following hypothesis which is true if $\boldsymbol{K}$ is positive definite:

Assumption 5.1 *The matrix $\boldsymbol{K}$ satisfies*

$$\min_{\mathcal{I}}(\min(\text{eig}(\boldsymbol{K}_{\mathcal{I}\mathcal{I}}))) > 0$$

where $\mathcal{I}$ is any subset of $\{1 \dots M\}$ with $|\mathcal{I}| \leq q$, $\boldsymbol{K}_{\mathcal{I}\mathcal{I}}$ is a square sub-matrix of $\boldsymbol{K}$, and $\min(\text{eig}(.))$ *is the smallest eigenvalue of a matrix.*

Note that shrinking is a heuristic, and although using it should speed up the algorithm, proofs of convergence will no longer hold.

The Regression Case

As we highlighted at the beginning of the chapter, the decomposition algorithm we proposed is a generalization of the algorithm for *classification tasks* proposed by Joachims (1999). We thus did not contribute anything for classification tasks. However, as our algorithm is slightly different than other algorithms for the case of a regression task, we focus now on the regression case in this section.

Comparisons With Other Algorithms

Many authors have proposed decomposition algorithms for regression. For instance, Shevade et al. (2000) proposed two modifications of the SMO algorithm

REF S. S. Keerthi and E. G. Gilbert. Convergence of a generalized SMO algorithm for SVM classifier design. *Machine Learning*, 46(1–3):351–360, 2002.

REF C. J. Lin. On the convergence of the decomposition method for support vector machines. *IEEE Transactions on Neural Networks*, 12(6):1288–1298, 2001.

REF T. Joachims. Making large-scale support vector machine learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. The MIT Press, 1999.

REF S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R. Murthy. Improvements to the SMO algorithm for SVM regression. *IEEE Transaction on Neural Networks*, 11(5):

from Platt (1999a) for regression, based on work from a previous paper on classification problems from the same team (see Keerthi et al., 2001). Laskov (2000) proposed also a decomposition method for regression problems which is very similar to the second modification proposed by Shevade et al. (2000). In fact, Laskov's method for a sub-problem of size 2 uses the same selection algorithm as well as the same termination criterion as Shevade et al.

Their method for selecting the working set is *very* similar to the one we show in this chapter, but while we propose to select variables α_l independently of their counterparts α_l^* (with notations of Chapter 3, hidden in our general formalization (5.6) through the substitution (5.10)), they propose to select simultaneously *pairs* of variables (α_l, α_l^*). Even if this seems to be a small difference, let us note that for the optimal solution as well as during the optimization process $\alpha_l \, \alpha_l^* = 0 \; \forall l$, in the case of algorithms such as the one we proposed here, as proven by Lin (2001). Thus, one of the two variables α_l or α_l^* is always equal to 0, and choosing the α_l and α_l^* independently can help to quickly eliminate many variables, thanks to the *shrinking* phase, as verified in practice later in the chapter. This has, of course, a direct impact on the speed of our algorithm: working with pairs of variables would force the algorithm to do many computations with null variables until the end of the optimization process.

Smola and Schölkopf (1998) also proposed earlier to use a decomposition algorithm for regression based on SMO, using an analytical solution for the sub-problems, but again they proposed to select two pairs of variables (two α_l and their corresponding α_l^*) instead of two variables as we propose in this paper.

Finally, Flake and Lawrence (2002) proposed a modification of SMO for re-

---

1188–1183, 2000.

REF J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. The MIT Press, 1999a.

REF S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. Murthy. Improvements to platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13(3):637–649, 2001.

REF P. Laskov. An improved decomposition algorithm for regression support vector machines. In S. A. Solla, T. K. Leen, and K. R. Müller, editors, *Advances in Neural Information Processing Systems 12*. The MIT Press, 2000.

REF C. J. Lin. On the convergence of the decomposition method for support vector machines. *IEEE Transactions on Neural Networks*, 12(6):1288–1298, 2001.

REF A. Smola and B. Schölkopf. A tutorial on support vector regression. Technical Report NeuroCOLT NC-TR-98-030, Royal Holloway College, University of London, 1998.

REF G. W. Flake and S. Lawrence. Efficient SVM regression training with SMO. *Machine Learning*, 46(1–3):271–290, 2002.

gression that uses the heuristics proposed by Platt (1999a) and those from Smola and Schölkopf (1998), but their modification uses a new variable $\lambda_l = \alpha_l - \alpha_l^*$. Once again, this forces the use of pairs $(\alpha_l,\, \alpha_l^*)$ during the computations.

The originality of our algorithm is thus to select *independently* the variables α_l and α_l^*, which has the side effect of efficiently adapting the shrinking step proposed in classification by Joachims (1999) (it is indeed more difficult to think of an efficient shrinking method in the context of pairs of variables). This also helps to simplify the resolution of the sub-problem in the case where $q = 2$. Finally, the following experiments suggest that this idea leads to faster convergence times.

Experimental Results

We compared our SVM implementation for regression problems (SVMTorch, included now in the Torch library introduced in Chapter 4) to the one from Flake and Lawrence (2002) using their publicly available software Nodelib. This is interesting because Nodelib is based on SMO where the variables α_l and α_l^* are selected simultaneously, which is not the case for SVMTorch, as highlighted previously. Note also that Nodelib includes some enhancements compared to SMO which are different from those proposed by Shevade et al. (2000).

Both these algorithms use an internal cache in order to be able to solve large-scale problems. All the experiments presented in this chapter have been done on a Pentium III 750Mhz processor. The parameters of the algorithms were not chosen to obtain the best generalization performance, since the goal was to *compare the speed* of the algorithms. However, we have chosen them in order to obtain reasonable results. Both programs used the same parameters with regard to cache, precision, etc. For Nodelib, the other parameters were set using the default values proposed by the authors.

We compared the programs on four different tasks. As this is the only time

---

REF J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. The MIT Press, 1999a.

REF T. Joachims. Making large-scale support vector machine learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. The MIT Press, 1999.

REF G. W. Flake and S. Lawrence. Efficient SVM regression training with SMO. *Machine Learning*, 46(1–3):271–290, 2002.

REF S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R. Murthy. Improvements to the SMO algorithm for SVM regression. *IEEE Transaction on Neural Networks*, 11(5): 1188–1183, 2000.

	# Train	# Test	Dim
Kin	6192	2000	32
Artificial	20000	2000	100
Forest	25000	10000	54
Sunspots	40000	2500	12

▲ **Table 5.1.** *For each data set, "# Train" is the maximum available number of training examples and "# Test" is the number of test examples. "Dim" is the input dimension.*

in this thesis that we are dealing with regression tasks, we introduce them now:

Kin This data set is available at

`http://www.cs.toronto.edu/~delve/data/kin/desc.html`,

and represents a realistic simulation of the forward dynamics of an 8 link all-revolute robot arm. The task is to predict the distance of the end-effector from a target, given features like joint positions, twist angles, etc.

Sunspots Using a series representing the number of sunspots per day, we created one input/output pair for each day: the yearly average of the year starting the next day had to be predicted using the 12 previous yearly averages.

Artificial This is an easy artificial data set based on **Sunspots**: we create a daily series where each value is the yearly average centered on that day. The task is to predict a value given the 100 previous ones.

Forest This data set is a classification task as presented in Chapter 4 with 7 classes, where only the first 35000 examples were used here. We transformed it into a regression task where the goal was to predict $+1$ for examples of class 2 and -1 for the other examples. Note that since class 2 was over-represented in the data set, this transformation leads to a more balanced problem.

The training set size and the testing set size can be found in Table 5.1. Note that all experiments used a Gaussian kernel instead of an explicit Φ function, as explained in Chapter 3. For the experiments involving SVMTorch, we have tested a version with shrinking but without verifying at the end of the optimization whether all the suppressed variables verified the KKT conditions

	Working set size q				
	2	4	10	50	100
Kin	11	14	16	28	54
Artificial	98	149	190	629	1537
Forest	272	406	462	670	981
Sunspots	7	11	15	45	89

▲ **Table 5.2.** *Training time (in seconds) as a function of the working set size q, with SVMTorch (shrinking activated).*

(SVMTorch), with no shrinking (SVMTorchN), and a version with shrinking and unshrinking at the end of the optimization (SVMTorchU). As will be seen in the results, the first method converges much quicker, with a small negative impact on the generalization performance *in general*.

Optimal Working Set Size

Using the first 10000 examples of each data set (except 6192 for Kin), we trained different models using various values of the working set size q, from 2 to 100. We used a fixed cache size of 100MB and turned on the shrinking. The optimizer used to solve the sub-problems of size $q > 2$ was a conjugate gradient method with projection, written by Bottou (personal communication). Table 5.2 gives the results of these experiments. It is clear that $q = 2$ is always faster than any other value of q. Thus, in the following experiments, we have always used $q = 2$.

Small Data Sets

Let us now compare SVMTorch and Nodelib on small data sets. Only the first 5000 training examples were used here, and the size of the cache was set to 300MB, so that the whole kernel matrix could be kept in memory. This allows us to perform a comparison of the algorithms themselves (and not the implementations), as there is no cache handling. We give the results in Table 5.3, in terms of time, final value of the cost (5.6) and Mean Absolute Error (MAE) on the training and test tests. As can be seen, for all the data sets, SVMTorch is usually many times faster than Nodelib (except for Artificial in the case of SVMTorch without shrinking). Since the whole matrix of the quadratic problem was in memory, handling of the cache had no effect on the speed results. Thus one can conclude that one of the main differences between these algorithms is the selection of the sub-problem, and that selecting the α_l inde-

Data Set	Model	Time	# SV	Final Cost	MAE Train	MAE Test
	SVMTorch	7	936	-173977.85	0.30	0.31
Kin	SVMTorchU	15	936	-173977.85	0.30	0.31
	SVMTorchN	45	941	-173982.65	0.30	0.31
	Nodelib	157	932	-174019.67	0.30	0.31
	SVMTorch	31	342	-13594.01	0.25	0.52
Artificial	SVMTorchU	166	367	-13703.07	0.24	0.51
	SVMTorchN	448	370	-13701.41	0.24	0.51
	Nodelib	231	342	-13707.29	0.24	0.51
	SVMTorch	21	993	-1012.46	0.51	0.80
Forest	SVMTorchU	65	1051	-1030.78	0.41	0.80
	SVMTorchN	110	1058	-1030.43	0.41	0.80
	Nodelib	542	1032	-1031.54	0.41	0.80
	SVMTorch	2	420	-3489571.13	9.65	10.12
Sunspots	SVMTorchU	9	422	-3489630.53	9.65	10.11
	SVMTorchN	38	422	-3489628.27	9.64	10.11
	Nodelib	327	422	-3489630.65	9.64	10.11

▲ **Table 5.3.** *Experiments on small training sets. "SVMTorch" is SVMTorch with shrinking, "SVMTorchN" is SVMTorch without shrinking, "SVMTorchU" is SVMTorch with shrinking and unshrinking at the end, "Time" is the training time in seconds, "# SV" is the number of support vectors, "Final Cost" is the value of (5.6) at the end of the training, "MAE Train" is the Mean Absolute Error (MAE) over the training set, "MAE Test" is the MAE over the test set.*

pendently of the α_l^* is very efficient, as suspected previously. However, Nodelib gave slightly better results in terms of the final cost function value probably due to a slightly difference in the termination criteria, but not in terms of MAE. Note that for problems of this size, shrinking does not cause the performance to deteriorate too much (look at the values of the final cost function as well as the training and test set performance) but significantly speeds up the algorithm.

Large Data Sets

Let us now turn to experiments using large data sets. Table 5.4 shows results using the whole training sets for all data sets, again using a cache size of 300MB. Since the problems are now too big to be kept in memory, the implementation of the cache becomes very important and comparisons of the algorithms used in SVMTorch and Nodelib become more difficult. Nevertheless, it is clear that SVMTorch is always faster, except again on Artificial in the cases with no shrinking or with unshrinking, but the performance on the test sets is similar. However, note that shrinking sometimes leads to very poor results in terms of test set performance, as is the case on Forest. It is thus clear that shrinking should be used with care, *particularly for large data sets*, and the parameter that decides when to eliminate a variable should be tuned carefully before running a series of experiments on the same data set.

Size of the Cache

We also performed experiments to measure the effect of the size of the cache on the training time. Table 5.5 shows the results for different cache sizes, from 10 to 100MB. In these experiments, we used the first 10000 examples of each data set (6192 for the smaller Kin). The only clear conclusion from these experiments is that the higher the size of the cache, the faster SVMTorch is, but the relation is completely problem dependent. Note that in the case of Kin and Sunspots data sets, it seems that 10MB of memory cache is enough to contain all inner-products of examples used during the optimization. Adding more cache is useless and does not improve the training time.

Scaling With Respect to the Size of the Training Set

Finally, we tried to evaluate how SVMTorch (with shrinking) and Nodelib scaled with respect to the size of the training set. In order to be unaffected by the implementation of the cache system, we computed the training time for training sets of sizes 500, 1000, 2000, 3000, 4000, and 5000, so that the

Data Set	Model	Time	# SV	Final Cost	MAE Train	MAE Test
Kin	SVMTorch	11	1140	-212439.78	0.30	0.31
	SVMTorchU	32	1140	-212439.78	0.30	0.31
	SVMTorchN	86	1140	-212439.78	0.30	0.31
	Nodelib	273	1138	-212478.38	0.30	0.31
Artificial	SVMTorch	235	706	-39569.14	0.21	0.34
	SVMTorchU	4394	817	-40025.98	0.20	0.33
	SVMTorchN	9182	824	-40016.55	0.20	0.34
	Nodelib	2653	764	-40043.94	0.20	0.33
Forest	SVMTorch	4573	3019	-56266.94	1.63	1.82
	SVMTorchU	40669	4080	-78297.27	0.40	0.93
	SVMTorchN	79237	4233	-78294.56	0.39	0.93
	Nodelib	87133	4088	-78384.15	0.39	0.93
Sunspots	SVMTorch	67	1771	-11215476.03	8.97	12.72
	SVMTorchU	1290	1822	-11229107.83	8.96	12.59
	SVMTorchN	2606	1820	-11229098.49	8.96	12.59
	Nodelib	24022	1818	-11229124.45	8.96	12.59

▲ **Table 5.4.** *Experiments on large training sets. See Table 5.3 for a description of the fields.*

	Size of the cache (in MB)									
	10	20	30	40	50	60	70	80	90	100
Kin	11	11	11	11	11	11	11	11	11	11
Artificial	359	182	106	100	99	98	98	98	98	98
Forest	869	715	622	519	450	391	355	316	305	272
Sunspots	7	7	7	7	7	7	7	7	7	7

▲ **Table 5.5.** *Training time (in seconds) with respect to the size of the cache (in MB).*

	Kin	Artificial	Forest	Sunspots
Scale SVMTorch	1.81	1.72	1.82	1.85
Scale Nodelib	1.83	1.93	2.09	2.44

▲ **Table 5.6.** *Scaling of SVMTorch and Nodelib for each data set. Results give the slope of the linear regression in the log-log domain of time versus training size.*

whole matrix of the quadratic problem could be kept in memory. Given the results, we performed a linear regression of the log of the time given the log of the training size. Table 5.6 gives the slope of this linear regression for each problem, which gives an idea of how SVMTorch scales: it appears to be slightly better than quadratic, and slightly better than Nodelib.

Contributions

This chapter is a strongly re-arranged version of the following published paper:

R. Collobert and S. Bengio. SVMTorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1:143–160, 2001.

The mathematics in this paper appear significantly different because at the time it was written the links with Joachims (1999) classification algorithm were not completely clear. Thus, for this chapter, all the mathematics were rewritten, in a more general way to be able to deal with classification and regression at the same time. This clearly shows our main contribution is a re-formulation of the regression problem, which allows us to apply directly Joachims's algorithm. The results given in this chapter are a summary of experiments given in the paper. Finally, note that the algorithm we presented in this chapter is implemented in the Torch library (presented in Chapter 3) and is one of the most used widely SVM software packages today.

Conclusion

We have presented in a unified way a decomposition algorithm intended to efficiently solve large-scale SVM problems, for both classification and regression

REF T. Joachims. Making large-scale support vector machine learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. The MIT Press, 1999.

tasks. This algorithm followed the same principles as those used by Joachims (1999) in his classification algorithm. In the regression case, we showed that compared to previously proposed decomposition algorithms, we have proposed an original method to select the variables in the working set. We have also shown how to solve analytically sub-problems of size 2, as it is done in the SMO algorithm, proposed by Platt (1999a). In addition, we highlighted that there exists a convergence proof for our algorithm. Concerning the implementation, an internal cache keeping part of the kernel matrix in memory enables the program to solve large problems without the need to keep quadratic resources in memory and without the need to recompute every kernel evaluation, which leads to a fast overall algorithm. Finally, an experimental comparison with another algorithm has shown significant time improvement for large-scale problems and training time generally scaling slightly less than quadratically with respect to the number of examples.

Even with this state-of-the-art SVM training algorithm, it is often intractable to train SVMs on large databases, due to the scaling factor, which is slightly less than quadratic at best when the memory cache is sufficient. To be able to deal with large databases with such algorithms, we proposed a *divide and conquer* method: break up the training set into small subsets, and apply the SVM algorithm on each subset. The challenge is to find a smart way to partition the training set, while retaining the good generalization performance for which the SVMs are well known. This is the subject of the next chapter.

REF J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. The MIT Press, 1999a.

6 Divide and Conquer

One of the challenges for statistical learning is to be able to deal with large data sets, as in *data mining*. Indeed, many machine learning algorithms such as Support Vector Machines (SVMs) are known to require resources which are at least quadratic in the number of training examples, which raises a serious research challenge if we want to deal with data sets of millions of examples. To overcome this problem, one of the first ideas proposed in this thesis is to apply a *divide and conquer* strategy over already known machine learning algorithms. In other words, we would like to split up training tasks into small pieces. The main concern is finding a way to split the tasks such that the generalization performance is not decreased when compared to the original machine learning algorithm performance. We propose in this chapter 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 subset of the partition. Using a set of generative models with a probabilistic extension allows a representation of the gater such that all pieces of the model are trained locally. We propose experiments on a realistic classification task. With SVM experts, time complexity empirically grows *linearly* with the number of examples, while *generalization* performance can be enhanced. We also justify the use of our hard mixture algorithms by giving the cost functions they minimize. Indeed, we show that in a probabilistic framework these cost functions are upper bounds on negative log-likelihoods of conditional and joint densities of the training data.

Motivation

As more and more data is collected worldwide, the interest in extracting useful information from these data sets with *data mining* algorithms brings several challenges to statistical learning researchers. One of these challenges is the sheer size of the data sets: many learning algorithms require training times that grow too fast with respect to the number of training examples. This is the case with SVMs and Gaussian processes (see Williams and Rasmussen, 1996), both non-parametric learning methods that can be applied to classification, regression, and conditional probability estimation. Both require $O(L^3)$ training time (for L training 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 $O(L^2)$ than $O(L^3)$ as shown by Joachims (1999) and previously in Chapter 5. It would therefore be extremely useful to have general-purpose algorithms which allow the decomposition of a learning problem in such a way as to drastically reduce the training time, so that it grows closer to $O(L)$.

Cheap Computers

Another motivation for the work presented in this chapter 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 a factor depending on the number of available computers with such *loosely-coupled clusters*, to the contrary of a parallelization method such as the one proposed by Estévez et al. (2002) which requires clusters with efficient communication between nodes.

Divide and Conquer

The basic idea proposed in this chapter is to use an iterative divide and conquer strategy to learn a partition of the data such that, ideally (1) the partition is

REF 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.

REF T. Joachims. Making large-scale support vector machine learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. The MIT Press, 1999.

REF P. A. Estévez, Hélène Paugam-Moisy, Didier Puzenat, and Manuel Ugarte. A scalable parallel algorithm for training a hierarchical mixture of neural experts. *Parallel Computing*, 28:861–891, 2002.

“simple”, i.e. it can be learned with good generalization by a model 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 for a test point can be obtained by mixing the predictions of the different experts, in which their predictions are weighted with the output of the gater. Through this process, a Mixture of Experts (MoEs) as presented in Chapter 3 is obtained, but which has not been trained in the usual way (maximum likelihood, mean squared error minimization, etc...). Note that we will focus only on classification tasks in this chapter, but our ideas could easily be extended to other tasks.

Practical Implementations

In this chapter we also propose practical implementations of our mixture when using MLPs and SVMs as experts. Note that the ideas of MLP mixtures (see Jacobs et al., 1991) or of SVM mixtures (see Kwok, 1998) are not new, although previous attempts trained each model on the *whole* training set. We instead advocate mixtures in which each model 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 model, and have experimentally lead to results that are *at least as good as one SVM or one MLP*. 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. Additionally this mixture can be easily parallelized, which could improve again *significantly* the training time.

Standard MoEs

We consider in this chapter a binary classification task, as already defined in Chapter 3. We previously introduced MoEs, as a model $f_{\boldsymbol{\theta}}$ which performs the decomposition into K experts $f_{\boldsymbol{\theta},k}$

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{k=1}^{K} g_{\boldsymbol{\theta},k}(\boldsymbol{x})\, f_{\boldsymbol{\theta},k}(\boldsymbol{x}) \quad \text{with} \quad \sum_{k=1}^{K} g_{\boldsymbol{\theta},k} = 1 \text{ and } g_{\boldsymbol{\theta},k} \geq 0 \ \ \forall k\,, \qquad (6.1)$$

REF R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixture of local experts. *Neural Computation*, 3(1):79–87, 1991.

REF 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.

given an input vector $\boldsymbol{x}$, where the weights of each expert is given by a gater $g_{\boldsymbol{\theta}}(\cdot) = \{g_{\boldsymbol{\theta},1}(\cdot), \ldots, g_{\boldsymbol{\theta},K}(\cdot)\}$. The generic vector $\boldsymbol{\theta}$ refers to the parameters of the model.

It is interesting to consider MoEs in a probabilistic framework, where the goal is to estimate the conditional distribution $P(y|\boldsymbol{x})$ given an example $(\boldsymbol{x}, y)$. In that case, we consider the discrete variable $E(\boldsymbol{x})$ which assigns an example $(\boldsymbol{x}, y)$ to one expert, knowing only $\boldsymbol{x}$. We can thus write the decomposition

$$\mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x}) = \sum_{k=1}^{K} \mathbf{P}_{\boldsymbol{\theta}}(k|\boldsymbol{x})\, \mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x}, k)\,, \tag{6.2}$$

where $\boldsymbol{\theta}$ represents the parameters of the model, and where the conditional distribution $\mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x}, k) = \mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x}, E = k)$ defines the expert k. The gater is also given by the distribution $\mathbf{P}_{\boldsymbol{\theta}}(k|\boldsymbol{x}) = \mathbf{P}_{\boldsymbol{\theta}}(E = k|\boldsymbol{x})$, and probabilistically assigns each example to an expert. Usually this kind of mixture is trained using a *log-likelihood maximization* technique, that is, by minimizing $-\sum_{l=1}^{L} \log \mathbf{P}_{\boldsymbol{\theta}}(y_l|\boldsymbol{x}_l)$ over the training set (An example of this kind of mixture is given by Jordan and Jacobs, 1994).

Note that the probabilistic and non-probabilistic versions are very similar. The mixture (6.2) in a probabilistic framework is indeed a case of the general mixture (6.1) when writing

$$f_{\boldsymbol{\theta},k}(\boldsymbol{x}) = \mathbf{P}_{\boldsymbol{\theta}}(y = 1|\boldsymbol{x}, k) \text{ and } g_{\boldsymbol{\theta},k}(\boldsymbol{x}) = \mathbf{P}_{\boldsymbol{\theta}}(k|\boldsymbol{x})\,. \tag{6.3}$$

In fact, the only difference is that in the probabilistic case we add the constraints $f_{\boldsymbol{\theta},k} \geq 0,\ \forall k$, and the training algorithm *must* maximize conditional probabilities $\mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x})$.

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 to take advantage of such a decomposition to *split up the training task into small pieces* instead. That is the key point of the new models.

Hard Mixture With Global Gater

We propose Algorithm 6.1 to train the mixture (6.1) in a general framework. This algorithm can also be used to train the mixture (6.2) in a probabilistic

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

framework, using the relations in (6.3). The idea is intuitively obvious: one *iterates to discover a good partition* of the training set, which potentially could better represent the structure of the training set. At each iteration, the experts are trained over one of the subsets given by the partition, and then the new partition is re-generated according to the "confidence" the gater has in each expert, for a given example. In a general framework, this corresponds to assigning an example $\boldsymbol{x}$ to the expert corresponding to the largest gater output $g_{\boldsymbol{\theta},k}(\boldsymbol{x})$. In a probabilistic framework, the confidence of the gater is given by $\mathbf{P}_{\boldsymbol{\theta}}(k|\boldsymbol{x})$. Note that we redistribute an example $(\boldsymbol{x}, y)$ to the most appropriate expert *knowing only the input* $\boldsymbol{x}$.

Algorithm 6.1 Training algorithm for a hard mixture with a *global* gater.

1. *Divide* the training set into K random subsets D_k of size near L/K

repeat

  2. Train each expert $f_{\boldsymbol{\theta},k}$ *separately* to minimize a "local criterion" on D_k
  3. Keeping the experts fixed, *train* the mixture (6.1), and thus the non-fixed *gater* $g_{\boldsymbol{\theta}}$, to minimize a "global criterion" on the whole training set
  4. *Reconstruct* K subsets:

  for $l \in \{1..L\}$ **do**

    a. Sort the experts in descending order according to the values $g_{\boldsymbol{\theta},k}(x_l)$

    b. Assign the example to the first expert in the list which has less than $(L/K+1)$ examples in order to ensure a balance between the experts

  end for

until termination criterion

Local and global criteria in Algorithm 6.1 are chosen arbitrary. For example, when using an MLP gater and SVMs (which do not output probabilities) as experts, one could use the SVM criterion for the local criterion, and the Mean Squared Error (MSE) criterion for the global criterion. When using MLPs (which are able to output probabilities) for the gater and the experts, one could use the log-likelihood criterion in both cases. Note that step 2 of this algorithm can be easily implemented in *parallel* as each expert can be trained separately on a different computer. Also note that step 3 can be an approximate minimization that can continue from the solution found at the end of the previous outer loop iteration. In step 4, it is crucial to enforce balance between the experts. This prevents one expert from "capturing" all the training examples. Finally, the termination criterion could be a given number of iterations or an increasing validation error.

Hard Mixture With Local Gater

One possible drawback of the previous model is that the gater must be trained over the *whole* data set, which could create a bottleneck in training time. Thus, we propose, in a *probabilistic framework*, to *break up the gater itself* into submodels (one per expert) that can be trained separately. Indeed, using Bayes' rule, the k-th output $\mathbf{P}_{\boldsymbol{\theta}}(k|\boldsymbol{x})$ of the gater can be rewritten as

$$\mathbf{P}_{\boldsymbol{\theta}}(k|\boldsymbol{x}) = \frac{\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}|k)\,\mathbf{P}_{\boldsymbol{\theta}}(k)}{\sum_{j=1}^{K}\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}|j)\,\mathbf{P}_{\boldsymbol{\theta}}(j)}\,, \tag{6.4}$$

where $\mathbf{P}_{\boldsymbol{\theta}}(k)$ represents the prior probability of an example to be attributed to the k-th expert and $\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}|k)$ represents the set of examples corresponding to the k-th expert. We thus associate to each expert a *generative model* $\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}|k)$ that can be *trained solely* on the subset D_k. Algorithm 6.1 can then be directly applied using (6.4) when the conditional probability $\mathbf{P}_{\boldsymbol{\theta}}(k|\boldsymbol{x})$ is needed. However, we previously highlighted that in Algorithm 6.1 we use only the knowledge of $\boldsymbol{x}$ to re-assign the example $(\boldsymbol{x}, y)$. In the case of a probabilistic framework, we should have considered the target y as well, since it is known during training (even if we did not in this thesis, as the first mixture was originally designed to deal with non-probabilistic models such as SVMs). In the case we consider now (breaking up the gater into generative models) we can apply once again Bayes' rule

$$\mathbf{P}_{\boldsymbol{\theta}}(k|\boldsymbol{x}, y) = \frac{\mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x}, k)\,\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}|k)\,\mathbf{P}_{\boldsymbol{\theta}}(k)}{\sum_{j=1}^{K}\mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x}, j)\,\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}|j)\,\mathbf{P}_{\boldsymbol{\theta}}(j)}\,. \tag{6.5}$$

It is then straightforward to adapt Algorithm 6.1 using the decompositions (6.4) and (6.5), as given in Algorithm 6.2. Note that in each iteration of the outer loop, the training of both the "broken up gater" and the experts can be achieved in parallel. Only the re-assignment of the examples remains non-parallelizable.

Extension to Regression Tasks

Algorithm 6.1 can be applied to regression tasks without any modifications, in the case of non-probabilistic framework. When using a probabilistic framework, one could consider the decomposition of the conditional *expectation*

$$\mathbf{E}_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) = \sum_{k=1}^{K}\mathbf{P}_{\boldsymbol{\theta}}(k|\boldsymbol{x})\,\mathbf{E}_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}, k)\,,$$

Algorithm 6.2 Training algorithm for a hard mixture with a *local* gater.

1. *Divide* the training set into K random subsets D_k of size near L/K

repeat

  2. Train each expert $\mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x}, k)$ *separately* over D_k
  3. Train each local gater $\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}|k)$ *separately* over D_k
  4. Estimate the priors $\mathbf{P}_{\boldsymbol{\theta}}(k)$ using $|D_k|/L$
  5. *Reconstruct* K subsets:

  for $l \in \{1..L\}$ **do**

    a. Sort the experts in descending order according to the posterior $\mathbf{P}_{\boldsymbol{\theta}}(k|\boldsymbol{x}_l, y_l)$ computed using (6.5)

    b. Assign the example to the first expert in the list which has less than $(L/K+1)$ examples in order to ensure a balance between the experts

  end for

until termination criterion

instead of the decomposition of the conditional probability (6.2). Then, Algorithm 6.1 and Algorithm 6.2 can be easily extended to regression by using $\mathbf{E}_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}, k)$ instead of $\mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x}, k)$.

What Criterion is Minimized?

The two algorithms we presented for training a mixture of experts in a *hard* manner are based on an *intuitive* decomposition of the training set through the iterations. We will justify our intuition in this section, in the case of a probabilistic framework.

Local Gater, Probabilistic Framework

Let us first focus on Algorithm 6.2 in a probabilistic framework. We also consider the criterion

$$J_{6.2}(\boldsymbol{\theta},\, \boldsymbol{e}) = -\sum_{l=1}^{L}\sum_{k=1}^{K} e_{lk} \log[\, \mathbf{P}_{\boldsymbol{\theta}}(y_l|\boldsymbol{x}_l, k)\, \mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}_l|k)\, \mathbf{P}_{\boldsymbol{\theta}}(k)\,]\,, \tag{6.6}$$

where $e_{lk} \in \{0, 1\}$ is a binary variable that selects the k-th expert for example l, with the *selection constraints* $\forall l, \sum_k e_{lk} = 1$ and *balancing constraints* $\forall k, \sum_l e_{lk} \approx L/K$. To relate this to Algorithm 6.2, note that we are trying to perform the double minimization

$$\min_{\boldsymbol{\theta}} \min_{\boldsymbol{e}} J_{6.2}(\boldsymbol{\theta},\, \boldsymbol{e})\,.$$

The idea of Algorithm 6.2 is to perform a "coordinate descent" on $J_{6.2}(\boldsymbol{\theta}, \boldsymbol{e})$. In the first stage of each iteration $\boldsymbol{e}$ is fixed and $\boldsymbol{\theta}$ is modified in order to decrease $J_{6.2}(\boldsymbol{\theta}, \boldsymbol{e})$. As the probabilities corresponding to the expert and the gater decouple in (6.6) they can be maximized separately, as in steps 2 and 3 of the algorithm. In the second stage, $\boldsymbol{\theta}$ is fixed and $\boldsymbol{e}$ (the assignment of examples to experts) is modified in order to decrease $J_{6.2}(\boldsymbol{\theta}, \boldsymbol{e})$. Given an example $(\boldsymbol{x}_l, y_l)$, choosing the expert k which yields the greatest $\mathbf{P}_{\boldsymbol{\theta}}(k|\boldsymbol{x}_l, y_l)$ is the best we can do to minimize $J_{6.2}(\boldsymbol{\theta}, \boldsymbol{e})$ according to equation (6.5).

Furthermore, the criterion $J_{6.2}(\boldsymbol{\theta}, \boldsymbol{e})$ is an upper bound on the *joint* negative log-likelihood

$$\mathcal{L}_2(\boldsymbol{\theta}) = -\sum_{l=1}^{L} \log \mathbf{P}_{\boldsymbol{\theta}}(y_l, \boldsymbol{x}_l)\,, \tag{6.7}$$

since with the constraints on $\boldsymbol{e}$ we can derive

$$\begin{aligned}
-\sum_{l=1}^{L} \log[\sum_{k=1}^{K} \mathbf{P}_{\boldsymbol{\theta}}(y_l, \boldsymbol{x}_l, k)\,] &\leq -\sum_{l=1}^{L} \log[\, e_{lk} \sum_{k=1}^{K} \mathbf{P}_{\boldsymbol{\theta}}(y_l, \boldsymbol{x}_l, k)\,] \\
&\leq -\sum_{l=1}^{L}\sum_{k=1}^{K} e_{lk} \log[\,\mathbf{P}_{\boldsymbol{\theta}}(y_l, \boldsymbol{x}_l, k)\,] \\
&\leq -\sum_{l=1}^{L}\sum_{k=1}^{K} e_{lk} \log[\,\mathbf{P}_{\boldsymbol{\theta}}(y_l|\boldsymbol{x}_l, k)\,\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}_l|k)\,\mathbf{P}_{\boldsymbol{\theta}}(k)\,]\,.
\end{aligned}$$

Thus, Algorithm 6.2 minimizes an upper bound on the negative joint log-likelihood $\mathcal{L}_2(\boldsymbol{\theta})$. Note that both cost functions (the negative log-likelihood and $J_{6.2}(\boldsymbol{\theta}, \boldsymbol{e})$) will take similar values when the gater creates a hard partition. In this case, given an example $(\boldsymbol{x}_l, y_l)$, the gater outputs $\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}_l|k)$ in (6.6) will be zero for all k except for one index $k_l^\star$, where $\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}_l|k_l^\star) = 1$. Moreover, as the variables e_{lk} are chosen to minimize (6.6), it insures that $e_{lk_l^\star} = 1$. Thus, the likelihood (6.7) which be rewritten as

$$\begin{aligned}
\mathcal{L}_2(\boldsymbol{\theta}) &= -\sum_{l=1}^{L} \log[\sum_{k=1}^{K} \mathbf{P}_{\boldsymbol{\theta}}(y_l|\boldsymbol{x}_l, k)\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}_l|k)\,\mathbf{P}_{\boldsymbol{\theta}}(k)] \\
&= -\sum_{l=1}^{L} \log[\mathbf{P}_{\boldsymbol{\theta}}(y_l|\boldsymbol{x}_l, k_l^\star)\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}_l|k_l^\star)\,\mathbf{P}_{\boldsymbol{\theta}}(k_l^\star)]\,,
\end{aligned}$$

and the criterion (6.6) are equal.

Global Gater, Probabilistic Framework

In the case of Algorithm 6.1, a similar bound can be derived when the update of the training set partition is achieved using the knowledge of both the inputs and targets of the examples, *i.e.* when using the conditional probability $\mathbf{P}_{\boldsymbol{\theta}}(k|\boldsymbol{x}, y)$

instead of only $\mathbf{P}_{\boldsymbol{\theta}}(k|\boldsymbol{x})$. Under this assumption, and using the same notations as for the bound derived previously for the mixture using a "local gater", we consider the criterion

$$J_{6.1}(\boldsymbol{\theta}, \boldsymbol{e}) = -\sum_{l=1}^{L}\sum_{k=1}^{K} e_{lk} \log[\,\mathbf{P}_{\boldsymbol{\theta}}(y_l|\boldsymbol{x}_l, k)\,\mathbf{P}_{\boldsymbol{\theta}}(k|\boldsymbol{x}_l)\,]\,.$$

Once again, it is easy to see that Algorithm 6.1 is performing a double minimization with respect to $\boldsymbol{\theta}$ and then $\boldsymbol{e}$ at each iteration. In particular, when $\boldsymbol{\theta}$ is fixed, if we consider Bayes' rule

$$\mathbf{P}_{\boldsymbol{\theta}}(k|\boldsymbol{x}, y) = \frac{\mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x}, k)\,\mathbf{P}_{\boldsymbol{\theta}}(k|\boldsymbol{x})}{\sum_{j=1}^{K}\mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x}, j)\,\mathbf{P}_{\boldsymbol{\theta}}(j|\boldsymbol{x})}\,,$$

then assigning an example $(\boldsymbol{x},\, y)$ to the k-expert using $\mathbf{P}_{\boldsymbol{\theta}}(k|\boldsymbol{x}, y)$ is the best we can do to minimize $J_{6.1}(\boldsymbol{\theta},\, \boldsymbol{e})$. It is also straightforward to see that the criterion $J_{6.1}(\boldsymbol{\theta},\, \boldsymbol{e})$ is an upper bound on the *conditional* negative log-likelihood

$$\mathcal{L}_1(\boldsymbol{\theta}) = -\sum_{l=1}^{L} \log \mathbf{P}_{\boldsymbol{\theta}}(y_l|\boldsymbol{x}_l)\,.$$

As in Algorithm 6.2, $J_{6.1}(\boldsymbol{\theta},\, \boldsymbol{e})$ is equal to the likelihood $\mathcal{L}_1(\boldsymbol{\theta})$ if the gater creates a hard partition. Finally, note that in the ideal case where gaters create hard partitions, Algorithm 6.1 is minimizing the *conditional* log-likelihood $\mathcal{L}_1(\boldsymbol{\theta})$, which is a more simple task than the minimization of the *joint* log-likelihood $\mathcal{L}_2(\boldsymbol{\theta})$ achieved by Algorithm 6.2.

Global Gater, General Framework

Unfortunately, a mathematical justification for Algorithm 6.1 could not be found in a non-probabilistic framework. However, as we will see in the experiments, this algorithm seems to work pretty well in practice.

About Other Mixtures

The idea of MoEs is quite old and has given rise to very popular algorithms since their introduction in the machine learning research community by Jacobs et al. (1991). However, in these models the gater and the experts are trained on the *whole* training set instead of subsets (using gradient descent

REF R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixture of local experts. *Neural Computation*, 3(1):79–87, 1991.

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

or the Expectation-Maximization algorithm as proposed by Jordan and Jacobs, 1994). Also, their parameters are usually trained simultaneously. Hence such an algorithm is quite demanding in terms of resources when the training set is large.

In the more recent *Support Vector Mixture* model proposed by Kwok (1998), the experts (which were typically MLPs) are replaced by SVMs and a learning algorithm is given. Once again, the resulting mixture is trained jointly on the whole data set.

Haruno et al. (2001) also proposed an algorithm, in which each expert is associated to a generative model, again trained on the whole training set.

In a *divide-and-conquer* approach introduced by Rida et al. (1999), 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 indeed trains the experts separately on small data sets, like our algorithm. However, there is no notion of iteratively re-assigning the examples to the experts according to gater's predictions of how well each expert performs on each example. Our experiments as shown later in this chapter suggest that this element is essential to the success of the algorithm.

Finally, the *Bayesian Committee Machine* proposed by Tresp (2000) is a technique to partition the data into several subsets, train SVMs or Gaussian Processes on the individual subsets, and use a specific combination scheme based on the covariance of the test data to combine the predictions. This method scales linearly with the size of the training data, but it cannot operate on a single test example. This algorithm also assigns the examples randomly to the experts.

Experiments: Hard Mixture With Global Gater

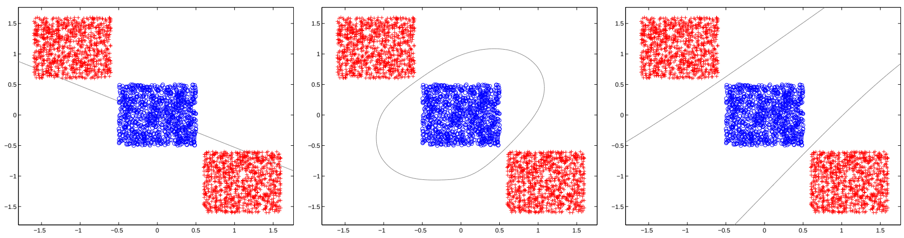
In this section we present two sets of experiments comparing the hard mixture with a "global" gater to other machine learning algorithms in a two-class

REF 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.

REF M. Haruno, D. M. Wolpert, and M. Kawato. MOSAIC model for sensorimotor learning and control. *Neural Computation*, 13(10):2201–2220, 2001.

REF A. Rida, A. Labbi, and C. Pellegrini. Local experts combination through density decomposition. In *Proceedings of UAI'99*. Morgan Kaufmann, 1999.

REF V. Tresp. A bayesian committee machine. *Neural Comp.*, 12(11):2719–2741, 2000.



(a) Linear SVM (b) Gaussian SVM (c) Mixture of two linear SVMs

▲ **Figure 6.1.** *Classification problem where the goal is to separate blue circles and red crosses. Comparison of the decision surfaces obtained by (a) a linear SVM, (b) a Gaussian SVM, and (c) a linear mixture of two linear SVMs.*

classification task framework. As the hard mixture with a global gater was originally designed for non-probabilistic models, we tested it with SVMs experts. Thus, the "local" criterion in Algorithm 6.1 is the SVM criterion. The gater is an MLP, with one hidden layer. The MLP has soft-max outputs (as defined in Chapter 3), to ensure positive values which sum to one. We chose the MSE criterion

$$Q((\boldsymbol{x}, y), f_{\boldsymbol{\theta}}) = \frac{1}{2}(y - \tanh(f_{\boldsymbol{\theta}}(\boldsymbol{x})))^2 ,$$

as the "global" criterion. Note that we added a hyperbolic tangent at the output of the mixture, which helped improve the training and generalization performance. All these experiments have been performed using Athlon 1.2Ghz CPUs.

A Toy Problem

In the first series of experiments, we test the mixture on an artificial toy problem for which we generated 10,000 training examples and 10,000 test examples. The problem has two non-linearly separable classes and has two input dimensions. In Figure 6.1 we show the decision surfaces obtained first by a linear SVM, then by a Gaussian SVM, and finally by the proposed mixture of SVMs. In the last case, the gater was a simple linear function and there were two linear SVMs in the mixture. This artificial problem shows clearly that the algorithm works and is able to combine, even linearly, very simple models in order to produce a non-linear decision surface.

A Large-Scale Realistic Problem: Forest

We performed a series of experiments on part of the *UCI Forest* data set presented in Chapter 4. The hard mixtures had from 10 to 50 SVM experts with Gaussian kernel; the MLP gater had between 25 and 500 hidden units. The same hyper-parameters were selected for all iterations of the algorithm and for all the SVM experts. We compared our models to

- two MLPs (one trained with the MSE criterion and the other one trained with the Cross-Entropy (CE) criterion), where the number of hidden units was selected on the validation set,
- an 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.

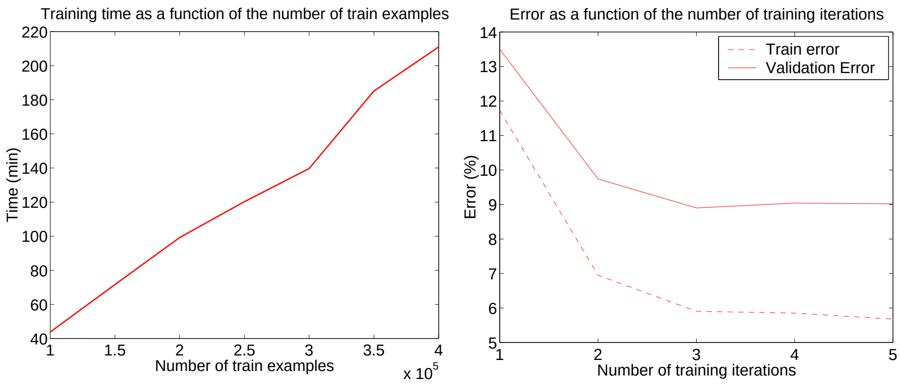
Table 6.1 gives the results of the first series of experiments. Note that for the MLPs the *iteration* column indicates the number of training epochs, whereas for hard mixtures it is the number of outer loop iterations. We selected two variants of the hard SVM mixture, one for best generalization performance and the other one for a good trade-off between generalization performance and training time. The hard SVM mixture is much faster, even on one machine, than the SVM. Since the mixture could easily be parallelized (each expert can be trained separately), we also reported the time it takes to train in parallel, that is with one computer per expert. The hard mixture also outperforms all models in terms of generalization performance. It is important to note that the *choice of the partition* used to train the experts is also *crucial*, since the uniform mixture performed badly.

Finally, note the difference in performance between an MLP trained with the MSE criterion and an MLP trained with the CE criterion. In trying to interpret these results, one can say that the choice of the criterion is crucial and some optimization problems exist when using the MSE criterion with MLPs. We will discuss this further in Chapter 7.

In order to determine how the algorithm scales with respect to the number of examples, we then compared one of the hard mixture of experts (50 experts, 150 hidden units in the gater) on different training set sizes. Figure 6.2a shows the training time (using 50 computers) of the mixture of SVMs with a training set size varying from 100,000 to 400,000 examples. In this case only, we used the extra training examples available in the Forest database. *For this range*

Model	Test Error (%)	Time (minutes) 1 CPU	Time (minutes) Parallel	Iteration
MLP (MSE, 500 hidden units)	14.2	105	–	100
	12.9	2100	–	2000
MLP (CE, 500 hidden units)	11.6	105	–	100
	8.9	1050	–	1000
SVM	10.5	7595	–	–
Uniform SVM mixture (50 experts)	20.3	45	1	1
Hard mixture of SVMs with global gater (50 experts, gater with 150 hidden units)	9.3	120	40	5
Hard mixture of SVMs with global gater (50 experts, gater with 500 hidden units)	**8.1**	**310**	**200**	5

▲ **Table 6.1.** *Comparison of performance between an MLP trained with MSE, an MLP trained with CE, an SVM, a uniform SVM mixture where the gater always output the same value $1/K$ for each expert, and the hard mixture of SVMs with a global gater (Algorithm 6.1). We present the testing error performance, the time it took to train with one computer (1 CPU), and the time it took to train with one computer per expert (Parallel). We also give the number of iterations used to train each model.*



(a) (b)

▲ **Figure 6.2.** *Experiments with a hard mixture of SVMs with a global gater (50 experts, 150 hidden units in the gater) on the Forest database. (a) shows the training time with respect to the number of training examples (from 100,000 to 400,000 examples), when training in parallel with 50 machines. (b) represents training and validation errors of the mixture with respect to the number of training iterations, on 100,000 training examples.*

and this particular data set, the mixture of SVMs appears to scale linearly with respect to the number of examples, and not quadratically as a classical SVM. In fact, 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 two months to solve the same problem with a single SVM. Figure 6.2b shows the evolution of the training and validation errors of the same hard mixture, during 5 iterations of Algorithm 6.1. This convincing evidence that iterative partitioning is essential in order to obtain good performance. It is also clear that the empirical convergence of the outer loop is extremely rapid.

Experiments: Hard Mixture With Local Gater

The set of experiments proposed in this section concerns the training Algorithm 6.2 of the hard mixture with a "local" gater. As standard SVMs do not give probabilities as output, we first present results with MLPs as experts to confirm that the approach works well with gradient-based learning algorithms. Then we present some results using SVMs as experts, with the SVM output being fed to a logistic regressor in order to obtain conditional probabilities, as proposed by Platt (1999b).

REF J. C. Platt. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In Smola, Bartlett, Schölkopf, and Schuurmans, editors,

MLP Experts

The experiments described here are again performed on the Forest data set. We chose *Gaussian Mixtures* for the generative models (that is for the "local" gaters). Gaussian mixtures are able to approximate any kind of distribution by performing the decomposition

$$\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{n=1}^{N_g} \mathbf{P}_{\boldsymbol{\theta}}(n)\, \mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}|n)\,,$$

where N_g is the number of Gaussians, and the n-th Gaussian is defined with

$$\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}|n) = \frac{1}{(2\,\pi)^{d/2}|\boldsymbol{\Sigma}_n|^{1/2}} \exp\left[-\frac{1}{2}\,(\boldsymbol{x} - \boldsymbol{\mu}_n)^{\mathrm{T}} \boldsymbol{\Sigma}_n^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_n) \right]\,.$$

The vector of parameters $\boldsymbol{\theta}$ of the mixture are the weights $\mathbf{P}_{\boldsymbol{\theta}}(n)$ of each Gaussian, the covariance matrices $\boldsymbol{\Sigma}_n$, and the means $\boldsymbol{\mu}_n$. We trained them using the Expectation-Maximization algorithm as proposed by Dempster et al. (1977). A one-hidden-layer MLP trained with a *log-likelihood maximization criterion* was chosen as the expert model. In other words, we maximize for the k-th expert

$$\sum_{l \in D_k} \log \mathbf{P}_{\boldsymbol{\theta}}(y_l|\boldsymbol{x}_l, k)\,. \tag{6.8}$$

Here, we compare the hard mixture in a probabilistic framework 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 on the total training set

$$\sum_{l=1}^{L} \log \mathbf{P}_{\boldsymbol{\theta}}(y_l|\boldsymbol{x}_l)\,. \tag{6.9}$$

Once again, we give results obtained with two MLPs, one trained with the MSE criterion and the other one trained with the CE criterion. The results are summarized in Table 6.2. 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. Training was stopped earlier if training error did not decrease significantly). The hard mixture appears to work reasonably well. On this data set, we can obtain generalization performance similar to an MLP trained with a reasonable time,

---

Advances in Large Margin Classifiers, pages 61–73. MIT Press, 1999b.

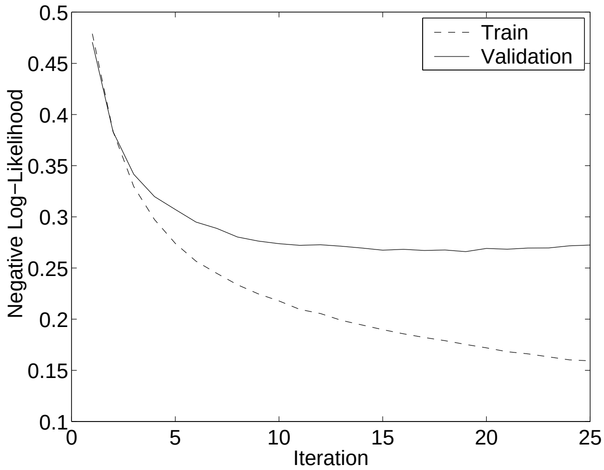
REF A. P. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of Royal Statistical Society*, 39:185–197, 1977.

Model	Test Error (%)	Time (minutes) 1 CPU	Parallel	Iteration
MLP (MSE, 500 hidden units)	14.2 12.9	105 2100	– –	100 2000
MLP (CE, 500 hidden units)	11.6 8.9	105 1050	– –	100 1000
Standard mixture (10 experts, 50 hidden units per expert and 150 units for the gater)	11.6 9.1	124 290	– –	65 150
Hard mixture, local gater (20 experts, 25 hidden units per expert and 20 Gaussians per $\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}\|k)$)	10.9	21	**1.3**	15

▲ **Table 6.2.** *Comparison of performance on the Forest data set between MLPs, a standard mixture, and a hard mixture of MLPs with "local" gater trained with Algorithm 6.2.*

in a short time when using sequential training (on only one computer). More impressively, we can obtain the same results in a much shorter time if we use parallelization (one computer per expert). Note however that it did not performed as well as the hard mixture with a global gater. Figure 6.3 shows the importance of the iterative process of our model for the training as well as generalization error, as previously shown for the hard mixture with global gater.

We performed one more experiment to compare the hard mixtures with global and local gaters in terms of training time. The experiment was performed on the Forest database, and the hyper-parameters were chosen to obtain good generalization results in both cases. The hard mixture with local gater had 20 experts, 25 hidden units per expert and 20 Gaussians per local gater. The hard mixture with global gater had 20 MLP experts, and an MLP gater with 150 hidden units. The hard mixture with global gater trained for more than 30 minutes to obtain a generalization error comparable to the hard mixture with local gater after training for *only* 1.3 *minutes*, as shown in Table 6.2. At first glance it seems that *the training time bottleneck caused to the gater disappeared thanks to the hard probabilistic mixture*. However, the hard mixture with a *global* gater was able to yield *better generalization results*, as with SVMs experts in Table 6.1.



▲ **Figure 6.3.** *Evolution of the negative log-likelihood with the number of iterations for the hard mixture with "local" gater, on the Forest database. The mixture has 20 MLP experts (25 hidden units, 20 Gaussians per local gater).*

SVM Experts

Similar experiments were performed on the Forest database with the hard mixture with "local" gater, but using SVMs plus a logistic regressor (as proposed by Platt, 1999b) as probabilistic experts, rather than MLPs. Therefore, given the SVM output $f_{\boldsymbol{\theta}}(\boldsymbol{x})$, we consider the conditional probability

$$\mathbf{P}_{a,w}(y|\boldsymbol{x}) = \frac{1}{1 + \exp(a - y\, w\, f_{\boldsymbol{\theta}}(\boldsymbol{x}))}\,,$$

and we train the parameters $a \in \mathbb{R}$ and $w \in R$ by minimizing the conditional negative log-likelihood over the training set

$$\mathcal{L}(a, w) = -\sum_{l=1}^{L} \log \mathbf{P}_{a,w}(y_l|\boldsymbol{x}_l)\,. \tag{6.10}$$

The training is very fast, because there are only two parameters to train. Equation (6.10) allows us to interpret the output of the SVM $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ as a conditional probability $\mathbf{P}_{a,w}(y|\boldsymbol{x})$. We thus can use the hard mixture with local gater (given in Algorithm 6.2) with SVM experts. Table 6.3 shows the best result (in terms of generalization error) obtained on the Forest database. Results are not as good (in time and generalization performance) as the hard MLP mixture with

REF J. C. 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, 1999b.

Model	Test Error (%)	Time (minutes) 1 CPU	Parallel	Iteration
20 SVM experts and 10 Gaussians per $\mathbf{P}_{\boldsymbol{\theta}}(\boldsymbol{x}\|k)$	10.7	2240	157	16

▲ **Table 6.3.** *Performance of the hard mixture with local gater, using SVMs experts and Gaussian mixtures models for the gaters. Result on the Forest database.*

a local gater in Table 6.2. This could be explained by the fact that the MLP experts are always trained in a stochastic way, starting from the previous iteration of the mixture algorithm. In the end, they "see" more training examples than the SVM experts which are retrained "from scratch" at each iteration.

Global or Local?

The generalization performance of a hard mixture with a local gater were in all cases worst than what we obtained with a global gater, especially with SVM experts. Obviously, a local gater is much less powerful than a global gater for re-distributing examples to the experts. Moreover even if we were able to obtain reasonable generalization results using the hard mixture with a local gater in a short time, we must admit that tuning the Gaussian Mixture Models in the local gater is very tricky compared to the tuning of the MLP global gater. In the end, the global gater seems to be the way to go...

A Note on Training Complexity

For hard mixtures with both global and local gaters, suppose that we choose the number of experts K such that the number of examples per expert $M = L/K$ 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 M, the training time for training the experts in one outer-loop iteration of the hard mixtures becomes on the order of:

$$O(KM^p) = O(LM^{p-1}) = \mathbf{O}(\mathbf{L})\,.$$

If the gater is not localized (as in Algorithm 6.1), then it may be a bottleneck. In the experiments we proposed, we did not measure the cost of training the gater, but as it was an MLP, it was most likely greater than $O(L)$, even though our experiments showed the training time of the hard mixture to scale linearly with L.

Contributions

This chapter is a synthesis of the following published papers:

- R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of SVMs for very large scale problems. *Neural Computation*, 14(5):1105–1114, 2002a.

  In this paper, we introduced for the first time the notion of "hard" mixture of experts, and proposed training Algorithm 6.1 with a "global" gater. At this time the goal was to find a way to train SVMs on large databases. Indeed, SVMs are a very "fashioned" algorithm, but known to be intractable on large training sets. Therefore, we proposed in this paper experiments with hard mixtures of SVMs.

- R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of SVMs for very large scale problems. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems, NIPS 14*, pages 633–640. MIT Press, 2002b.

  This paper is very similar to the previous one, with a few more details, and some experiments on another realistic database, to validate another time the algorithm in practice.

- R. Collobert, Y. Bengio, and S. Bengio. Scaling large learning problems with hard parallel mixtures. *International Journal on Pattern Recognition and Artificial Intelligence (IJPRAI)*, 17(3):349–365, 2003.

  In this paper, we proposed the hard mixture of experts with a "local" gater and Algorithm 6.2. We also added experiments with MLP experts, in addition to the ones with SVM experts.

Note that in this chapter, we extended Algorithm 6.1 to a probabilistic framework. We also made a clear link between the hard mixture with "global" gater and the one with a "local" gater in this probabilistic framework.

Conclusion

In this chapter we have presented 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 both in terms of training time and generalization performance on a real life database. Moreover, the algorithms scale linearly with the number of examples, at least in the ranges we presented. Both an algorithm with a global gater (that is where the gater is trained on the whole training set) and a local gater (where the local gaters are trained on the same subset as their corresponding expert) were presented, with a demonstration that in a probabilistic framework they actually minimize a well-defined criterion.

These results are extremely encouraging and suggest that the proposed method could allow training SVM-like models for sets of data on the order of millions in a reasonable time. In the experiments, two types of "gater" models were proposed, one based on a single MLP (for the mixture with a global gater), and the other based on local Gaussian Mixture Models (for the mixture with a local gater). The latter has 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 are obtained with the MLP gater. Surprisingly, even faster results (with similar generalization) are obtained if the SVM experts are replaced by MLP experts. If training of the MLP gater with stochastic gradient descent grows less than quadratically in time (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.

Finally, "hard" mixtures have one significant drawback: the number of *hyper-parameters* is quite large. Hyper-parameters exist for the experts, for the gater, and for local and global training algorithms of the mixture. In practice, tuning a mixture of experts is a nightmare. Thus, in the next chapter, we will focus on MLPs, which are much easier to use in practice. MLPs are also less resource consuming than the SVMs studied in Chapter 5, and are more suitable for large scale problems. However, we have raised some optimization issues in MLPs. Indeed, we observed performance differences between an MLP trained with the MSE criterion, an MLP trained with the CE criterion, and mixtures of experts. We will try to give an explanation of these differences, which could help us to find better ways to train MLPs.

7 Optimization for MLPs

Optimization by *gradient descent* is widely used by various machine learning algorithms such as back-propagation in Multi Layer Perceptrons (MLPs), Radial Basis Functions (see Bishop, 1995) or Mixture of Experts (MoEs). Several researchers proposed various enhancements (see LeCun et al., 1998) to gradient descent, but this technique remains a *first order* technique: it only uses the local gradient of the cost to minimize, as described in Chapter 3. Many finer methods, based on a *second order* approximation of the cost exist, as reviewed by Battiti (1992). Most of them, such as the original Newton method, but also Quasi-Newton, Gauss-Newton, Levenberg-Marquardt and Natural Gradient (introduced by Amari, 1998) methods need to compute the inverse of the Hessian (or an approximation of it). Using the Sherman-Morrison-Woodbury (Sherman and Morrison, 1949 and Woodbury, 1950) formula to compute iteratively this inverse (for details, see Bottou, 1998), the time and space complexities of the resulting algorithms grow in $O(M^2)$ per iteration, where M is the number of parameters. Thus, these algorithms are not well suited

REF C. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

REF Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In G.B. Orr and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, pages 9–50. Springer, 1998.

REF R. Battiti. First and second-order methods for learning: Between steepest descent and Newton's method. *Neural Computation*, 4(2):141–166, 1992.

REF S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 14(2): 251–276, 1998.

REF J. Sherman and W. J. Morrison. Adjustment of an inverse matrix corresponding to changes in the elements of a given column or given row of the original matrix. *Annals of Mathematics and Statistics*, 20:621, 1949.

REF M. A. Woodbury. Inverting modified matrices. Technical Report Technical Report 42, Statistical Research Group, Princeton University, 1950.

REF L. Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks.* Cambridge University Press, Cambridge, UK, 1998.

for very large data sets and models. Some improvements have been proposed, such as the Stochastic Meta Descent method (Schraudolph, 2002), which proposes a method to compute iteratively the product of a vector and the inverse of the Hessian, with $O(M)$ time and space complexities. However, in practice these methods are too complex to implement, and most people rely on gradient descent or stochastic gradient descent. As a consequence, instead of dealing with the Hessian during training as several other methods do, in this chapter we propose to study the Hessian of the cost function *before* training. This will also help us to understand differences in performance observed in Chapter 6 on several similar gradient-based models. Note that because we suspect some *optimization issues* in MLPs, we will only consider *training performance* in this chapter.

Framework Definition

We consider in this chapter the two-class classification problem defined in Chapter 3. We focus on two important models presented in Chapter 3 as well. We give a short reminder of the notation here, for convenience. First, we are interested in MLPs with one hidden layer and N hidden units, defined by

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = b + \sum_{n=1}^{N} w_n \, h(\boldsymbol{v}_n \cdot \boldsymbol{x}) \,, \tag{7.1}$$

where h is a transfer function (the hyperbolic transfer function for our experiments). Here $(\boldsymbol{w}, b) \in \mathbb{R}^{N+1}$ are the output weights, and $\boldsymbol{v}_n \in \mathbb{R}^{d+1} \times \mathbb{R}$ represents the weights of the n-th hidden unit. To simplify the notation, we suppose that the last coordinate of an input example $\boldsymbol{x}$ is 1, which implies that the bias of the n-th unit is represented by the last coordinate of $\boldsymbol{v}_n$.

We will also consider a certain class of MoEs, defined by the following decomposition with K experts:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{k=1}^{K} g_{\boldsymbol{\theta},k}(\boldsymbol{x}) \, f_{\boldsymbol{\theta},k}(\boldsymbol{x}) \,, \tag{7.2}$$

under the conditions

$$\sum_{k=1}^{K} g_{\boldsymbol{\theta},k} = 1 \quad \text{and} \quad g_{\boldsymbol{\theta},k} \geq 0 \quad \forall k \,. \tag{7.3}$$

REF N. N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.

The function $f_{\boldsymbol{\theta},k}$ is the output of the k-expert, which will be an MLP as described by (7.1), in this chapter. The gater $g_{\boldsymbol{\theta}}(\cdot) = \{g_{\boldsymbol{\theta},1}(\cdot), \ldots, g_{\boldsymbol{\theta},K}(\cdot)\}$ is taken as an MLP as well, with soft-max outputs, as described in Chapter 3.

Training

As presented in Chapter 3, we consider the training of MLPs and MoEs by minimizing the empirical risk

$$J : \boldsymbol{\theta} \mapsto \frac{1}{L} \sum_{l=1}^{L} Q((\boldsymbol{x}_l, y_l), f_{\boldsymbol{\theta}}), \tag{7.4}$$

using *stochastic* gradient descent, and given a criterion Q. We focus on the Mean Squared Error (MSE) criterion defined with

$$Q((\boldsymbol{x}, y), f_{\boldsymbol{\theta}}) = \frac{1}{2} (y - f_{\boldsymbol{\theta}}(\boldsymbol{x}))^2 ,$$

and on the Cross-Entropy (CE) criterion given by

$$Q((\boldsymbol{x}, y), f_{\boldsymbol{\theta}}) = \log(1 + \exp(-y\, f_{\boldsymbol{\theta}}(\boldsymbol{x}))) .$$

We will also consider the training of an MLP (7.1) with an added hyperbolic tangent at the output, which is equivalent to use the criterion

$$Q((\boldsymbol{x}, y), f_{\boldsymbol{\theta}}) = \frac{1}{2} (y - \tanh(f_{\boldsymbol{\theta}}(\boldsymbol{x})))^2 . \tag{7.5}$$

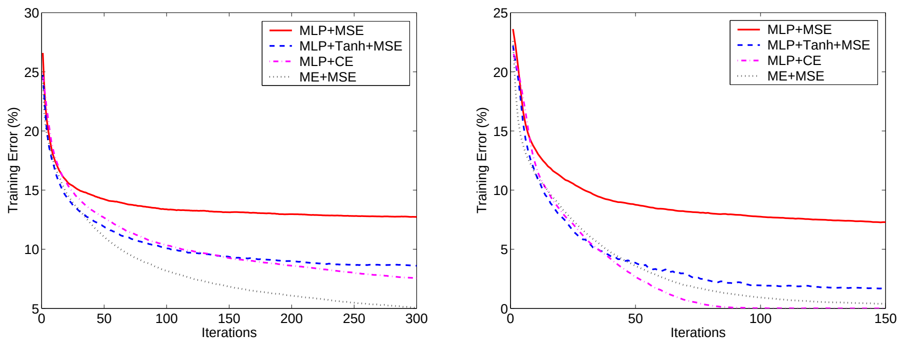
Preliminary Experiments

In order to highlight the optimization problems we suspected from the training of MLPs in Chapter 6, we performed a preliminary comparison (see Table 7.1) of *training performance* of several architectures on the Forest and the Connect-4 databases presented in Chapter 4. We considered several MLPs trained with different criteria and a MoE trained using the MSE criterion. The number of hidden units of the MLPs was chosen to obtain the best training performance with the MSE criterion. *We were not able to reduce the training errors by adding more hidden units* to MLPs trained with the MSE criterion. Apart from the MLP number of hidden units, all other hyper-parameters were chosen to maximize the training performance for each model. The hyper-parameters of the mixtures were chosen such that the mixtures had slightly fewer parameters than the MLPs.

The fact that, on these two data sets, an MLP trained with the CE criterion leads to significantly better training performance than an MLP trained with

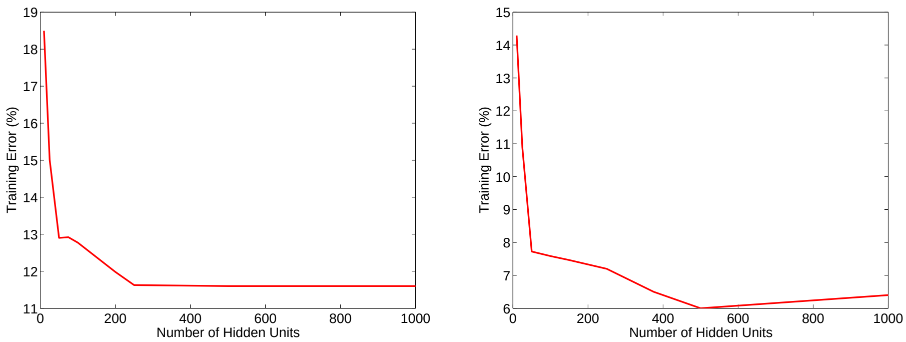
Architecture	Train Err. (%)	
	Forest	Connect-4
MLP+MSE	11.6	6.0
MLP+Tanh+MSE	7.1	1.4
MLP+CE	4.4	0.0
MoE+MSE	4.0	0.2

▲ **Table 7.1.** *Training errors (on the Forest and Connect-4 databases) for several architectures: an MLP trained with the MSE and CE criteria, an MLP with a hyperbolic tangent output trained with the MSE criterion, and a MoE trained with the MSE criterion. All MLPs had 500 hidden units. The mixture hyper-parameters were chosen such that the mixtures had less parameters than MLPs.*



(a) (b)

▲ **Figure 7.1.** *Comparison of the training error (in %) with respect to the number of iterations over the training set for several architectures: an MLP trained with the MSE and CE criteria, an MLP with a hyperbolic tangent output trained with the MSE criterion, and an MoE trained with the MSE criterion. All MLPs had 500 hidden units. The hyper-parameters of the mixtures were chosen such that the mixtures had less parameters than MLPs. Results are shown on the Forest database in (a) and on the Connect-4 database in (b).*



(a) (b)

▲ **Figure 7.2.** *Training error (in %) with respect to the number hidden units for an MLP trained with the MSE criterion on the Forest database (a) and on the Connect-4 database (b).*

the MSE criterion (in much less time as shown in Figure 7.1), and that it is not possible to decrease the classification error by adding hidden units (see Figure 7.2), clearly denotes *an optimization issue with the MLP trained with the MSE criterion*. Indeed, MLPs with one hidden layer are universal approximators, as pointed out in Chapter 3. It should be possible, with enough hidden units, to reduce the training error to zero. Here, we are clearly *under-fitting* at least with the MSE criterion. This is a serious problem, because as shown in Chapter 2, under-fitting can imply bad generalization performance. The fact that, in Chapter 6, the MLP with the MSE criterion leads to worse generalization performance than the MLP with the CE criterion could be explained because we are much more under-fitting when using the MSE criterion than when using the CE criterion. However, it seems that the MSE criterion is not solely responsible for this problem, as the mixtures trained with MSE criterion lead to better results. Simply adding a hyperbolic tangent to the output of the MLP seems to improve the training performance.

A Local Cost Study

In order to understand these differences of performance, we propose to study the *local* behavior (with respect to the parameters $\boldsymbol{\theta}$) of each model and its corresponding cost function. Given a model $f_{\boldsymbol{\theta}}(\cdot)$ where we want to optimize the parameters $\boldsymbol{\theta}$, the "stochastic" version of cost function (7.4)

$$J_{\boldsymbol{x},y}(\boldsymbol{\theta}) = Q((\boldsymbol{x},\, y),\, f_{\boldsymbol{\theta}})$$

can be approximated with respect to $\boldsymbol{\theta}$ around $\boldsymbol{\theta}^o$, using a second order Taylor expansion:

$$\begin{aligned} J_{\boldsymbol{x},y}(\boldsymbol{\theta}) = J_{\boldsymbol{x},y}(\boldsymbol{\theta}^o) \\ + (\boldsymbol{\theta} - \boldsymbol{\theta}^o)^{\mathrm{T}} \frac{\partial J_{\boldsymbol{x},y}(\boldsymbol{\theta}^o)}{\partial \boldsymbol{\theta}} \\ + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^o)^{\mathrm{T}} \boldsymbol{H}_{\boldsymbol{x},y}(\boldsymbol{\theta}^o) \ (\boldsymbol{\theta} - \boldsymbol{\theta}^o) \\ + o(\|\boldsymbol{\theta} - \boldsymbol{\theta}^o\|_2^2) \end{aligned} \tag{7.6}$$

where $\partial J_{\boldsymbol{x},y}(\boldsymbol{\theta}^o)/\partial \boldsymbol{\theta}$ and $\boldsymbol{H}_{\boldsymbol{x},y}(\boldsymbol{\theta}^o)$ are the gradient vector and the Hessian matrix of $J_{\boldsymbol{x},y}$ with respect to $\boldsymbol{\theta}$ evaluated at $\boldsymbol{\theta}^o$, respectively. We use $\|.\|_2$ as the Euclidean norm for vectors, and $o(\|\boldsymbol{\theta} - \boldsymbol{\theta}^o\|_2^2)$ to represent a term negligible with respect to $\|\boldsymbol{\theta} - \boldsymbol{\theta}^o\|_2^2$. Remember that if we write $\boldsymbol{\theta} = (\theta_1, \theta_2 \dots)$, then the coefficient (i, j) of the Hessian matrix of the function $J_{\boldsymbol{x},y}$ is equal to

$$H^{i,j}_{\boldsymbol{x},y} = \frac{\partial J_{\boldsymbol{x},y}}{\partial \theta_i \, \partial \theta_j} \, .$$

It is obvious that if we want the stochastic gradient descent to work, then the first derivative should be non zero for a given misclassified example $(\boldsymbol{x}, y)$. It is also known that the Hessian is important as well, and related to the time of convergence as shown by (LeCun et al., 1991). Thus, we will first consider the Hessian in a general way, and then show that in the particular case of a *block-diagonal* Hessian, the training problem is much simpler than for a full Hessian.

The Advantage of a Block Diagonal Hessian

Let us consider a model $f_{\boldsymbol{\theta}}(\cdot)$ where the parameter vector $\boldsymbol{\theta}$ can be segmented into several *sub-vectors* $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \dots \boldsymbol{\theta}_M)$. Given the current state $\boldsymbol{\theta}^o$, and if we ignore negligible terms with respect to $\|\boldsymbol{\theta} - \boldsymbol{\theta}^o\|_2^2$, then the local quadratic approximation (7.6) can be rewritten as:

$$\begin{aligned} J_{\boldsymbol{x},y}(\boldsymbol{\theta}) = J_{\boldsymbol{x},y}(\boldsymbol{\theta}^o) \\ + \sum_i (\boldsymbol{\theta}_i - \boldsymbol{\theta}_i^o)^{\mathrm{T}} \frac{\partial J_{\boldsymbol{x},y}(\boldsymbol{\theta}^o)}{\partial \boldsymbol{\theta}} \\ + \sum_{i,j} \frac{1}{2}(\boldsymbol{\theta}_i - \boldsymbol{\theta}_i^o)^{\mathrm{T}} \boldsymbol{H}^{i,j}_{\boldsymbol{x},y}(\boldsymbol{\theta}^o) \ (\boldsymbol{\theta}_j - \boldsymbol{\theta}_j^o) \, , \end{aligned} \tag{7.7}$$

REF Y. LeCun, I. Kanter, and S. Solla. Second order properties of error surfaces: learning time, generalization. In R. P. Lippman, J. M. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 918–924, Denver, CO, 1991. Morgan Kaufmann.

where $\boldsymbol{H}^{i,j}_{\boldsymbol{x},y}$ is the matrix

$$\frac{\partial^2 J_{\boldsymbol{x},y}}{\partial \boldsymbol{\theta}_i \, \partial \boldsymbol{\theta}_j}\,,$$

using vectorial derivation.

Truly Block-Diagonal Hessian

In the ideal case where $\boldsymbol{H}_{\boldsymbol{x},y}$ is truly block-diagonal, that is if $\boldsymbol{H}^{i,j}_{\boldsymbol{x},y} = 0$ for $i \neq j$, this leads to

$$J_{\boldsymbol{x},y}(\boldsymbol{\theta}) = J_{\boldsymbol{x},y}(\boldsymbol{\theta}^o) + \sum_i J^i_{\boldsymbol{x},y}(\boldsymbol{\theta}_i) \tag{7.8}$$

where

$$\begin{aligned} J^i_{\boldsymbol{x},y}(\boldsymbol{\theta}_i) = {} & (\boldsymbol{\theta}_i - \boldsymbol{\theta}^o_i)^{\mathrm{T}} \, \frac{\partial J_{\boldsymbol{x},y}(\boldsymbol{\theta}^o)}{\partial \boldsymbol{\theta}_i} \\ & + \frac{1}{2}(\boldsymbol{\theta}_i - \boldsymbol{\theta}^o_i)^{\mathrm{T}} \, \boldsymbol{H}^{i,i}_{\boldsymbol{x},y}(\boldsymbol{\theta}^o) \; (\boldsymbol{\theta}_i - \boldsymbol{\theta}^o_i)\,. \end{aligned}$$

Equation (7.8) shows that the error function $J_{\boldsymbol{x},y}(\boldsymbol{\theta})$ can be split into M independent error functions $J^i_{\boldsymbol{x},y}(\boldsymbol{\theta}_i)$. In other words, the optimization of a sub-vector $\boldsymbol{\theta}_i$ is locally independent of the others. Therefore, the optimization problem is *much simpler* than with a full Hessian where the modification of one parameter would affect the modification of the others.

Almost Block-Diagonal Hessian

If $\boldsymbol{H}_{\boldsymbol{x},y}$ is not truly block-diagonal, one can still compute the difference between the true local cost $J_{\boldsymbol{x},y}$ and the approximation given by (7.8) around $\boldsymbol{\theta}^o$:

$$\begin{aligned} \| J_{\boldsymbol{x},y}(\boldsymbol{\theta}) - J_{\boldsymbol{x},y}(\boldsymbol{\theta}^o) + \sum_i J^i_{\boldsymbol{x},y}(\boldsymbol{\theta}_i) \|_2 &= \| \frac{1}{2} \sum_{i \neq j} (\boldsymbol{\theta}_i - \boldsymbol{\theta}^o_i)^{\mathrm{T}} \, \boldsymbol{H}^{i,i}_{\boldsymbol{x},y}(\boldsymbol{\theta}^o) \; (\boldsymbol{\theta}_i - \boldsymbol{\theta}^o_i) \|_2 \\ &\leq \frac{1}{2} \sum_{i \neq j} \| \boldsymbol{H}^{i,j}_{\boldsymbol{x},y}(\boldsymbol{\theta}^o) \|_2 \, \| \boldsymbol{\theta}_i - \boldsymbol{\theta}^o_i \|_2 \, \| \boldsymbol{\theta}_j - \boldsymbol{\theta}^o_j \|_2 \,, \end{aligned}$$

where $\| \cdot \|_2$ denotes the spectral norm of a matrix (note that it corresponds here to the maximum eigenvalue of the matrix, the Hessian being symmetric). Thus, as the spectral norm $\| \boldsymbol{H}^{i,j}_{\boldsymbol{x},y} \|_2$ of the blocks of the Hessian outside the diagonal tends to zero, equation (7.8) becomes more accurate, and the training of each sub-vector $\boldsymbol{\theta}_i$ becomes more independent. Generally speaking, the more block-diagonal the Hessian, the easier it is to train the model.

A Second Order Method

Choosing an architecture *before training* which leads to a block-diagonal Hessian may lead to more easily trained models. This can be viewed as a kind of second order method: instead of computing the Hessian during training as many methods do, we can consider it before training. Indeed, we will show in the next section that for common models, simple changes in the architecture can strongly affect the Hessian.

Illustration

We will now illustrate the above concepts on the architectures we proposed in the preliminary experiments, summarized in Table 7.1. We will analyze the gradient of the cost for each situation. We also propose a mathematical analysis of the "instantaneous" Hessian $\boldsymbol{H}_{\boldsymbol{x},y}$, as well as an empirical evaluation of the "total" Hessian

$$\boldsymbol{H} = \frac{1}{L} \sum_{l=1}^{L} \boldsymbol{H}_{\boldsymbol{x}_l, y_l} \tag{7.9}$$

of the cost function (7.4) (which is what we would really like to minimize).

MoEs and MSE Criterion

Let us first consider the mixture of MLPs $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{k=1}^{K} g_k(\boldsymbol{x})\, f_k(\boldsymbol{x})$ as given by (7.2). If we introduce $\boldsymbol{u}_i$ as the weight *vector* of the expert f_i, then using the MSE criterion, the gradient of the cost can be computed as follows

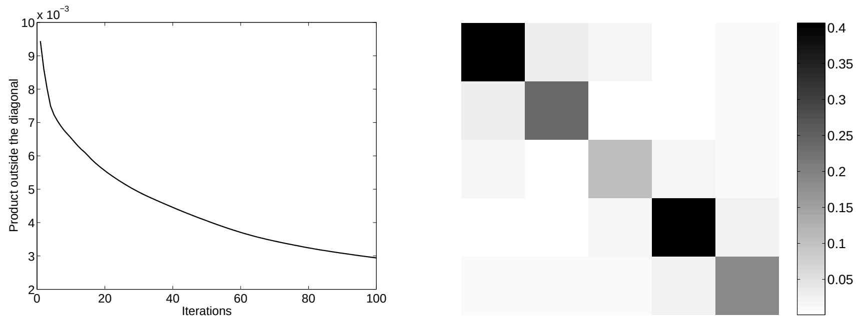
$$\frac{\partial J_{\boldsymbol{x},y}}{\partial \boldsymbol{u}_i} = -(y - f_{\boldsymbol{\theta}}(\boldsymbol{x}))\, g_i(\boldsymbol{x})\, \frac{\partial f_i(\boldsymbol{x})}{\partial \boldsymbol{u}_i} \,.$$

The gradient behaves as we would like: the first term $(y - f_{\boldsymbol{\theta}}(\boldsymbol{x}))$ is non-null when the example $(\boldsymbol{x}, y)$ is misclassified. At least one gater output $g_i(\boldsymbol{x})$ is always non-null because the gater outputs must sum to one. Hence, at least one expert will always receive some gradient when the example is misclassified.

Furthermore, the instantaneous Hessian *outside the block-diagonal* (one block for each pair of experts (i, j)) becomes:

$$\frac{\partial^2 J_{\boldsymbol{x},y}}{\partial \boldsymbol{u}_i\, \partial \boldsymbol{u}_j} = g_i(\boldsymbol{x})\, g_j(\boldsymbol{x}) \frac{\partial f_i(\boldsymbol{x})}{\partial \boldsymbol{u}_i} \left(\frac{\partial f_j(\boldsymbol{x})}{\partial \boldsymbol{u}_j} \right)^{\mathrm{T}} \quad (i \neq j) \,. \tag{7.10}$$

Note that the underlying idea of MoEs is to partition the input space using a gater, in which case a given example would be handled by only one expert. Thus, in the ideal case, given an example $(\boldsymbol{x},\, y)$, all but one $g_i(\boldsymbol{x})$ should be near zero. To verify this, we computed the average of the product $g_i(\boldsymbol{x})\, g_j(\boldsymbol{x})$ over the training set on the Forest data set, and over all pairs of distinct gater outputs $(i \neq j)$, with respect to the number of training iterations, as shown in Figure 7.3a. It is clear that the product $g_i\, g_j$ tends to be very small



(a) Average product of distinct pairs of gater outputs

(b) Spectral norm of each block in the Hessian matrix

▲ **Figure 7.3.** *Block-diagonality of the Hessian matrix. Figure (a) shows that the term $g_i(\boldsymbol{x})\, g_j(\boldsymbol{x})$ $(i \neq j)$, which appears in the Hessian equation outside the block-diagonal, tends to very small values on average during training. This is performed on 100,000 examples of the Forest data set, with 10 experts, 25 hidden units per expert and 100 hidden units for the gater. Figure (b) is a description of the Hessian matrix; each block corresponds to a pair of experts, and is represented by its spectral norm. The color of each block is associated with the corresponding value in the right column. This has been computed with 10,000 examples from the Forest data set, after 5 iterations with 5 experts, 10 hidden units per expert, and 25 hidden units for the gater.*

in practice, and thus the Hessian (7.10) converges to zero outside the block-diagonal. To further verify this assertion, we also computed the spectral norm in Figure 7.3b

$$\left\| \frac{\partial^2 J_{\boldsymbol{x},y}}{\partial \boldsymbol{u}_i \, \partial \boldsymbol{u}_j} \right\|_2$$

for each block (i, j) of the total Hessian (7.9) after few training iterations. We see that the Hessian quickly becomes diagonal during training. Hence, the optimal weights of an expert i do not depend locally on the optimal weights of an expert $j \neq i$, early in the training. The optimization problem is broken into several smaller ones, which simplifies the problem.

MLPs With MSE and CE Criteria

Let us now focus on the case of MLPs as given by (7.1) trained with the MSE and CE criteria. The first derivative of the instantaneous cost $J_{\boldsymbol{x},y}$ is

$$\frac{\partial J_{\boldsymbol{x},y}}{\partial \boldsymbol{v}_i} = -(y - f_{\boldsymbol{\theta}}(\boldsymbol{x}))\, w_i\, h'(\boldsymbol{v}_i \cdot \boldsymbol{x})\, \boldsymbol{x}$$

for the MSE criterion, and

$$\frac{\partial J_{\boldsymbol{x},y}}{\partial \boldsymbol{v}_i} = -\mathbf{P}_{\boldsymbol{\theta}}(-y|\boldsymbol{x})\, w_i\, h'(\boldsymbol{v}_i \cdot \boldsymbol{x})\, \boldsymbol{x} \tag{7.11}$$

for the CE criterion, where $\mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x})$ is given using the relation

$$\mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x}) = \frac{1}{1+\exp(-y\, f_{\boldsymbol{\theta}}(\boldsymbol{x}))}\,,$$

from the definition of the CE criterion, as explained in Chapter 3. Note that given a misclassified example $(\boldsymbol{x},\, y)$ the first term will be non-null (either $(y - f_{\boldsymbol{\theta}}(\boldsymbol{x}))$ or $\mathbf{P}_{\boldsymbol{\theta}}(-y|\boldsymbol{x})$). Thus, if the initial values of the output layer satisfy $w_i \neq 0 \forall i$, all hidden units should receive a gradient.

If we now compute the Hessian derived from the MSE criterion using a vector notation, outside the block-diagonal we have:

$$\frac{\partial^2 J_{\boldsymbol{x},y}}{\partial \boldsymbol{v}_i\, \partial \boldsymbol{v}_j} = w_i\, w_j\, h'(\boldsymbol{v}_i \cdot \boldsymbol{x})\, h'(\boldsymbol{v}_j \cdot \boldsymbol{x})\, \boldsymbol{x}\, \boldsymbol{x}^{\mathrm{T}} \quad (i \neq j)\,. \tag{7.12}$$

Note that there is no obvious reason for this Hessian to tend to zero. If we compute the Hessian with the CE criterion we get:

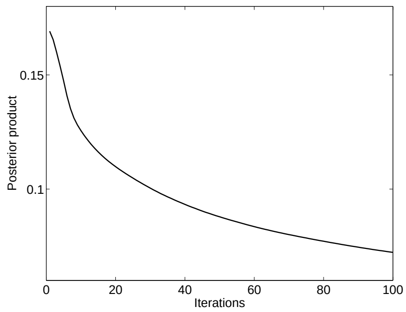
$$\frac{\partial^2 J_{\boldsymbol{x},y}}{\partial \boldsymbol{v}_i\, \partial \boldsymbol{v}_j} = \mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x})\, \mathbf{P}_{\boldsymbol{\theta}}(-y|\boldsymbol{x})\, w_i\, w_j\, h'(\boldsymbol{v}_i \cdot \boldsymbol{x}) h'(\boldsymbol{v}_j \cdot \boldsymbol{x})\, \boldsymbol{x}\, \boldsymbol{x}^{\mathrm{T}} \quad (i \neq j)\,.$$

Here the term $\mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x})\, \mathbf{P}_{\boldsymbol{\theta}}(-y|\boldsymbol{x}) = \mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x})(1 - \mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x}))$ will tend quickly to zero, since we are training the MLP to maximize $\mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x})$. This is verified in practice, as shown in Figure 7.4a. This term will push the Hessian obtained with the CE criterion toward block-diagonality (*one block for each unit*) (see Figure 7.4b), whereas the Hessian obtained with the MSE criterion remains full, as shown in Figure 7.5. This can explain the difference performance obtained in our preliminary comparison given in Table 7.1: when using the CE criterion, the minimization of the cost function is locally *more simpler* than when using the MSE criterion. Indeed, the landscape of the cost function with respect to the weights of a unit i is almost independent on a different unit j, using the CE criterion.

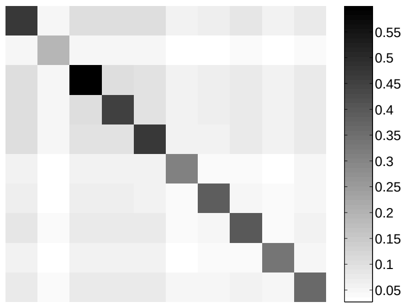
MLPs With Tanh Output and MSE criterion

It is commonly known that adding a hyperbolic tangent to the output layer of the MLP proposed in (7.1) may improve the training performance, as verified by our experiments shown in Table 7.1. As previously stated, it is equivalent to train the MLP (7.1) using the criterion (7.5). The gradient of the local cost $J_{\boldsymbol{x},y}$ is easily given by

$$\frac{\partial J_{\boldsymbol{x},y}}{\partial \boldsymbol{v}_i} = -(y - z)\,(1 - z^2)\, w_i\, h'(\boldsymbol{v}_i \cdot \boldsymbol{x})\, \boldsymbol{x}\,, \tag{7.13}$$

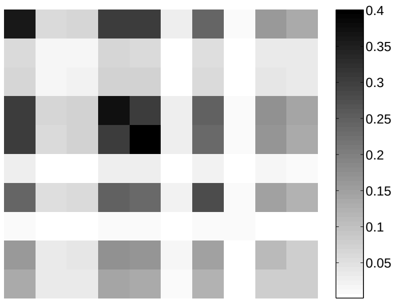


(a) Average product of posterior probabilities $p(y|x)p(-y|x)$

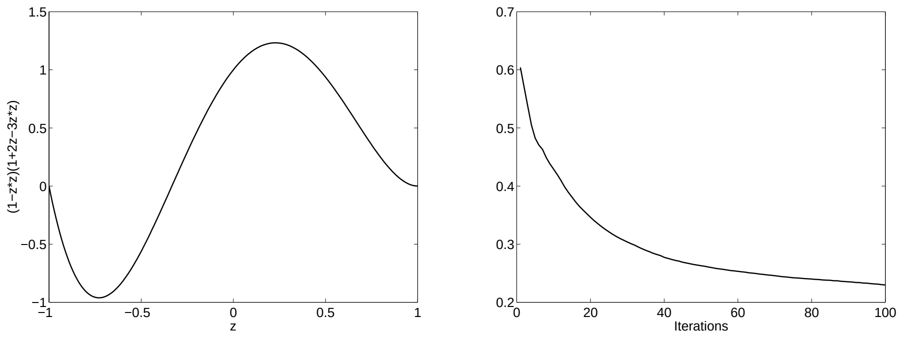


(b) Spectral norm of each block in the Hessian matrix

▲ **Figure 7.4.** *Block-diagonality of the Hessian matrix of an MLP trained with the CE criterion. Figure (a) shows that the term* $\mathbf{P}_{\boldsymbol{\theta}}(y|\boldsymbol{x})\mathbf{P}_{\boldsymbol{\theta}}(-y|\boldsymbol{x})$*, which appears in the Hessian equation outside the block-diagonal, tends to very small values on average during training. This is performed on the Forest data set, with 100,000 training examples and 200 hidden units. Figure (b) is a description of the Hessian matrix with 10 hidden units; each block corresponds to a pair of hidden units, and is represented by its spectral norm. The color of each block is associated with the corresponding value in the right column. It has been computed on the Forest data set with 10,000 training examples, after one iteration.*



▲ **Figure 7.5.** *Hessian matrix for an MLP with 10 hidden units trained with the MSE criterion. Each block corresponds to a pair of hidden units, and is represented by its spectral norm. The color of each block is associated with the corresponding value in the right column. It has been computed on the Forest data set with 10,000 training examples, after 10 iterations.*



(a) Plot of the function $c(z) = (1-z^2)(1+2z\,y-3z^2)$.

(b) Average over the examples of the coefficient $|c(z)|$ during training.

▲ **Figure 7.6.** *Block-diagonality of the Hessian for an MLP trained with the MSE criterion and hyperbolic tangent outputs. Figure (a) shows that the coefficient $c(z)$ involved in the Hessian tends to zero when the output $z = \tanh(f(x))$ tends to ± 1. Figure (b) shows that the average over the training examples of the absolute value of the coefficient $c(z)$ decreases during training. Figures (a) and (b) have been computed on the Forest database with 500 hidden units and 100,000 training examples.*

where we denote

$$z = \tanh(f_{\boldsymbol{\theta}}(\boldsymbol{x})) \tag{7.14}$$

to simplify the notation, and using the fact that $\tanh'(\cdot) = 1 - \tanh(\cdot)^2$. Each block of the Hessian outside the block-diagonal can then be written as follows:

$$\frac{\partial^2 J_{\boldsymbol{x},y}}{\partial \boldsymbol{v}_i\, \partial \boldsymbol{v}_j} = (1-z^2)(1+2z\,y-3z^2)\, w_i\, w_j\, h'(\boldsymbol{v}_i \cdot \boldsymbol{x})\, h'(\boldsymbol{v}_j \cdot \boldsymbol{x})\, \boldsymbol{x}\, \boldsymbol{x}^{\mathrm{T}} \quad (i \neq j)\,. \tag{7.15}$$

Note the difference between the Hessian in (7.15) and the Hessian obtained when training an MLP without the hyperbolic tangent output in (7.12) is simply the coefficient

$$c(z) = (1-z^2)(1+2z\,y-3z^2)\,.$$

We plotted this coefficient with respect to z in Figure 7.6a for the case of $y = 1$, and for a range of z values between -1 and 1. This corresponds to the values the hyperbolic tangent can take in (7.14). It is interesting to note that this coefficient tends to zero when $z = \tanh(f(x))$ tends to ± 1. As we are training the MLP for this purpose, $c(z)$ will tend to small values during training, as shown in Figure 7.6b. Once again, the Hessian will tend to have small values outside the block-diagonal. Note, however, that there is a drawback when z tends to ± 1: the first derivative (7.13) will decrease, in particular *for misclassified examples*, due to the coefficient $1-z^2$. However, for an MLP with a CE criterion, the first derivative (7.11) is (in general) small only for well-classified examples. Even if the hyperbolic tangent output improves training results for an MLP with the MSE criterion, we could expect better results in general with the CE criterion, as shown in Table 7.1.

Contributions

This content of this chapter was first published in the following paper:

R. Collobert and S. Bengio. A gentle hessian for efficient gradient descent. In *IEEE International Conference on Acoustic, Speech, and Signal Processing, ICASSP*, 2004a.

In this chapter, we expanded on details in the derivations, added an illustration of our theory with a certain class of mixture of experts, and introduced more figures. The application to MLPs of the "margin criterion" coming from SVMs, was postponed until Chapter 8.

Conclusion

In this chapter we highlighted some *under-fitting* problems while training certain large models with gradient descent. Under-fitting problems are quite serious, because they prevent good generalization performance, as seen in Chapter 2. We thus analyzed gradient descent algorithms with respect to a local study of the minimized cost function. In particular, we pointed out that the Hessian matrix plays a major role in the effectiveness of any gradient descent algorithm. We showed, both empirically and theoretically, that a block-diagonal Hessian should yield more efficient training algorithms. Based on several common models, we pointed out that the choice of model architecture influences the Hessian matrix. In particular, we explained why the CE criterion should be selected for classification problems instead of the more classical MSE criterion when training MLPs. We also explained why MoEs may lead to better results than MLP trained with the MSE criterion. Our analysis can now be used to create models more well adapted to gradient descent. Finally, while efficient optimization is important, it is not sufficient to obtain good generalization performance: the ultimate goal of machine learning. Thus, we will now focus on what may help machine learning algorithms to generalize. We will consider the margin idea introduced in the SVM algorithm, which is known to improve generalization performance.

8 A Margin for Everyone

Since the Perceptron algorithm proposed by Rosenblatt (1957), several machine learning algorithms have been proposed for classification problems, as already introduced in Chapter 3. Among them, two algorithms had a large impact on the research community. The first one is the Multi Layer Perceptron (MLP), which became useful with the introduction of the back-propagation training algorithm by LeCun (1985) and Rumelhart et al. (1986). The second one, proposed more recently, is the Support Vector Machine (SVM) algorithm proposed by Vapnik (1995). SVMs supplied the large margin classifier idea, often referred to as a key point for improving generalization performance for binary classification tasks, following Vapnik's claim. We propose comparing Perceptrons, MLPs and SVMs in this chapter, according to their generalization ability, which is the main concern of machine learning, as highlighted in Chapter 2. As the margin idea introduced in SVMs appears to be a nice way to control the generalization ability, we will try to give a better understanding of Perceptrons and MLPs through the lens of the SVM margin.

REF F. Rosenblatt. The perceptron: a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, N.Y., 1957.

REF Y. LeCun. A learning scheme for asymmetric threshold networks. In *Proceedings of Cognitiva 85*, pages 599–604, Paris, France, 1985.

REF D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by back-propagating errors. In D.E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, 1986.

REF V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, second edition, 1995.

Framework Definition

Once again, we consider the two-class classification problem defined in Chapter 3. We focus on Perceptrons (and their Φ-Machine extension), MLPs and SVMs which were already defined in Chapter 3. However, we want to first give a rapid overview of these models in the same framework, in order to show how close they are. The decision function $f_{\boldsymbol{\theta}}(.)$ of all these models can be written in its general form as

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w} \cdot \Phi(\boldsymbol{x}) + b\,, \tag{8.1}$$

where $\boldsymbol{w}$ is a real vector of weights and $b \in \mathbb{R}$ is a bias. The generic vector of parameters $\boldsymbol{\theta}$ represents all the parameters of the decision function (that is, $\boldsymbol{w}$, b, and all parameters of Φ, if any). For Φ-machines and SVMs, Φ is an arbitrarily chosen function; in the special case of $\Phi(\boldsymbol{x}) = \boldsymbol{x}$, it leads respectively to Perceptrons and linear SVMs. We also consider MLPs with one hidden layer and N hidden units. In that case the hidden layer can be decomposed as $\Phi(\cdot) = (\Phi_1(\cdot),\, \Phi_2(\cdot),\, \dots,\, \Phi_N(\cdot))$, where the n-th hidden unit is described with

$$\Phi_n(x) = h(\boldsymbol{v}_n \cdot x + a_n)\,. \tag{8.2}$$

Here $(\boldsymbol{v}_n, a_n) \in \mathbb{R}^d \times \mathbb{R}$ represents the weights of the n-th hidden unit, and h is a transfer function which is usually a sigmoid or a hyperbolic tangent. Note that the hyperspace defined with $\{\Phi(\boldsymbol{x}),\ \boldsymbol{x} \in \mathbb{R}^d\}$ will be called "feature space", or sometimes "hidden layer space" in the special case of MLPs.

SVM Training

We already showed in Chapter 3 that SVMs aim at minimizing the cost

$$J\,:\,(\boldsymbol{\theta},\,\boldsymbol{\xi}) \mapsto \frac{\mu}{2}\,\|\boldsymbol{w}\|^2 + \frac{1}{L}\sum_{l=1}^{L} \xi_l \tag{8.3}$$

under the constraints

$$\forall l \in \{1 \dots 2L\} \quad \varphi_l(\boldsymbol{\theta},\,\boldsymbol{\xi}) \leq 0\,, \tag{8.4}$$

where

$$\forall l \in \{1 \dots L\} \quad \begin{cases} \varphi_l(\boldsymbol{\theta},\,\boldsymbol{\xi}) = -\xi_l \\ \varphi_{L+l}(\boldsymbol{\theta},\,\boldsymbol{\xi}) = 1 - \xi_l - y_l\, f_{\boldsymbol{\theta}}(\boldsymbol{x}_l)\,. \end{cases} \tag{8.5}$$

By noticing that

$$\begin{aligned} \forall l, \quad |1 - y_l\, f_{\boldsymbol{\theta}}(\boldsymbol{x}_l)|_+ &= \max(0,\, 1 - y_l\, f_{\boldsymbol{\theta}}(\boldsymbol{x}_l)) \\ &= \min\{\, \xi_l \ /\ \xi_l \geq 0,\ \xi_l \geq 1 - y_l\, f_{\boldsymbol{\theta}}(\boldsymbol{x}_l)\,\}\,, \end{aligned}$$

it appears that it is equivalent to minimize the cost function

$$\boldsymbol{\theta} \mapsto \frac{\mu}{2} \|\boldsymbol{w}\|^2 + \frac{1}{L} \sum_{l=1}^{L} |1 - y_l \, f_{\boldsymbol{\theta}}(\boldsymbol{x}_l)|_+ \,, \tag{8.6}$$

where $|z|_+ = \max(0, z)$.

Reminder of KKT Conditions

We showed in Chapter 3 that finding a minimum of (8.3) under the constraints (8.4) is achieved using a Lagrangian technique. In particular, according to Appendix A, if $(\boldsymbol{w}, b, \boldsymbol{\xi})$ is a minimum of (8.3) it is *necessary* that Lagrange multipliers $(\boldsymbol{\alpha}, \boldsymbol{\eta})$ exist such that $(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\eta})$ satisfies the *Karush-Kuhn and Tucker* (KKT) conditions, that we rewrite here for convenience:

$$\begin{aligned}
\boldsymbol{w} = \frac{1}{\mu} \sum_{l=1}^{L} \alpha_l \, y_l \, \Phi(\boldsymbol{x}_l) & \\
\sum_{l=1}^{L} \alpha_l \, y_l = 0 & \\
\alpha_l \left[1 - \xi_l - y_l \left(\boldsymbol{w} \cdot \Phi(\boldsymbol{x}_l) + b\right)\right] = 0 & \quad \forall l \\
\frac{1}{L} - \alpha_l - \eta_l = 0 & \quad \forall l \\
\eta_l \, \xi_l = 0 & \quad \forall l \\
\alpha_l \geq 0 & \quad \forall l \\
\eta_l \geq 0 & \quad \forall l \,.
\end{aligned} \tag{8.7}$$

As the minimization problem (8.3) is convex with convex constraints (8.4), we are also ensured that KKT conditions are *sufficient*. In other words, if we find $(\boldsymbol{w}, b, \boldsymbol{\xi})$ such that the constraints (8.4) are satisfied, then if $(\boldsymbol{\alpha}, \boldsymbol{\eta})$ exists such that the KKT conditions (8.7) hold, it guarantees that $(\boldsymbol{w}, b, \boldsymbol{\xi})$ is a minimum of the SVM problem.

Perceptron, Φ-machine and MLP Training

As presented in Chapter 3, the training of Perceptrons, Φ-machines and MLPs is usually achieved with the minimization of the empirical risk

$$\boldsymbol{\theta} \mapsto \frac{1}{L} \sum_{l=1}^{L} Q((\boldsymbol{x}_l, y_l), f_{\boldsymbol{\theta}}) \,, \tag{8.8}$$

using gradient descent until reaching a local optimum, given a criterion Q. We focus here on *stochastic* gradient descent, which gives good results in time and training performance for large data sets (LeCun et al., 1998). Concerning the criterion Q, we mentioned in Chapter 3 that the most often used ones are the Mean Squared Error (MSE) and the Cross-Entropy (CE) criteria. We also pointed out that from a probabilistic point of view, the CE criterion is more suited for classification. Finally, according to Chapter 7, the CE criterion leads to a simpler optimization problem than the MSE criterion. Thus, it is better to choose the CE criterion for classification, which yields

$$Q((\boldsymbol{x}, y), f_{\boldsymbol{\theta}}) = \log(1 + \exp(-y\, f_{\boldsymbol{\theta}}(\boldsymbol{x}))),$$

in (8.8), according to derivations made in Chapter 3. In order to avoid overfitting, we also highlighted that two methods are often used while training MLPs: either we perform early stopping (that is, we stop the training process before reaching a local optimum) or we add regularization terms over the parameters of the model, which leads to the minimization of

$$\boldsymbol{\theta} \mapsto \frac{\mu}{2}\,\|\boldsymbol{w}\|^2 + \frac{\nu}{2}\sum_{n=1}^{N}\|\boldsymbol{v}_n\|^2 + \frac{1}{L}\sum_{l=1}^{L}\log(1 + \exp(-y_l\, f_{\boldsymbol{\theta}}(\boldsymbol{x}_l))). \tag{8.9}$$

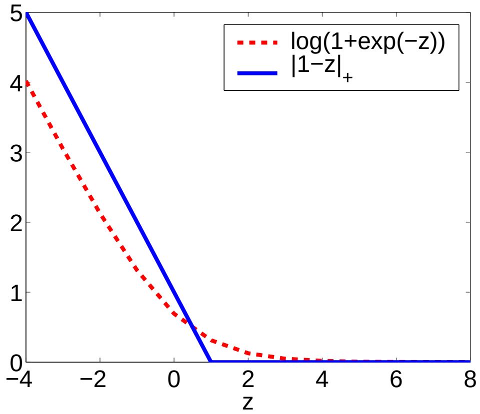
Regularization is then controlled by tuning the weight decay parameters μ and ν. Note that the second term is not present when training Perceptrons or Φ-machines. Let us now point out that the CE criterion $z \mapsto \log(1 + \exp(-z))$ used for MLPs is a "smooth" version of the "hard" margin criterion $z \mapsto |1 - z|_+$ used for SVMs, as shown in Figure 8.1. Moreover, it has been shown recently by Rosset et al. (2004) that the use of the CE criterion was related to the maximization of the margin. Thus, we make the assumption here that the difference between the "smooth" and the "hard" criteria is not crucial to understand the behavior of MLPs. We will therefore consider in the rest of this chapter the "hard" version of the criterion. Hence, instead of minimizing (8.9), we will consider the minimization of

$$\boldsymbol{\theta} \mapsto \frac{\mu}{2}\|\boldsymbol{w}\|^2 + \frac{\nu}{2}\sum_{n}\|\boldsymbol{v}_n\|^2 + \frac{1}{L}\sum_{l}|1 - y_l\, f_{\boldsymbol{\theta}}(\boldsymbol{x}_l)|_+ . \tag{8.10}$$

With this criterion, it appears obvious that Φ-machines are equivalent to SVMs and Perceptrons are equivalent to linear SVMs. Only the training method remains different: Perceptrons are trained by applying stochastic gradient descent on (8.6), whereas SVMs are trained by minimizing a quadratic problem under constraints, as shown in Chapter 5.

REF Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In G.B. Orr and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, pages 9–50. Springer, 1998.

REF S. Rosset, J. Zhu, and T. Hastie. Margin maximizing loss functions. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.



▲ **Figure 8.1.** *The CE criterion* $z \mapsto \log(1 + \exp(-z))$ *versus the margin criterion* $z \mapsto |1 - z|_+$.

On Stochastic Gradient With Non-Differentiable Cost

Using the stochastic gradient method to minimize the cost (8.10) is debatable because the function $z \mapsto |1-z|_+$ is not differentiable everywhere. We propose to study this problem, using Theorem 3.2 stated by Bottou (1998) which proves the convergence of the stochastic gradient descent method. Keeping the same notations introduced for this theorem, we consider the stochastic update

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \lambda_t \, G_{\boldsymbol{\theta}}(\boldsymbol{x}_l, \, y_l) \,,$$

where we define here $G_{\boldsymbol{\theta}}(\cdot)$ as

$$G_{\boldsymbol{\theta}}(\boldsymbol{x}, \, y) = \begin{cases} \dfrac{\partial Q((\boldsymbol{x}, \, y), \, f_{\boldsymbol{\theta}})}{\partial \boldsymbol{\theta}} & \text{if } Q(\cdot, \, f_{\boldsymbol{\theta}}) \text{ is differentiable in } (\boldsymbol{x}, \, y) \\ \mathbf{0} & \text{otherwise}. \end{cases} \tag{8.11}$$

We can now apply Theorem 3.2 if the assertion *ii*) remains true:

$$\int G_{\boldsymbol{\theta}}(\boldsymbol{x}, \, y) \, P(\boldsymbol{x}, \, y) \, d(\boldsymbol{x}, \, y) = \frac{\partial R(f_{\boldsymbol{\theta}})}{\partial \boldsymbol{\theta}} \,, \tag{8.12}$$

where $P(\boldsymbol{x}, \, y)$ is the true (unknown) distribution of the data and the expected risk $R(\cdot)$ is defined by

$$\boldsymbol{\theta} \mapsto \int Q((\boldsymbol{x}, \, y), \, f_{\boldsymbol{\theta}}) \, P(\boldsymbol{x}, \, y) \, d(\boldsymbol{x}, \, y) \,,$$

(where we removed the regularization terms in order to simplify notations) as already introduced in Chapter 3. According to definition (8.11) and Lebesgue's

REF L. Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

dominated convergence theorem, the validity of assertion (8.12) is ensured if

$$\theta \mapsto Q((\boldsymbol{x}, y), f_{\boldsymbol{\theta}}) \tag{8.13}$$

is bounded by an integrable function and differentiable everywhere *except for a set of* $(\boldsymbol{x}, y)$ *of measure zero*. This is the case for Perceptrons and MLPs using the margin criterion (8.6), where (8.13) is only undifferentiable on the manifolds defined by

$$\boldsymbol{w} \cdot \Phi(\boldsymbol{x}) + b = \pm 1 \,,$$

i.e. on a set of measure zero. Thus, the *expected* risk remains differentiable even if the "local" criterion Q is not differentiable everywhere. With an *infinite* number of examples, Theorem 3.2 guarantees that stochastic gradient descent minimizes the *expected* risk.

With a *finite* number of examples, if Q is not differentiable everywhere, the *empirical* risk *is not* differentiable as well. We can no longer apply Theorem 3.2 to the empirical risk, as we did in Chapter 3 using a counting measure. Note that recently Ermoliev and Norkin (1997) proposed an extension of Theorem 3.2 for a class of *generalized differentiable* functions. In particular, they highlight the fact that compositions of differentiable functions with "min" and "max" functions are generalized differentiable, which is the case in this chapter. Unfortunately, their theorem requires the first derivative of the risk to be bounded by a *constant*, which is not the case in general (except for the Simple MLP algorithm that we propose at the end of this chapter). Thus, we cannot say in general that the stochastic gradient descent method will converge to an extremal point of the empirical risk. However, with "enough" training examples we are ensured to approach an extremal point of the expected risk (Bottou, 1998).

Note on the Original Perceptron Algorithm

Note that the original Perceptron algorithm as introduced by Rosenblatt (1957) and presented in Chapter 3 is equivalent to using the criterion

$$Q((\boldsymbol{x}, y), f_{\boldsymbol{\theta}}) = | - y\, f_{\boldsymbol{\theta}}(\boldsymbol{x})|_{+} \,, \tag{8.14}$$

REF Y. M. Ermoliev and V. I. Norkin. Stochastic generalized gradient method with application to insurance risk management. Technical Report IR-97-021, International Institute for Applied Systems Analysis, 1997.

REF L. Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

REF F. Rosenblatt. The perceptron: a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, N.Y., 1957.

with stochastic gradient descent. Because of the definition of the algorithm which updates the weights *only* if an example is misclassified, the hyperplane found by the original Perceptron algorithm can be very close to the data. It is thus not possible to talk about margins with this algorithm, and this algorithm is therefore not our main interest in this chapter. Moreover, we will show that it is not easy to control the generalization ability of this model: if we want to add a regularization parameter $\|\boldsymbol{w}\|^2$ in the cost (8.14) (as for other models) which yields

$$\boldsymbol{\theta} \mapsto \frac{\mu}{2}\,\|\boldsymbol{w}\|^2 + \frac{1}{L}\sum_{l=1}^{L} |-y_l\,(\boldsymbol{w}\cdot\boldsymbol{x}_l + b)|_+\,,$$

it is easy to see that the minimization of this cost leads to a hyperplane with zero weights. Indeed, if we apply the same derivations as for SVMs, it is easy to see that the minimization of (8.1.2) is equivalent to minimizing

$$\boldsymbol{\alpha} \mapsto \boldsymbol{\alpha}^{\mathrm{T}}\,K\boldsymbol{\alpha}$$

under the constraints

$$\sum_{l=1}^{L} \alpha_l\,y_l = 0 \;\text{ and }\; 0 \le \alpha_l \le \frac{1}{L} \quad \forall l\,,$$

where the matrix K is given by

$$K_{lm} = y_l\,y_m\,\boldsymbol{x}_l \cdot \boldsymbol{x}_m\,.$$

The KKT conditions are the same as the one given for SVMs (8.7), except

$$\alpha_l\,[1 - \xi_l - y_l\,(\boldsymbol{w}\cdot\Phi(\boldsymbol{x}_l) + b)] = 0 \qquad \forall l$$

which becomes

$$\alpha_l\,[-\xi_l - y_l\,(\boldsymbol{w}\cdot\Phi(\boldsymbol{x}_l) + b)] = 0 \qquad \forall l\,.$$

It is obvious that the KKT conditions are satisfied if all parameters are zero. A regularization parameter with the original Perceptron algorithm is thus useless, and we can only rely on early stopping for controlling the generalization ability of the algorithm.

Stochastic Gradient With Weight Decay

We just showed that Φ-machines (respectively Perceptrons) trained with the hard margin criterion (8.6) are equivalent to SVMs (respectively linear SVMs). We pointed out that only the training method remains different: Perceptrons are trained by stochastic gradient descent on (8.6), whereas SVMs are trained by minimizing a quadratic problem under constraints, as shown in Chapter 5. Therefore, we would like to see what happens when training Perceptrons with stochastic gradient descent, using what we know about SVMs. We will consider the two main solutions used to control the generalization ability of a Perceptron; first the use of a regularization term in the cost function (as for SVMs), and then the use of early stopping instead. We first consider the use of a regularization term in this section: we focus on the cost function (8.6) in the linear case, which can be rewritten as:

$$(\boldsymbol{w},\, \boldsymbol{b}) \mapsto \frac{\mu}{2}\,\|\boldsymbol{w}\|^2 + \frac{1}{L}\sum_{l=1}^{L} |1 - y_l\,(\boldsymbol{w}\cdot\boldsymbol{x}_l + b)|_+\,. \tag{8.15}$$

For a given training example $(\boldsymbol{x}_l,\, y_l)$, the stochastic update for the weight vector w is easily computed as:

$$\boldsymbol{w} \leftarrow \begin{cases} (1-\lambda\mu)\,\boldsymbol{w} + \lambda\, y_l\, \boldsymbol{x}_l & \text{if } y_l\,(\boldsymbol{w}\cdot\boldsymbol{x}_l + b) \le 1 \\ (1-\lambda\mu)\,\boldsymbol{w} & \text{otherwise} \end{cases} \tag{8.16}$$

given a learning rate λ. From (8.7), we know that the optimal $\boldsymbol{w}$ is a linear combination of training examples. Therefore, the update (8.16) is equivalent to the following update written in the dual space:

$$\forall k,\quad \alpha_k \leftarrow \begin{cases} (1-\lambda\mu)\,\alpha_l + \lambda\mu & \text{if } k = l \\ & \text{and } y_l\,(\boldsymbol{w}\cdot\boldsymbol{x}_l + b) \le 1 \\ (1-\lambda\mu)\,\alpha_k & \text{otherwise}\,. \end{cases} \tag{8.17}$$

A study of stochastic gradient descent in the dual space has already been presented by Kivinen et al. (2002), who re-discovered gradient descent in the case of SVMs. However, we would like here to use the formulation (8.17) to give a lower bound on the number of iterations needed to train a Perceptron (or a linear SVM) with stochastic gradient descent. Let us consider now a training example $(\boldsymbol{x}_l,\, y_l)$ which becomes useless at some point during training. We note $\alpha_l^\star$ the value of its corresponding weight at this point of the training. According to what we said in Chapter 3, α_l has to be decreased from $\alpha_l^\star$ to zero if it is not a support vector. Using (8.17), it appears that in the optimistic case α_l will be equal to

$$(1-\lambda\mu)^{L\,N_e}\,\alpha_l^\star$$

after N_e epochs over the whole training set. Thus, just dividing $\alpha_l^\star$ by two will require N_e epochs, where N_e satisfies the equation

$$N_e \log(1-\lambda\mu) = -\frac{\log(2)}{L}\,.$$

REF J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14, pages 785–792. MIT Press, 2002.

For small values of $\lambda\mu$, an equivalent of N_e is therefore given by

$$N_e \sim \frac{\log(2)}{L\lambda\mu} .$$

With a similar analysis, if the l-th example is going to be misclassified, then it has to be a support vector at bound, which means that its corresponding weight α_l has to be increased until reaching $1/L$. In the optimistic case, according to the update (8.16), we can compute (for $N_e \geq 1$)

$$\alpha_l = \lambda\mu \sum_{i=0}^{N_e-1} (1-\lambda\mu)^{L\,i} \quad = \lambda\mu \, \frac{1-(1-\lambda\mu)^{L\,N_e}}{(1-\lambda\mu)^L} ,$$

after N_e epochs over the whole training set (if α_l is first initialized to 0). If α_l needs to reach $1/L$, N_e has to satisfy the equation

$$N_e \log(1-\lambda\mu) = \frac{1}{L} \log\left[1 - \frac{1-(1-\lambda\mu)^L}{L\lambda\mu}\right] .$$

For small $\lambda\mu$ we obtain an equivalent of the necessary number of iterations N_e over the training set:

$$N_e \sim -\frac{\log(\lambda\mu)}{L\lambda\mu} .$$

This leads to the following theorem:

Theorem 8.1 *Training a Perceptron using a weight decay (or a linear SVM) with stochastic gradient descent requires at least on the order of $O(\frac{1}{L\lambda\mu})$ epochs over the whole training set to reach the true solution.*

Thus, when the learning rate λ and the weight decay parameter μ are small, which is often the case, it quickly becomes computationally expensive to reach the true SVM solution with stochastic gradient descent. A typical example is given for $\lambda\mu = 10^{-7}$ and $L = 10^4$, which requires at least 10^3 iterations (These values correspond to the best generalization performance using 10^4 examples on the Forest and Connect-4 databases presented in Chapter 4).

Stochastic Gradient With Early Stopping

We already highlighted that control of the generalization ability of a Perceptron is usually achieved in two different ways: the use of a regularization term in the cost function, and the use of early stopping. Early stopping halts training before reaching a local optimum, usually according to an estimate of the generalization error on a "validation" set. Let us now consider Perceptrons trained (using stochastic gradient descent) by minimizing the cost function (8.15) without the regularization term:

$$(\boldsymbol{w}, b) \mapsto \frac{1}{L} \sum_{l=1}^{L} |1 - y_l (\boldsymbol{w} \cdot \boldsymbol{x}_l + b)|_+ . \tag{8.18}$$

We will now show that in this framework the margin of a Perceptron can still be controlled both by tuning the learning rate used in the gradient descent procedure and by early stopping. As the margin size controls the capacity and thus the generalization performance, as pointed out in Chapter 3, it gives a justification for not using a regularization term.

Deriving stochastic gradient descent equations for the cost function in (8.18) leads to Algorithm 8.1, that we call the "Margin Perceptron" algorithm. Note

Algorithm 8.1 Margin Perceptron Algorithm

Initialize $\boldsymbol{w}$ and b to zero
repeat
  for $l \in \{1 \dots L\}$ **do**
    if $y_l (\boldsymbol{w} \cdot \boldsymbol{x}_l + b) \leq 1$ **then**
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \lambda \, y_l \, \boldsymbol{x}_l$
      $b \leftarrow b + \lambda \, y_l$
    end if
  end for
until termination criterion

that this algorithm is not new, and has even been proposed by Duda and Hart (1973), as a variant of the original Perceptron training algorithm presented in Chapter 3.

Linearly Separable Classes

We first want to consider this algorithm for case in which the classes in the training set are linearly separable. We first consider the case where the bias b is fixed to zero. In this case, we consider the separating hyperplane

$$\boldsymbol{x} \mapsto \boldsymbol{u} \cdot \boldsymbol{x} \tag{8.19}$$

such that $y_l (\boldsymbol{u} \cdot \boldsymbol{x}_l) \geq 1 \; \forall l$, which has the maximal margin $\rho_{\max} = \frac{2}{\|\boldsymbol{u}\|}$. We then derive equations similar to the well-known Novikoff theorem (1962) for

REF R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley & Sons, New York, 1973.

REF A. B. J. Novikoff. On convergence proofs on perceptrons. In Polytechnic Institute of Brooklyn, editor, *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622, 1962.

the original Perceptron convergence proof. We denote the weight vector of the Perceptron after t updates in Algorithm 8.1 as $\boldsymbol{w}_t$, and the index of the example updated at t as l_t. First, we have

$$\begin{aligned} \boldsymbol{u}\cdot\boldsymbol{w}_t &= \boldsymbol{u}\cdot\boldsymbol{w}_{t-1} + \lambda\, y_{l_t}\, \boldsymbol{u}\cdot\boldsymbol{x}_{l_t} \\ &\geq \boldsymbol{u}\cdot\boldsymbol{w}_{t-1} + \lambda \\ &\geq t\,\lambda\,. \end{aligned} \tag{8.20}$$

Moreover, if we consider R the radius of the training data (that is, $\|\boldsymbol{x}_l\| \leq R \;\; \forall l \in \{1 \ldots L\}$), we have

$$\begin{aligned} \|\boldsymbol{w}_t\|^2 &= \|\boldsymbol{w}_{t-1}\|^2 + 2\,\lambda\, y_{l_t}\, \boldsymbol{w}_{t-1}\cdot\boldsymbol{x}_{l_t} + \lambda^2 \|\boldsymbol{x}_{l_t}\|^2 \\ &\leq \|\boldsymbol{w}_{t-1}\|^2 + 2\,\lambda + \lambda^2 R^2 \\ &\leq t\,\lambda^2\,(\tfrac{2}{\lambda} + R^2)\,. \end{aligned} \tag{8.21}$$

Using (8.20) and (8.21), we obtain with the Cauchy-Schwarz inequality:

$$\begin{aligned} t\,\lambda &\leq \boldsymbol{u}\cdot\boldsymbol{w}_t \\ &\leq \|\boldsymbol{u}\|\,\|\boldsymbol{w}_t\| \\ &\leq \tfrac{2}{\rho_{\max}}\,\lambda\,\sqrt{t}\,\sqrt{\tfrac{2}{\lambda} + R^2} \end{aligned} \tag{8.22}$$

which leads to an upper bound on the number of updates:

$$t \leq \frac{4}{\rho_{\max}^2}\left(\frac{2}{\lambda} + R^2\right)\,. \tag{8.23}$$

This shows that the Margin Perceptron converges in a finite number of updates for the linearly separable case, which has already been demonstrated (Duda and Hart, 1973). However, if we introduce (8.23) into (8.21), we are able to compute a lower bound on the margin $\rho = \frac{2}{\|\boldsymbol{w}_t\|}$ found by the Margin Perceptron:

$$\rho \geq \rho_{\max}\frac{1}{2 + R^2\,\lambda}\,. \tag{8.24}$$

We can generalize this to the case of separating hyperplanes with biases by performing the following substitutions in the previous equations

$$\begin{aligned} \boldsymbol{x}_l &\leftarrow (\boldsymbol{x}_l,\, 1) \\ \boldsymbol{w} &\leftarrow (\boldsymbol{w},\, b) \\ \boldsymbol{u} &\leftarrow (\boldsymbol{u},\, a)\,, \end{aligned} \tag{8.25}$$

REF R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley & Sons, New York, 1973.

where b is the bias of the Perceptron hyperplane and a is the bias of the maximal margin hyperplane (8.19). Substituting (8.25) into (8.24) leads to

$$\begin{aligned} \rho &= \frac{2}{\|\boldsymbol{w}_t\|} \\ &\geq \frac{2}{\sqrt{\|\boldsymbol{w}_t\| + b^2}} \\ &\geq \frac{2}{\sqrt{\|\boldsymbol{u}\|^2 + a^2}} \, \frac{1}{2 + (R+1)^2\,\lambda}\,. \end{aligned} \tag{8.26}$$

Considering KKT conditions (8.7) (for a maximal margin hyperplane) applied to a support vector $(\boldsymbol{x}_+, y_+)$ of the positive class and a support vector $(\boldsymbol{x}_-, y_-)$ of the negative class (not at bounds), we can state

$$\begin{aligned} 1 - \boldsymbol{u}\cdot\boldsymbol{x}_+ - a &= 0 \\ 1 + \boldsymbol{u}\cdot\boldsymbol{x}_- + a &= 0\,. \end{aligned} \tag{8.27}$$

Combining equalities (8.27) and applying Cauchy-Schwarz inequality yields

$$\begin{aligned} |a| &= \frac{1}{2}|\,\boldsymbol{u}\cdot(\boldsymbol{x}_+ + \boldsymbol{x}_-)\,| \\ &\leq \frac{1}{2}\|\boldsymbol{u}\|\,(\|\boldsymbol{x}_+\| + \|\boldsymbol{x}_-\|) \\ &\leq \|\boldsymbol{u}\|\,R\,. \end{aligned}$$

Applying this bound to (8.26) allows us to state the following theorem:

Theorem 8.2 *If the classes are linearly separable, the Margin Perceptron algorithm will converge in a finite number of iterations, and its final margin ρ will satisfy*

$$\rho \;\geq\; \rho_{\max}\,\frac{1}{\sqrt{1+R^2}}\,\frac{1}{2+R^2\,\lambda}\,.$$

Therefore, the smaller the learning rate, the larger the margin of the Perceptron. Note that Graepel et al. (2001) already established a relation between the existence of a large margin classifier and the sparseness in the dual space of the solutions found by the original Perceptron algorithm. Here, in the case of the Margin Perceptron algorithm, we instead relate the margin found by the Perceptron to the largest existing margin.

General Case

In the non-separable case, we would like to be able to control the margin (that is $\rho = \frac{2}{\|\boldsymbol{w}\|}$), while minimizing the number of errors in the margin (8.18). SVMs

REF T. Graepel, R. Herbrich, and R. Williamson. From margin to sparsity. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13, pages 210–216. MIT Press, 2001.

control the trade-off between these two quantities through the weight decay parameter μ in the cost function (8.15). By design, the Margin Perceptron minimizes the number of errors in the margin through training iterations. Using (8.21) with (8.25), the evolution of the margin can be described using the following theorem:

Theorem 8.3 *After t updates, the margin ρ_t of the Margin Perceptron is bounded by the following:*

$$\rho_t \geq \frac{2/\lambda}{\sqrt{t(2/\lambda + (R+1)^2)}} \,.$$

Thus, the margin increases with small number of updates or a low learning rate. As the number of updates and the learning rate control the margin, they also control the generalization performance. Finally, since early stopping limits the number of updates, Theorem 8.3 justifies early stopping. We give a practical example of controlling the margin using the learning rate and the number of updates in Figure 8.2. We also give an illustration in Figure 8.3 which clearly shows the advantage of early stopping. Note that we give a comparison with the original Perceptron algorithm Algorithm 3.1, which is much harder to control. Indeed, because of the lack of the margin, the original Perceptron hyperplane tends to be close to the data, and is thus not very robust to noise.

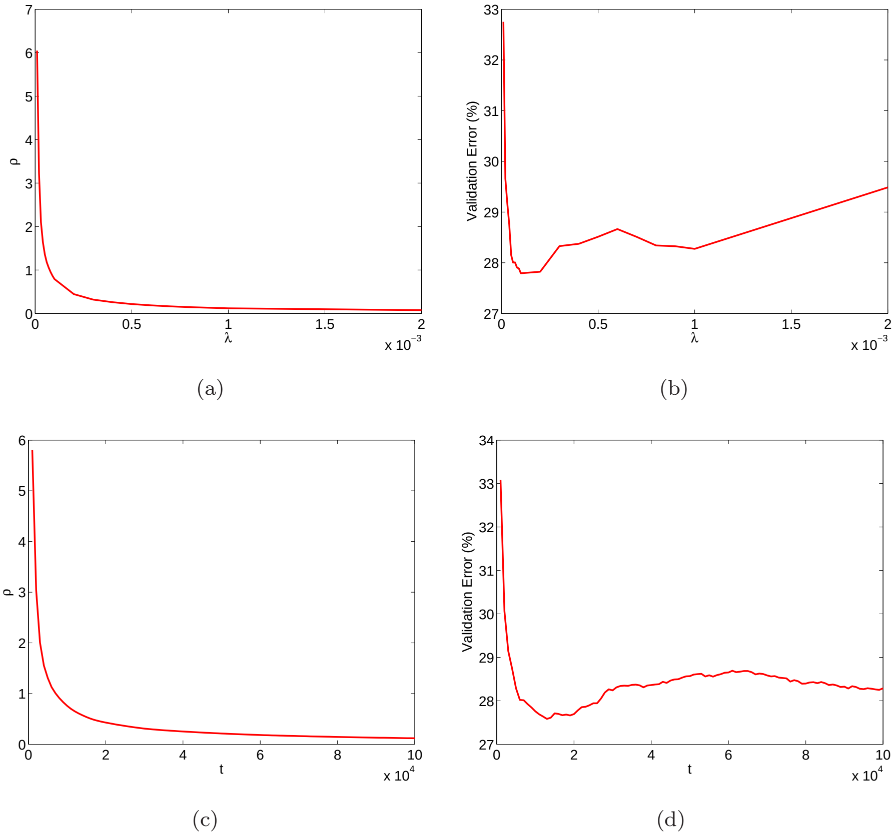
Extension to Non-Linear Models

Theorem 8.1 is still valid without restriction if we choose an arbitrary Φ function which maps the input space into an arbitrary space, and if we consider examples $(\Phi(\boldsymbol{x}_l),\, y_l)$ instead of $(\boldsymbol{x}_l,\, y_l)$. Theorem 8.2 and Theorem 8.3 remain valid as long as there exists a constant $R \in \mathbb{R}$ such as $\|\Phi(\boldsymbol{x}_l)\| \leq R \;\forall l \in \{1 \dots L\}$. Thus, if the constraint is respected, all results apply to Φ-machines and non-linear SVMs, including SVMs with kernels (a kernel being an inner product in an arbitrary space described by a function Φ, as stated in Chapter 3).

Extension to MLPs

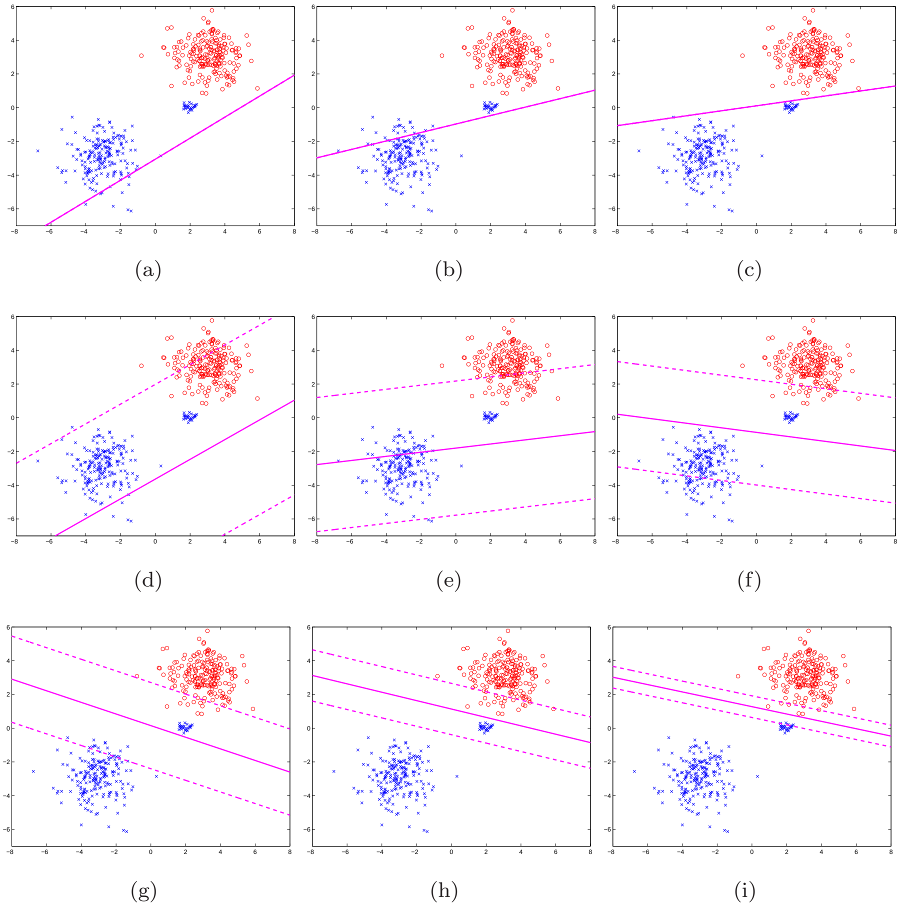
We consider now the minimization of (8.10), where the function f is an MLP with one hidden layer, described by (8.1) and (8.2). First, we introduce the notion of margin in MLPs. As for SVMs, the MLPs minimization problem (8.10) can be rewritten as the minimization of

$$J \,:\, (\boldsymbol{\theta},\, \boldsymbol{\xi}) \mapsto \frac{\mu}{2}\,\|\boldsymbol{w}\|^2 + \frac{\nu}{2}\sum_{n=1}^{N}\|\boldsymbol{v}_n\|^2 + \frac{1}{L}\sum_{l=1}^{L}\xi_l \qquad (8.28)$$



(a) (b) (c) (d)

▲ **Figure 8.2.** *Practical examples of the control of the margin, with the Margin Perceptron algorithm. Figures (a) and (b) represent the margin ρ and the validation error with respect to the learning rate λ respectively, (when the number of updates is fixed). Figures (c) and (d) represent the margin ρ and the validation error with respect to the number of updates t respectively, (when the learning rate λ is fixed). Results were obtained with 200 training examples on the Connect-4 database, averaged on 10 trials.*



(a) (b) (c) (d) (e) (f) (g) (h) (i)

▲ **Figure 8.3.** *Visual examples showing the evolution of the separating hyperplane and its margin during training with the Margin Perceptron. The goal is to separate two classes: crosses (blue) and circles (red). We have added some noisy crosses near (just below) the circles. Figures (a), (b) and (c) show the hyperplane found with the original Perceptron algorithm until convergence after* 10, 40, *and* 60 *updates of the parameters. It is not possible to control the margin, and the hyperplane is not robust to noise. Figures (d)–(i) show the hyperplane found by the Margin Perceptron algorithm, after* 10, 40, 60, 120, 500, *and* 2000 *updates of the parameters. Dashed lines represents the margin of the hyperplane. Early stopping thus allows us to choose solution (g), which is robust to noise.*

under the constraints

$$\forall l \;\; \varphi_l(\boldsymbol{\theta},\, \boldsymbol{\xi}) \leq 0\,, \tag{8.29}$$

where φ_l is defined as in (8.5). Even if J and the constraints φ_l are no longer convex, the KKT constraints are still *necessary* for a minimum of (8.28) to exist, as stated by Theorem A.1. Thus, if $(\boldsymbol{\theta},\, \boldsymbol{\xi})$ is a minimum of (8.28), there exists $(\boldsymbol{\alpha},\, \boldsymbol{\eta})$ such that the KKT constraints are satisfied. The KKT conditions are the same as SVM conditions (8.7), plus the following:

$$\begin{aligned} \boldsymbol{v}_n &= \frac{w_n}{\nu} \sum_{l=1}^{L} y_l\, \alpha_l\, h'(\boldsymbol{v}_n \cdot \boldsymbol{x}_l + a_n)\, \boldsymbol{x}_l \quad \forall n \\ \sum_{l=1}^{L} y_l\, h'(\boldsymbol{v}_n & \cdot \boldsymbol{x}_l + a_n)\, \alpha_l = 0 \quad \forall n\,. \end{aligned} \tag{8.30}$$

Uniqueness of Lagrange Multipliers

Note that if $(\boldsymbol{\theta},\, \boldsymbol{\xi})$ is a minimum of (8.28), then the corresponding Lagrange multipliers $(\boldsymbol{\alpha},\, \boldsymbol{\eta})$ which satisfy the KKT conditions are unique. This is due (as stated by Theorem A.1) to the linear independence of the gradients $\partial\varphi_l / \partial\{\boldsymbol{\theta}, \boldsymbol{\xi}\}$ of the constraints (8.5). Indeed, in our case, for $1 \leq l \leq L$, we have

$$\forall k \;\; \frac{\partial \varphi_l}{\partial \xi_k} = -\delta_{lk} \;\; \text{and} \;\; \frac{\partial \varphi_l}{\partial b} = 0\,,$$

(where $\delta_{lk} = 0$ if $k \neq l$, and $\delta_{lk} = 1$ if $k = l$) and

$$\forall k \;\; \frac{\partial \varphi_{L+l}}{\partial \xi_k} = -\delta_{lk} \;\; \text{and} \;\; \frac{\partial \varphi_{L+l}}{\partial b} = -y_l \neq 0\,,$$

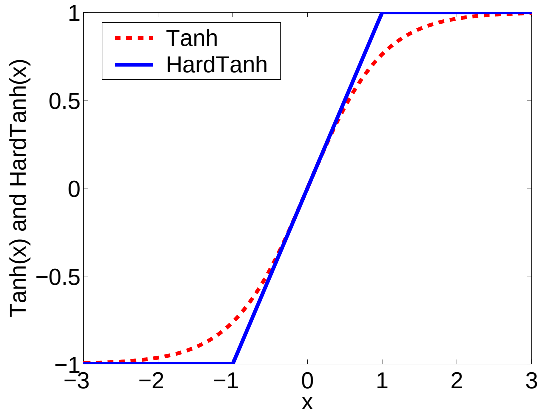
which guarantees the independence of the gradients.

Margin in the Hidden Layer Space

Let us consider an MLP which is an optimum of the minimization problem (8.28) under the constraints (8.29), with the output parameters $(\boldsymbol{w}^\star,\, b^\star)$ and the hidden layer $\Phi^\star$ described with the optimal hidden weights $\boldsymbol{v}_n^\star$ and $a_n^\star$ by

$$\Phi_n^\star(\boldsymbol{x}) = h(\boldsymbol{v}_n^\star \cdot \boldsymbol{x} + a_n^\star)\,.$$

As the KKT conditions of the MLP problem are necessary for the existence of an optimum, there exists Lagrange multipliers $(\boldsymbol{\alpha}^\star,\, \boldsymbol{\eta}^\star)$ such that they are satisfied. As KKT conditions of the SVM problem are a subset of KKT conditions of the MLP problem, note that SVM KKT conditions are satisfied with $(\boldsymbol{\alpha}^\star,\, \boldsymbol{\eta}^\star)$ for the weights $(\boldsymbol{w}^\star,\, b^\star)$ and the hidden layer $\Phi^\star$. Thus, it appears



▲ **Figure 8.4.** *The hyperbolic tangent function (Tanh) and the "hard" hyperbolic tangent function (HardTanh).*

that $(\boldsymbol{w}^\star, b^\star)$ are the optimal weights for the SVM problem (8.6) using the feature space described by $\Phi^\star$. In other words, the optimal MLP *maximizes the margin in its hidden layer space.*

Margin in the Input Space

We can give a margin interpretation of hidden layer units as well, if we first perform another simplification: instead of using a hyperbolic tangent for the transfer function h, we consider the following "hard" version of h (also drawn in Figure 8.4):

$$h(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 \leq x \leq 1 \\ 1 & \text{if } x > 1\,. \end{cases} \tag{8.31}$$

Note that the use of this transfer function still maintains the universal approximation property of MLPs, as shown by Hornik et al. (1989). One could argue that we loose the smooth non-linearity of the hyperbolic tangent which could reduce the approximation capabilities of MLPs, but in practice, it leads to performance comparable to a standard MLP for the same number of hidden units. Note that even if $h(\cdot)$ is undifferentiable in -1 and 1, $Q(\cdot, f_{\boldsymbol{\theta}})$ remains differentiable *almost* everywhere, and we can justify stochastic gradient descent using Theorem 3.2, as we did previously. Ignoring the non-differentiability in

REF K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

-1 and 1, the first derivative of $h(\cdot)$ can be computed as follows:

$$h'(z) = \begin{cases} 1 & \text{if } |z| \leq 1 \\ 0 & \text{otherwise} . \end{cases}$$

Thus, given an optimal MLP, the sums in (8.30) consider only examples $(\boldsymbol{x}_l,\, y_l)$ such that $|\boldsymbol{v}_n^\star \cdot \boldsymbol{x}_l + a_n^\star| \leq 1$. Moreover, for these examples one can notice that for the n-th hidden unit of the MLP we have

$$\begin{aligned}
1 - \xi_l - y_l\, f_{\boldsymbol{\theta}}(\boldsymbol{x}_l) &= 1 - \xi_l - y_l \left[b + \sum_{m=1}^{N} w_k^\star\, h(\boldsymbol{v}_m^\star \cdot \boldsymbol{x}_l + a_m^\star) \right] \\
&= \left[1 - y_l \left(b + \sum_{m \neq n} w_m^\star\, h(\boldsymbol{v}_m^\star \cdot \boldsymbol{x}_l + a_m^\star) \right) \right] \\
&\quad - \xi_l - y_l\, w_n^\star\, h(\boldsymbol{v}_n^\star \cdot \boldsymbol{x}_l + a_n^\star) \\
&= \left[1 - y_l \left(b + \sum_{m \neq n} w_m^\star\, h(\boldsymbol{v}_m^\star \cdot \boldsymbol{x}_l + a_m^\star) \right) \right] \\
&\quad - \xi_l - y_l\, w_n^\star\, (\boldsymbol{v}_n^\star \cdot \boldsymbol{x}_l + a_n^\star) \\
&\leq 0 \,,
\end{aligned}$$

knowing that ξ_l has to satisfy the constraint (8.29). By verifying KKT conditions, the optimal weights $(\boldsymbol{v}_n^\star,\, a_n^\star)$ given by the optimal MLP for the n-hidden unit are a solution of the SVM problem (8.3) for the examples $(\boldsymbol{x}_l,\, y_l)$ such that $|\boldsymbol{v}_n^\star \cdot \boldsymbol{x}_l + a_n^\star| \leq 1$, and where the "standard" SVM constraints

$$y_l\, (\boldsymbol{v}_n \cdot \boldsymbol{x}_l + a_n) \geq 1 - \xi_l$$

are replaced with the constraints

$$y_l\, w_n^\star\, (\boldsymbol{v}_n \cdot \boldsymbol{x}_l + a_n) \geq \left[1 - y_l \left(b + \sum_{m \neq n} w_m^\star\, h(\boldsymbol{v}_m^\star \cdot \boldsymbol{x}_l + a_m^\star) \right) \right] - \xi_l \,.$$

Summary

Our analysis leads to the following theorem:

Theorem 8.4 *If an MLP is a minimum of (8.10), then it maximizes the margin in the hidden layer space. In other words, $(\boldsymbol{w},\, b)$ is solution of the SVM problem (8.3) for the hidden layer Φ found by the MLP.*

Moreover, if we use the hard transfer function (8.31), then the n-th hidden unit is solution of a local SVM on examples $\boldsymbol{x}_l$ which satisfy $|\boldsymbol{v}_n \cdot \boldsymbol{x}_l + a_n| \leq 1$.

For this SVM, the standard separation constraints $y_l\,(\boldsymbol{v}_n \cdot \boldsymbol{x}_l + a_n) \geq 1$ are replaced by the constraints

$$y_l\, w_n\,(\boldsymbol{v}_n \cdot \boldsymbol{x}_l + a_n) \geq 1 - y_l \left[b + \sum_{m \neq n} w_m\, h(\boldsymbol{v}_m \cdot \boldsymbol{x}_l + a_m) \right] . \tag{8.32}$$

Finally, if $(\Phi(\boldsymbol{x}_l),\, y_l)$ is a "global" support vector for $\boldsymbol{w}$, then $(\boldsymbol{x}_l,\, y_l)$ will be a "local" support vector for the n-th unit if $|\boldsymbol{v}_n \cdot \boldsymbol{x}_l + a_n| \leq 1$.

This theorem has several important consequences. First, it shows that an optimal MLP for the cost function (8.10) is an SVM in the hidden layer space. It is therefore possible to apply the large margin classifier theory (Vapnik, 1995) to MLPs. It also provides an interesting interpretation of the role of the hidden units: each hidden unit focuses on a subset of the training set, and is a kind of local SVM, where the constraints depend on the classification performance of the other hidden units. Note also that the output weight vector $\boldsymbol{w}$ is a sparse combination of the features $\Phi(\boldsymbol{x}_l)$, whereas the hidden unit weights $\boldsymbol{v}_n$ are a sparse combination of the training examples $\boldsymbol{x}_l$.

Weight Decay or Early Stopping

In light of Theorem 8.4, we can now easily extend Theorem 8.1 and Theorem 8.3 to MLPs. With weight decays in the cost function (8.10), Theorem 8.1 is still valid for MLPs, because we can apply the same derivations to the hidden units. If we remove the weight decays, Theorem 8.3 is valid for controlling the "global" margin $\rho = \frac{2}{\|w\|}$ using early stopping or the learning rate. This is because we can derive the bound

$$\|\Phi(\boldsymbol{x})\| \leq N\,,$$

which does not depend on the parameters of the hidden layer, which change through the iterations. For hidden units, it is not possible to derive the margin, because constraints (8.32) are different for each training example; however, using derivations similar to those used for Theorem 8.3, it is possible to control the norms $\|v_n\|$.

Too Many Hyper-Parameters Remain

Still, we are left with too many hyper-parameters to select with this kind of MLP in practice: the weight decay parameters μ and ν, the learning rate for the hidden layer, and the learning rate for the output layer. Tuning the two learning

REF V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, second edition, 1995.

rates is particularly difficult. Therefore, we propose another simplification in the next section.

A Very Simple MLP

It has been shown by Auer et al. (2002) that a Perceptron committee

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = b + \sum_{n=1}^{N} h(\boldsymbol{v}_n \cdot \boldsymbol{x} + a_n)\,. \tag{8.33}$$

passed through a particular kind of transfer function is a universal approximator. Without this particular output transfer function, the Perceptron committee (8.33) can thus solve any kind of binary classification problems, given enough hidden units. This shows that we do not need output weights in MLPs for binary classification. However, if we use the cost function (8.10), we can no longer control the size of the margin in the hidden layer space. Thus, we propose to minimize

$$J\,:\,\boldsymbol{\theta} \mapsto \frac{1}{L} \sum_{l} |\beta - y_l\, f_{\boldsymbol{\theta}}(\boldsymbol{x})|_{+} \tag{8.34}$$

instead, where β is a hyper-parameter to be tuned. As the margin ρ in the hidden layer space is fixed and satisfies

$$\rho = \frac{2\,\beta}{\sqrt{N}}\,,$$

it is easy to control using β. Note that Theorem 8.4 from the previous section remains valid, provided that $w_n = 1\ \forall n$. Each hidden unit acts as a local SVM, and it is possible to control the norms $\|v_n\|$ (and thus the sparseness of the optimal v_n in the dual space) using the learning rate and early stopping.

Efficient Algorithm

Deriving stochastic gradient descent equations for the cost function (8.34) leads to Algorithm 8.2. This algorithm is of interest, because hidden units will quickly focus on a few examples and few updates will be performed. Moreover, it is simple to implement, and as it does not require any internal storage (such as derivatives for example), nor any hyperbolic tangent computation. It is thus *computationally efficient*.

REF P. Auer, H. Burgsteiner, and W. Maass. Reducing communication for distributed learning in neural networks. In J. R. Dorronsoro, editor, *ICANN'2002*, volume 2415 of *Lecture Notes in Computer Science*, pages 123–128. Springer, 2002.

Algorithm 8.2 Simple MLP Algorithm

Initialize $\boldsymbol{w}$ and b to zero
repeat
  for $l \in \{1 \dots L\}$ **do**
    if $y_l\, f_{\boldsymbol{\theta}}(\boldsymbol{x}) \leq \beta$ **then**
      for $n \in \{1 \dots N\}$ **do**
        if $|\boldsymbol{v}_n \cdot \boldsymbol{x}_l + a_n| \leq 1$ **then**
          $\boldsymbol{v}_n \leftarrow \boldsymbol{v}_n + \lambda\, y_l\, \boldsymbol{x}_l$
          $a_n \leftarrow a_n + \lambda\, y_l$
        end if
      end for
      $b \leftarrow b + \lambda\, y_l$
    end if
  end for
until termination criterion

Note that the theory developed in Chapter 7 can be applied to the margin criterion. If we ignore the rare case $y_l\, f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \beta$ where the cost function is not differentiable, we can locally derive the Hessian of the cost (8.34):

$$\frac{\partial^2 J(\boldsymbol{\theta})}{\partial \boldsymbol{v}_m\, \partial \boldsymbol{v}_n} = 0 \quad (m \neq n)\ .$$

Thus, the Hessian of the cost function is *completely block-diagonal* with respect to a pair of hidden units, which guarantees local independence of the hidden units during training as stated in Chapter 7. From an optimization standpoint the margin criterion leads to a *mathematically efficient* stochastic gradient descent.

Experiments

We performed experiments on the Forest and Connect-4 data sets presented in Chapter 4. For SVMs we used a 300MB kernel evaluation cache on top of the memory needed for the SVM structure, whereas MLPs did not need any additional memory. The MLP and SVM hyper-parameters were carefully tuned using validation sets. Both the Standard MLP and the Simple MLP used early stopping based on the validation set. Results are given in Table 8.1. Clearly, the Simple MLP algorithm gives better generalization performance than the other models in much less time. SVMs are also very computationally expensive in the test phase compared to MLPs. Around 30% of the training examples are support vectors on both databases, whereas MLPs contain only 500 hidden

Model	Test Err. (%)		Time Factor	
	Forest	Connect-4	Forest	Connect-4
SVM	10.5	11.4	428.8	357.1
Standard MLP	8.9	11.4	29.1	11.1
Simple MLP	**7.3**	**10.1**	1.0	1.0

▲ **Table 8.1.** *Test errors of SVMs (with a Gaussian kernel), MLPs (trained with the CE criterion), and the proposed Simple MLP on two different classification tasks (Forest and Connect-4). The time factors give the training time relative to the Simple MLPs, for similar generalization performance. This shows that not only is the Simple MLP significantly better than MLPs and SVMs statistically (with 99% confidence), it is also significantly faster (from 9 to 400 times faster!).*

units. Finally, the fact that the Simple MLP leads to better performance than SVMs shows that the margin idea is not sufficient: the *feature space* is crucial as well. Conceptually, SVMs are as simple as Φ-machines: they do not train the feature space, whereas MLPs do.

Margin or No Margin, That is the Question

It is interesting to note that a variant of Algorithm 8.2 was already presented by Nilsson (1965) (first proposed by Ridgway, 1962) to train Perceptron committees. At this time, the author was not aware of the margin idea, and gradient descent techniques were not yet introduced in machine learning. Instead of using the hard version of the hyperbolic tangent for the transfer function in the committee (8.33), Nilsson used the sign function. He also removed the output bias. This leads us to consider his model:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{n=1}^{N} \operatorname{sign}(\boldsymbol{v}_n \cdot \boldsymbol{x} + a_n)\,.$$

Note that $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ must be an integer, because of the sign function. If we suppose the number of hidden units to be odd, and if an example $(\boldsymbol{x}_l,\, y_l)$ is misclassified, we need only modify $\frac{1}{2}[\,|f_{\boldsymbol{\theta}}(\boldsymbol{x}_l)|+1\,]$ units so they correctly classify the example. This ensures that the example will be well-classified by the committee. This yields Algorithm 8.3, which we call the *Nilsson MLP*. Note that hidden units

REF N. J. Nilsson. *Learning Machines*. McGraw-Hill, 1965.

REF W. C. Ridgway. An adaptive logic system with generalizing properties. Technical Report 1556-1, Standford University, 1962.

Algorithm 8.3 Nilsson MLP Algorithm

Initialize $\boldsymbol{w}$ and b to zero
repeat
  for $l \in \{1 \ldots L\}$ **do**
    if $y_l\, f_{\boldsymbol{\theta}}(\boldsymbol{x}) \leq 0$ **then**
      Sort $|\boldsymbol{v}_n \cdot \boldsymbol{x}_l + a_n|$ for misclassifying hidden units n (such that $y_l\,(\boldsymbol{v}_n \cdot \boldsymbol{x}_l + a_n) \leq 0$) in ascending order. Let o_n be the ordered indices.
      for $n \in \{\, 1 \ldots \frac{1}{2}[\, |f_{\boldsymbol{\theta}}(\boldsymbol{x}_l)| + 1\,]\,\}$ **do**
        $\boldsymbol{v}_{o_n} \leftarrow \boldsymbol{v}_{o_n} + \lambda\, y_l\, \boldsymbol{x}_l$
        $a_{o_n} \leftarrow a_{o_n} + \lambda\, y_l$
      end for
    end if
  end for
until termination criterion

modified when an example is misclassified by the committee miss-classify the example, and are "closest" to the example.

The Nilsson MLP algorithm is to the Simple MLP algorithm what the original Perceptron algorithm is to the Margin Perceptron algorithm: it does not include the notion of the margin. First, the fact that the weights are updated only if an example is misclassified corresponds to $\beta = 0$ for the Simple MLP (*i.e.* not having a global margin). The global margin in the Simple MLP insures that on average *more than half* of the hidden units output the correct class (if $\beta > 0$) instead of only *half* for Nilsson MLP algorithm. Thus, with a global margin, the Simple MLP adds confidence to the output of the committee. In addition, the fact that Nilsson uses the sign transfer function instead of the hard version of the hyperbolic tangent function forces each hidden unit to act locally as the original Perceptron instead of the Margin Perceptron. The Nilsson MLP algorithm lacks the notion of margin, like the original Perceptron algorithm, but it is surprising how similar "non-margin-aware" algorithms are to "margin-aware" algorithms!

Finally, we give a comparison of these algorithms in Table 8.2 to highlight the importance of the margin in practice. The hyper-parameters were optimized for best generalization performance, using early stopping on the validation set.

Model	Test Err. (%)	
	Forest	Connect-4
Original Perceptron	31.6	25.6
Margin Perceptron	23.8	20.3
Nilsson MLP	10.5	17.2
Simple MLP	**7.3**	**10.1**

▲ **Table 8.2.** *Comparison of "margin aware" algorithms (Margin Perceptron and Simple MLP) and "non margin aware" algorithms (Original Perceptron and Nilsson MLP).*

Contributions

This chapter includes and extends ideas found in the following published paper:

R. Collobert and S. Bengio. Links between perceptrons, MLPs and SVMs. In *International Conference on Machine Learning, ICML*, 2004b.

Here we have included more detail (especially math) and a comparison between algorithms using the margin idea (the Margin Perceptron and the Simple MLP) and algorithms which do not (the Original Perceptron and the Nilsson MLP).

Conclusion

In this chapter, we have drawn new links between three well-known machine learning algorithms, Perceptrons, MLPs, and SVMs. In particular, after pointing out that apart from the training algorithms, Perceptrons are equivalent to linear SVMs. We have shown that this difference is important since it can be computationally expensive to reach the SVM solution by stochastic gradient descent, mainly due to the regularization term. Removing this term, we then showed that the margin can be controlled by other means, namely the learning rate and the mechanism of early stopping. In the case of linearly separable classes, we have shown a relation between the largest existing margin and the margin of solutions found with the Perceptron. Furthermore, we have shown that, under some assumptions, MLPs are in fact SVMs which maximize the margin in the hidden layer space, and hidden units are SVMs in the input space. The theory developed for controlling the margin of Perceptrons can also be applied to MLPs. Finally, we proposed a new algorithm to train MLPs which is both efficient (in time and space) and yields generalization performance at least as good as other models.

Many research directions could be explored in the future. For example, we stated that it can be intractable to reach the SVM solution with stochastic gradient descent. However, it is possible that we do not need to wait for the convergence of the SVM algorithm to its unique solution, and early stopping should work for SVMs as well (even if it will be quite computationally expensive in practice, because of the large number of support vectors). The impact of the radius of the training data which appears in the margin bounds could be studied as well. Finally, note that the only thing forcing the hidden units of a Simple MLP to be different is the (random) initialization. Thus, initialization of MLPs is probably crucial, and should be revised with knowledge of the theory we developed in this chapter.

9 Conclusion

Throughout this thesis, we have stressed the importance of improving existing machine learning techniques based on theoretical considerations, while keeping in mind an important practical aspect: the training time of the algorithm. Our main interests were three popular and powerful algorithms, Multi Layer Perceptrons (MLPs), Mixture of Experts (MoEs) and Support Vector Machines (SVMs).

SVMs appeared to suffer from a *resource consuming* training method. However, let us point out that the state-of-the-art method presented can be viewed as an improvement of stochastic gradient descent under constraints, and that SVMs are a kind of linear model in an arbitrary feature space. Why, in these conditions, does the training of such a model take more time than the training of MLPs with stochastic gradient descent? The practice of reaching the unique solution of SVMs at all costs should be considered cautiously: a "unique solution" does not mean necessarily a "better solution" (in terms of generalization performance).

The "divide and conquer" method we proposed based on a type of MoEs is an important finding. We showed that it is possible to *iteratively* determine a partition of the training set such that even if each expert is trained on a subset of the training set, the resulting mixture has good generalization performance. For this algorithm, our main concern is the number of hyper-parameters. Finding ways to overcome this drawback is an important research direction. In particular, since MLPs can be viewed as a MoEs, it would be interesting to investigate a method to intelligently distribute examples among hidden units, as for the experts in our mixture.

Overall, models based on a *stochastic gradient descent* training algorithm seem to be the most suited for large scale problems. However, we showed that small changes in the structure of a model can strongly affect the speed of the training. In particular, we pointed out that the training cost for a block-diagonal Hessian leads to the independent training of sub-vectors of the parameters of the model, which greatly simplifies the training process. Note the interesting parallel to the divide and conquer technique we proposed, which also forces the experts of our mixture model to train independently, by assigning them a subset of the training set. More research in this direction could be carried out in order to simplify the training of more complex models, MLPs with more than one hidden layer in particular.

We also highlighted the importance of the margin idea introduced by SVMs. We showed that the margin idea was underlying in a certain class of Perceptrons and MLPs, which gives them some advantage over algorithms not "margin-aware" in generalization performance. SVMs do not have a monopoly on the margin and, moreover, they lack the trainable feature space of MLPs. This impacts classification performance in the large scale experiments we proposed. It could be interesting to extend the margin idea to MLPs for classification tasks with more than two classes.

Finally, let us say that we are far from having found the "ultimate" model. Maybe it does not exist. Maybe using prior knowledge depending on the nature of each task is the way to go. Anyway, machine learning is an endless source of problems...

A Non Linear Programming

Here we give the main theorems of non linear programming *with inequality constraints* used in this thesis. This appendix summarizes theorems found in (Ciarlet, 1990), and previously proposed in (Fletcher, 1987) in a less fine way. Non linear programming with inequality constraints attempts to find $\boldsymbol{u}$ such that

$$\begin{cases} \boldsymbol{u} \in U = \{\, \boldsymbol{v} \in \mathbb{R}^n \,:\, \varphi_i(\boldsymbol{v}) \leq 0,\, 1 \leq i \leq m \,\} \\ \qquad\qquad J(\boldsymbol{u}) = \inf\limits_{\boldsymbol{v} \in U} J(\boldsymbol{v})\,. \end{cases} \tag{A.1}$$

Non linear programming with inequality constraints theory generalizes the well-known Lagrangian techniques found for minimizing problems with equality constraints. Thus, we give the definition of a Lagrangian in Definition A.1.

Definition A.1 *The* Lagrangian *associated to the minimization problem (A.1) is the function L defined by*

$$L : \begin{cases} V \times \mathbb{R}^m_+ & \longrightarrow & \mathbb{R} \\ (\boldsymbol{v},\, \boldsymbol{\mu}) & \longmapsto & J(\boldsymbol{v}) + \sum\limits_{i=1}^{m} \mu_i\, \varphi_i(\boldsymbol{v})\,. \end{cases} \tag{A.2}$$

KKT Conditions, General Case

Theorem A.1, below, gives the *necessary* conditions (named *Karush-Kuhn and Tucker* (KKT) *conditions*) for a minimum in non linear programming.

REF P. G. Ciarlet. *Introduction à l'analyse numérique matricielle et à l'optimisation.* Masson, 1990.

REF R. Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, 1987.

Theorem A.1 *Consider a set of functions* $\varphi_i : \Omega \subset V \to \mathbb{R}, 1 \le i \le m$, *defined on a open set* Ω *of a Hilbert space* V. *Consider also*

$$U = \{\, \boldsymbol{v} \in \Omega \,:\, \varphi_i(\boldsymbol{v}) \le 0,\, 1 \le i \le m \,\}\,,$$

a subset of Ω. *Given* $\boldsymbol{u} \in U$, *define*

$$I(\boldsymbol{u}) = \{\, 1 \le i \le m,\, \varphi_i(\boldsymbol{u}) = 0 \,\}\,.$$

It is assumed that functions φ_i, $i \in I(u)$ *are differentiable in* $\boldsymbol{u}$, *and functions* φ_i, $i \notin I(\boldsymbol{u})$ *are continuous in* $\boldsymbol{u}$. *Finally, we consider a function* $J : \Omega \to \mathbb{R}$ *differentiable in* $\boldsymbol{u}$.

If J *has a minimum in* $\boldsymbol{u}$ *with respect to the set* U, *then there exist* generalized Lagrangian multipliers λ_i, $1 \le i \le m$ *such that the following* Karush-Kuhn and Tucker *(KKT)* conditions *hold*

$$\begin{cases} J'(\boldsymbol{u}) + \displaystyle\sum_{i=1}^{m} \lambda_i\, \varphi_i'(\boldsymbol{u}) = 0 \\ \lambda_i \ge 0,\, 1 \le i \le m,\, \displaystyle\sum_{i=1}^{m} \lambda_i\, \varphi_i(\boldsymbol{u}) = 0\,. \end{cases} \tag{A.3}$$

Moreover, if the derivatives φ_i, $i \in I(\boldsymbol{u})$ *are linearly independent, then the Lagrangian multipliers are unique.*

KKT Conditions, Convex Case

If J and all constraints φ_i are supposed to be *convex*, then KKT conditions become *sufficient* for the existence of a minimum of problem (A.1), as stated in the following theorem.

Theorem A.2 *Consider* $J : \Omega \subset V \to \mathbb{R}$, *a function defined on a convex open set* Ω *of a Hilbert space* V. *Consider*

$$U = \{\boldsymbol{v} \in \Omega \,:\, \varphi_i(\boldsymbol{v}) \le 0,\, 1 \le i \le m\}$$

a subset of Ω. *Suppose the constraints* $\varphi_i : \Omega \subset V \to \mathbb{R}, 1 \le i \le m$ *to be convex. Let us also consider* $\boldsymbol{u} \in U$ *such that* φ_i *and* J *are differentiable in* $\boldsymbol{u}$ *(for* $1 \le i \le m$*).*

If J *admits a local minimum in* $\boldsymbol{u}$ *with respect to the set* U, *then there exits* λ_i *(for* $1 \le i \le m$*) such that KKT conditions (A.3) hold .*

Conversely, if $J : U \to \mathbb{R}$ *is convex and if there exists* λ_i, $1 \le i \le m$ *such that KKT conditions hold, then* J *admits a minimum in* $\boldsymbol{u}$ *with respect to* U.

Saddle Point

It is possible to relate the existence of a minimum of problem (A.1) with the existence of a *saddle point* (see Definition A.2) of the Lagrangian (A.2), as stated in Theorem A.3.

Definition A.2 *Consider two sets V and M and a function*

$$L : V \times M \to \mathbb{R}\,.$$

A point $(\boldsymbol{u}, \boldsymbol{\lambda})$ is said to be a saddle point *of function L if $\boldsymbol{u}$ is a minimum for $\boldsymbol{v} \in V \mapsto L(\boldsymbol{v}, \boldsymbol{\lambda})$ and $\boldsymbol{\lambda}$ is a maximum for $\boldsymbol{\mu} \in M \mapsto L(\boldsymbol{v}, \boldsymbol{\mu})$. In other words,*

$$\sup_{\boldsymbol{\mu} \in M} L(\boldsymbol{u}, \boldsymbol{\mu}) = L(\boldsymbol{u}, \boldsymbol{\lambda}) = \inf_{\boldsymbol{v} \in V} L(\boldsymbol{v}, \boldsymbol{\lambda})$$

must hold.

Theorem A.3 *If $(\boldsymbol{u}, \boldsymbol{\lambda}) \in V \times \mathbb{R}_+^m$ is a saddle point of the Lagrangian (A.2), then $\boldsymbol{u}$ belongs to U and is a solution of problem (A.1).*

Conversely, suppose that J and all constraints φ_i are convex and differentiable in $\boldsymbol{u} \in U$ $(1 \leq i \leq m)$. Then, if $\boldsymbol{u}$ is a solution of problem (A.1), there exists at least one $\boldsymbol{\lambda} \in \mathbb{R}_+^m$ such that $(\boldsymbol{u}, \boldsymbol{\lambda}) \in V \times \mathbb{R}_+^m$ is a saddle point of the Lagrangian (A.2).

In the convex case, the problem of finding a saddle point of the Lagrangian as stated in Theorem A.3 is often referred to being the *dual* problem of the *primal* problem (A.1). By extension, Lagrange multipliers introduced in the dual problem are sometimes called *dual variables*. In this thesis we use the expression *dual space* to refer to the space of the Lagrangian multipliers.

Bibliography

M. A. Aiserman, E. M. Braverman, and L. I. Rozonoer. Potential functions technique and extrapolation in learning system theory. In *Proceedings of I.F.A.C.*, 1966.

S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 14(2):251–276, 1998.

P. Auer, H. Burgsteiner, and W. Maass. Reducing communication for distributed learning in neural networks. In J. R. Dorronsoro, editor, *ICANN'2002*, volume 2415 of *Lecture Notes in Computer Science*, pages 123–128. Springer, 2002.

F. R. Bach and M. I. Jordan. Kernel independent component analysis. *Journal of Machine Learning Research*, 3:1–48, 2002.

R. Battiti. First and second-order methods for learning: Between steepest descent and Newton's method. *Neural Computation*, 4(2):141–166, 1992.

C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, 1992. ACM Press.

L. Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, Nimes, France, 1991a. EC2.

L. Bottou. *Une Approche Théorique de l'Apprentissage Connexioniste; Applications à la reconnaissance de la Parole.* PhD thesis, Université de Paris Sud, Orsay, 1991b.

L. Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA, 1984.

C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining Knowledge Discovery*, 2(2):121–167, 1998.

C. Burges and D. Crisp. Uniqueness of the SVM solution. In S. A. Solla, T. K. Leen, and K. R. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12, pages 223–229. MIT Press, 2001.

F. P. Cantelli. Sulla determinazione empirica della leggi di probilita. *Giornale dell'Instituto Italiano degli Attuari*, 4, 1933.

A. Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. In *Compte Rendu Hebdomadaire des Séances de l'Académie des Sciences*, volume 25, pages 536–538, Paris, France, 1847.

P. G. Ciarlet. *Introduction à l'analyse numérique matricielle et à l'optimisation.* Masson, 1990.

R. Collobert and S. Bengio. SVMTorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1:143–160, 2001.

R. Collobert and S. Bengio. A gentle hessian for efficient gradient descent. In *IEEE International Conference on Acoustic, Speech, and Signal Processing, ICASSP*, 2004a.

R. Collobert and S. Bengio. Links between perceptrons, MLPs and SVMs. In *International Conference on Machine Learning, ICML*, 2004b.

R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of SVMs for very large scale problems. *Neural Computation*, 14(5):1105–1114, 2002a.

R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of SVMs for very large scale problems. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems, NIPS 14*, pages 633–640. MIT Press, 2002b.

R. Collobert, S. Bengio, and J. Mariéthoz. Torch: a modular machine learning software library. Technical Report IDIAP-RR 02-46, IDIAP, 2002c.

R. Collobert, Y. Bengio, and S. Bengio. Scaling large learning problems with hard parallel mixtures. *International Journal on Pattern Recognition and Artificial Intelligence (IJPRAI)*, 17(3):349–365, 2003.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3): 273–297, 1995.

T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, 14:326–334, 1965.

T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley and Sons, 1991.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

A. P. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of Royal Statistical Society*, 39:185–197, 1977.

T. G. Dietterich. Approximate statistical test for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1924, 1998.

R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley & Sons, New York, 1973.

Y. M. Ermoliev and V. I. Norkin. Stochastic generalized gradient method with application to insurance risk management. Technical Report IR-97-021, International Institute for Applied Systems Analysis, 1997.

P. A. Estévez, Hélène Paugam-Moisy, Didier Puzenat, and Manuel Ugarte. A scalable parallel algorithm for training a hierarchical mixture of neural experts. *Parallel Computing*, 28:861–891, 2002.

T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13(1):1–50, 2000.

G. W. Flake and S. Lawrence. Efficient SVM regression training with SMO. *Machine Learning*, 46(1–3):271–290, 2002.

R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, 1987.

Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156, 1996.

Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 28(2):337–374, 2000.

S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

F. Girosi, M. Jones, and T. Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7(2):219–269, 1995.

T. Graepel, R. Herbrich, and R. Williamson. From margin to sparsity. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13, pages 210–216. MIT Press, 2001.

J. Hadamard. *Le problème de Cauchy et les équations aux dérivées partielles linéaires hyperboliques*. Hermann, 1932.

M. Haruno, D. M. Wolpert, and M. Kawato. MOSAIC model for sensorimotor learning and control. *Neural Computation*, 13(10):2201–2220, 2001.

W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67, 1970.

J. J. Hopfield. Learning algorithms and probability distributions in feed-forward and feed-back networks. In *Proceedings of the National Academy of Sciences*, volume 84, pages 8429–8433, 1987.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixture of local experts. *Neural Computation*, 3(1):79–87, 1991.

T. Joachims. Making large-scale support vector machine learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. The MIT Press, 1999.

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

M. Karpinski and A. Macintyre. Polynomial bounds for vc dimension of sigmoidal neural networks. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 200–208, 1995.

W. Karush. *Minima of functions of several variables with inequalities as side constraints*. Department of Mathematics. University of Chicago, 1939.

S. S. Keerthi and E. G. Gilbert. Convergence of a generalized SMO algorithm for SVM classifier design. *Machine Learning*, 46(1–3):351–360, 2002.

S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. Murthy. Improvements to platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13(3):637–649, 2001.

J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14, pages 785–792. MIT Press, 2002.

H. W. Kuhn and A. W. Tucker. Non-linear programming. In *Proceedings 2^{nd} Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, Berkeley, 1951. University of California Press.

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.

P. Laskov. An improved decomposition algorithm for regression support vector machines. In S. A. Solla, T. K. Leen, and K. R. Müller, editors, *Advances in Neural Information Processing Systems 12*. The MIT Press, 2000.

Y. LeCun. A learning scheme for asymmetric threshold networks. In *Proceedings of Cognitiva 85*, pages 599–604, Paris, France, 1985.

Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In G.B. Orr and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, pages 9–50. Springer, 1998.

Y. LeCun, I. Kanter, and S. Solla. Second order properties of error surfaces: learning time, generalization. In R. P. Lippman, J. M. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 918–924, Denver, CO, 1991. Morgan Kaufmann.

C. J. Lin. On the convergence of the decomposition method for support vector machines. *IEEE Transactions on Neural Networks*, 12(6):1288–1298, 2001.

L. Mason, J. Baxter, P. L. Bartlett, and M. Frean. Functional gradient techniques for combining hypotheses. In Smola, Bartlett, Schölkopf, and Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 61–73. MIT Press, 1999.

J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society*, A 209:415–446, 1909.

S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K. R. Müller. Fisher discriminant analysis with kernels. In Y. H. Hu, J. Larsen, E. Wilson, and S. Douglas, editors, *Neural Networks for Signal Processing IX*, pages 41–48. IEEE, 1999.

N. J. Nilsson. *Learning Machines*. McGraw-Hill, 1965.

A. B. J. Novikoff. On convergence proofs on perceptrons. In Polytechnic Institute of Brooklyn, editor, *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622, 1962.

E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In J. Principe, L. Giles, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII - Proceedings of the 1997 IEEE Workshop*, pages 276–285. IEEE Press, New York, 1997.

J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. The MIT Press, 1999a.

J. C. 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, 1999b.

D. C. Plaut, S. J. Nowlan, and G. E. Hinton. Experiments on learning by back propagation. Technical Report CMU-CS-86-126, Carnegie-Mellon University, Computer Science Department, 1986.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.

A. Rida, A. Labbi, and C. Pellegrini. Local experts combination through density decomposition. In *Proceedings of UAI'99*. Morgan Kaufmann, 1999.

W. C. Ridgway. An adaptive logic system with generalizing properties. Technical Report 1556-1, Standford University, 1962.

H. Robbins and S. Monro. A stochastic approximation method. In *Annals of Mathematical Statistics*, volume 22, pages 400–407, 1951.

F. Rosenblatt. The perceptron: a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, N.Y., 1957.

S. Rosset, J. Zhu, and T. Hastie. Margin maximizing loss functions. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by back-propagating errors. In D.E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, 1986.

B. Schölkopf, A. Smola, and K. R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

N. N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.

J. Sherman and W. J. Morrison. Adjustment of an inverse matrix corresponding to changes in the elements of a given column or given row of the original matrix. *Annals of Mathematics and Statistics*, 20:621, 1949.

S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R. Murthy. Improvements to the SMO algorithm for SVM regression. *IEEE Transaction on Neural Networks*, 11(5):1188–1183, 2000.

S. Siegel. *Nonparametric Methods for the Behavioral Sciences.* McGraw-Hill, 1956.

F. W. Smith. Pattern classifier design by linear programming. *IEEE Transactions on Computers*, C-17(4):367–372, 1968.

A. Smola and B. Schölkopf. A tutorial on support vector regression. Technical Report NeuroCOLT NC-TR-98-030, Royal Holloway College, University of London, 1998.

A. J. Smola, Z. L. Óvári, and R. C. Williamson. Regularization with dot-product kernels. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13, pages 308–314. MIT Press, 2001.

G. W. Snedecor and W. G. Cochran. *Statistical Methods.* Iowa State University Press, eighth edition, 1989.

A. N. Tikhonov and V. Y. Arsenin. *Solution of ill-posed problems.* Winston & Sons, 1977.

V. Tresp. A bayesian committee machine. *Neural Comp.*, 12(11):2719–2741, 2000.

A. M. Turing. Computing machinery and intelligence. *Mind: A Quarterly Review of Psychology and Philosophy*, 59(236):433–461, 1950.

V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, second edition, 1995.

V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24, 1963.

V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

V. N. Vapnik and A. Y. Chervonenkis. *The Theory of Pattern Recognition.* Nauka, 1974.

P. Vincent and Y. Bengio. Kernel matching pursuit. *Machine Learning*, 48(1): 165–187, 2002.

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.

M. A. Woodbury. Inverting modified matrices. Technical Report Technical Report 42, Statistical Research Group, Princeton University, 1950.

Index