Links between Perceptrons, MLPs and SVMs

Ronan Collobert COLLOBER@IDIAP.CH
Samy Bengio BENGIO@IDIAP.CH
IDIAP, Rue du Simplon 4, 1920 Martigny, Switzerland

Abstract

We propose to study links between three important classification algorithms: Perceptrons, Multi-Layer Perceptrons (MLPs) and Support Vector Machines (SVMs). We first study ways to control the capacity of Perceptrons (mainly regularization parameters and early stopping), using the margin idea introduced with SVMs. After showing that under simple conditions a Perceptron is equivalent to an SVM, we show it can be computationally expensive in time to train an SVM (and thus a Perceptron) with stochastic gradient descent, mainly because of the margin maximization term in the cost function. We then show that if we remove this margin maximization term, the learning rate or the use of early stopping can still control the margin. These ideas are extended afterward to the case of MLPs. Moreover, under some assumptions it also appears that MLPs are a kind of mixture of SVMs, maximizing the margin in the hidden layer space. Finally, we present a very simple MLP based on the previous findings, which yields better performances in generalization and speed than the other models.

1. Introduction

Since the Perceptron algorithm proposed by Rosenblatt (1957), several machine learning algorithms were proposed for classification problems. 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 (LeCun, 1987; Rumelhart et al., 1986). The second one, proposed more recently, is the Support Vector Machine algorithm (SVM) (Vapnik, 1995), which supplied the large margin classifier idea, which is known to improve the generalization performance for binary classification tasks. The aim of this paper is to give a better understanding of Perceptrons and MLPs applied to binary classification, using the knowledge of the margin introduced with SVMs. Indeed, one of the key problem in machine learning is the control of the generalization ability of the models, and the margin idea introduced in SVMs appears to be a nice way to control it (Vapnik, 1995).

After the definition of our mathematical framework in the next section, we first point out the equivalence between a Perceptron trained with a particular criterion (for binary classification) and a linear SVM. As in general Perceptrons are trained with stochastic gradient descent (LeCun et al., 1998), and SVMs are trained with the minimization of a quadratic problem under constraints (Vapnik, 1995), the only remaining difference appears to be the training method. Based on this remark, we study afterward traditional ways to control the capacity of a Perceptron when training with stochastic gradient descent. We first focus on the use of a weight decay parameter in section 3, which can lead to a computationally expensive training time. Then, we show in section 4 that if we remove this weight decay term during stochastic gradient descent, we can still control the margin, using for instance the learning rate or the mechanism of early stopping. These ideas are then extended in section 6 to MLPs with one hidden layer, for binary classification. Under some assumptions it also appears that MLPs are a kind of mixture of SVMs, maximizing the margin in the hidden layer space. Based on the previous findings, we finally propose a simplified version of MLPs, easy to implement and fast to train, which gives in general better generalization performance than other models.

2. Framework Definition

We consider a two-class classification problem, given a training set $(x_l, y_l)_{l=1\dots L}$ with $(x_l, y_l) \in \mathbb{R}^n \times \{-1, 1\}$. Here x_l represents the input vector of the l^{th} example, and y_l represents its corresponding class. We focus

Appearing in *Proceedings of the 21st International Conference on Machine Learning*, Banff, Canada, 2004.

on three models to perform this classification: Perceptrons (including their non-linear extension called Φ-machines[1]), Multi Layer Perceptrons (MLPs), and Support Vector Machines (SVMs). The decision function $f_\theta(.)$ of all these models can be written in its general form as

$$f_\theta(x) = w \cdot \Phi(x) + b\,, \tag{1}$$

where w is a real vector of weights and $b \in \mathbb{R}$ is a bias. The generic vector of parameters θ represents all the parameters of the decision function (that is, w, b, and all parameters of Φ, if any). For Φ-machines and SVMs, Φ is an arbitrary chosen function; in the special case of $\Phi(x) = x$, 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 $\Phi(\cdot) = (\Phi_1(\cdot),\, \Phi_2(\cdot),\, \ldots,\, \Phi_N(\cdot))$, where the i-th hidden unit is described with

$$\Phi_i(x) = h(v_i \cdot x + d_i)\,. \tag{2}$$

Here $(v_i, d_i) \in \mathbb{R}^n \times \mathbb{R}$ represents the weights of the i-th hidden unit, and h is a transfer function which is usually a sigmoid or an hyperbolic tangent. Note that the hyperspace defined with $\{\Phi(x),\ x \in R^n\}$ will be called "feature space", or sometimes "hidden layer space" in the special case of MLPs.

Let us now review the SVM algorithm (Vapnik, 1995): it aims at finding an hyper-plane $f_\theta(.)$ in the feature space, which separates the two classes while maximizing the margin (in the feature space) between this hyper-plane and the two classes. This is equivalent to minimizing the cost function

$$\theta \to \frac{\mu}{2}\,\|w\|^2 + \frac{1}{L}\sum_{l=1}^{L} |1 - y_l\, f_\theta(x_l)|_+\,, \tag{3}$$

where $|z|_+ = \max(0, z)$. $\mu \in \mathbb{R}^+$ is a hyper-parameter (also called weight decay parameter) which has to be tuned, and controls the trade-off between the maximization of the margin and the number of margin errors. SVMs are usually trained by minimizing a quadratic problem under constraints (Vapnik, 1995): if we introduce Lagrange multipliers $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_L) \in \mathbb{R}^L$, the minimization of (3) is equivalent to minimizing in the "dual space"

$$\alpha \to -\sum_{l=1}^{L}\alpha_l + \frac{1}{2}\sum_{k=1}^{L}\sum_{l=1}^{L} y_k\, y_l\, \alpha_k\, \alpha_l\, \Phi(x_k)\cdot\Phi(x_l)\,, \tag{4}$$

subject to the constraints

$$\sum_{l=1}^{L} y_l\, \alpha_l = 0 \quad \text{and} \quad 0 \le \alpha_l \le \frac{1}{L}\ \ \forall l\,. \tag{5}$$

The weight w is then given[2] by

$$w = \frac{1}{\mu}\sum_{l=1}^{L} y_l\, \alpha_l\, \Phi(x_l)\,. \tag{6}$$

Note that in general, only few α_l will be non-zero; the corresponding training examples $(x_l,\, y_l)$ are called "support vectors" (Vapnik, 1995) and give a sparse description of the weight w. Efficient optimization of (4) has been proposed by Joachims (1999). Note also that inner products $\Phi(x_k) \cdot \Phi(x_l)$ in the feature space are usually replaced by a kernel function, which allows efficient inner products in high dimensional feature spaces. However, we will not focus on kernels in this paper.

The training of Perceptrons, Φ-machines and MLPs is usually achieved by minimizing a given criterion $Q(f_\theta(.),\, .)$ over the training set

$$\theta \to \frac{1}{L}\sum_{l=1}^{L} Q(f_\theta(x_l),\, y_l)\,, \tag{7}$$

using gradient descent, until reaching a local optimum. We will focus on *stochastic* gradient descent, which gives good results in time and training performance for large data sets (LeCun et al., 1998). Note that the original Perceptron algorithm (Rosenblatt, 1957) is equivalent to the use of the criterion

$$Q(f_\theta(x_l),\, y_l) = |-y_l\, f_\theta(x_l)|_+\,. \tag{8}$$

However, as this criterion has some limitations in practice (see Bishop, 1995), we will not focus on it. Instead, the criterion usually chosen (for all kinds of MLPs) is either the Mean Squared Error or the Cross-Entropy criterion (Bishop, 1995). Minimizing (7) using the Mean Squared Error criterion is equivalent, from a likelihood perspective, to maximizing the likelihood

[1] Φ-machines were introduced very early in the machine learning community (Nilsson, 1965). They are to Perceptrons what non-linear SVMs are to SVMs: after choosing an arbitrary function Φ which maps the input space into another arbitrary space, we apply the Perceptron algorithm to examples $(\Phi(x_l), y_l)$ instead of $(x_l,\, y_l)$, which allows non-linear discrimination over training examples.

[2] Traditional SVM notations use a hyper-parameter C instead of the weight decay parameter μ. Here, we chose a notation coming from the MLP to establish clear links between all the models. The relation is given simply with $C = 1/(L\mu)$. Lagrange multipliers α_l given here also have to be divided by μ to recover traditional SVM notation.

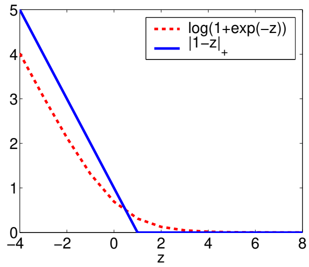


Figure 1. The Cross-Entropy criterion $z \rightarrow \log(1 + \exp(-z))$ versus the margin criterion $z \rightarrow |1-z|_+$.

under the hypothesis that the classes y_l are generated from a smooth function with added Gaussian noise. Since y_l is a binary variable, a Gaussian model is not really appropriate (Bishop, 1995), hence we prefer to focus here on the Cross-Entropy criterion, which maximizes the likelihood while considering y_l as coming from a Bernoulli distribution. The cost function (7) can be rewritten (all details are given in Bishop, 1995) as

$$\theta \rightarrow \frac{1}{L}\sum_{l=1}^{L} \log(1 + \exp(-y_l\, f_\theta(x_l))\,, \tag{9}$$

using the Cross-Entropy criterion. In order to avoid over-fitting, two methods are often used (Plaut et al., 1986) 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

$$\theta \rightarrow \frac{\mu}{2}\,\|w\|^2 + \frac{\nu}{2}\sum_{i=1}^{N}\|v_i\|^2 + \frac{1}{L}\sum_{l=1}^{L}\log(1+\exp(-y_l\, f_\theta(x)))\,. \tag{10}$$

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 point out that the Cross-Entropy criterion $z \rightarrow \log(1 + \exp(-z))$ used for MLPs is a "smooth" version of the "hard" margin criterion $z \rightarrow |1-z|_+$ used for SVMs, as shown in Figure 1. Moreover, it has been shown recently by Rosset et al. (2004) that the use of the Cross-Entropy criterion was related to the maximization of the margin. Thus, the difference between the "smooth" and the "hard" criteria is not crucial to understand the behavior of MLPs. For simplification, we will use in this paper the "hard" version of the criterion. Hence, instead of minimizing (10), we will consider the minimization of

$$\theta \rightarrow \mu\frac{\|w\|^2}{2} + \nu\frac{\sum_i \|v_i\|^2}{2} + \frac{1}{L}\sum_l |1 - y_l\, f_\theta(x)|_+\,. \tag{11}$$

With this criterion, it appears obvious that Φ-machines (respectively Perceptrons) are equivalent to SVMs (resp. linear SVMs). Only the training method remains different: Perceptrons are trained by stochastic gradient descent on (3), whereas SVMs are trained by minimizing the quadratic problem (4) under constraints (see (Joachims, 1999) for details on SVM training). Thus, we will study in the following two sections the training process of Perceptrons (or linear SVMs) with stochastic gradient descent, using what we now know about SVMs. We will consider the two main cases used to control the capacity of a set of Perceptrons; first the use of a regularization term in the cost function (as SVMs), and then the use of early stopping instead.

3. A Note on Stochastic Gradient with Weight Decay

In this section, we consider the cost function (3) in the linear case, which can be rewritten as:

$$(w,\, b) \rightarrow \frac{\mu}{2}\,\|w\|^2 + \frac{1}{L}\sum_{l=1}^{L}|1 - y_l\,(w\cdot x_l + b)|_+\,. \tag{12}$$

For a given training example $(x_l,\, y_l)$, the stochastic update for the weight vector w is:

$$w \leftarrow \begin{cases} (1-\lambda\mu)\, w + \lambda\, y_l\, x_l & \text{if } y_l\,(w\cdot x_l + b) \leq 1 \\ (1-\lambda\mu)\, w & \text{otherwise} \end{cases} \tag{13}$$

given a learning rate λ. From (6), we know that the optimal w is a linear combination of training examples. Therefore, the update (13) is equivalent to the following update written in the dual space:

$$\forall k,\;\; \alpha_k \leftarrow \begin{cases} (1-\lambda\mu)\,\alpha_l + \lambda\mu & \text{if } k = l \\ & \text{and } y_l\,(w\cdot x_l + b) \leq 1 \\ (1-\lambda\mu)\,\alpha_k & \text{otherwise}\,. \end{cases} \tag{14}$$

A study of stochastic gradient in the dual space has already been presented in Kivinen et al. (2002). However, we would like here to use the formulation (14) 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 $(x_l,\, y_l)$ which becomes useless at some point during training. In other words, if we note $\alpha_l^\star$ the value of its corresponding weight α_l at this point, α_l has to be decreased from $\alpha_l^\star$ to zero. Using (14), 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. It will therefore require at least around $\frac{\log(2)}{L\lambda\mu}$ epochs over the whole training set to only divide $\alpha_l^\star$ by two. With a similar analysis, if the l-th example has to be a support vector at bound, it can be shown that it will require around $\frac{\log(L\lambda\mu)}{L\lambda\mu}$ epochs (in the best case) for a weight α_l at zero to reach the bound $\frac{1}{L}$ given in (5). This leads to the following theorem:

Theorem 1 *Training a Perceptron using a weight decay (or a linear SVM) with stochastic gradient descent requires at least* $\frac{1}{L\lambda\mu}$ *epochs over the whole training set.*

Thus, in the case of a small learning rate λ and a small weight decay parameter μ, which is often the case, it becomes very quickly computationally expensive to reach the true SVM solution with stochastic gradient descent. A typical example could be given[3] with $\lambda\mu = 10^{-7}$ and $L = 10^4$, which requires at least 10^3 iterations.

4. A Justification of Early Stopping

We already highlighted that the control of the generalization ability of a Perceptron is usually achieved in two different ways: the first one is the use of a regularization term in the cost function; the second one is the use of early stopping, that is stopping training before reaching a local optimum, usually according to an estimate of generalization error on a "validation" set. Let us now consider Perceptrons training (using stochastic gradient descent) by minimizing the cost function (12) without the regularization term, that is:

$$(w,\,b) \rightarrow \frac{1}{L}\sum_{l=1}^{L} |1 - y_l\,(w\cdot x_l + b)|_+\,. \qquad (15)$$

We will now show in this framework that the margin of a Perceptron can be controlled both by tuning the learning rate used in the gradient descent, and by early stopping. As the margin size controls the capacity and thus the generalization performances (see Vapnik, 1995) it gives a justification for not using a regularization term.

Deriving stochastic gradient equations for cost function (15) leads to the simple Algorithm 1, that we call the "Margin Perceptron" algorithm. Note that this algorithm is far from being new, and is even proposed in Duda and Hart (1973), as a variant of the original Perceptron training algorithm.

[3]This corresponds to the best obtained generalization performances using 10^4 examples, on the two databases presented at the end of the paper.

Algorithm 1 Margin Perceptron Algorithm

```
Initialize w and b to zero
repeat
    for l ∈ {1..L} do
        if y_l (w · x_l + b) ≤ 1 then
            w ← w + λ y_l x_l
            b ← b + λ y_l
        end if
    end for
until termination criterion
```

We first want to consider this algorithm in the case where the classes in the training set are separable. For simplification reasons, we consider in the rest of this section the case where the bias b is fixed to zero (all results can be extended without this constraint, as shown in Collobert & Bengio, 2004b). In that case, we consider the separating hyper-plane $x \rightarrow u\cdot x$ such as $y_l\,(u\cdot x_l) \geq 1\ \forall l$, which has the maximal margin $\rho_{\max} = \frac{2}{\|u\|}$. We then derive equations similar to the well-known Novikoff theorem (1962) for the original Perceptron convergence proof. We note w^t the weight vector of the Perceptron after t updates in Algorithm 1, and l_t the index of the example updated at t. First, we have

$$\begin{array}{rcl} u\cdot w^t &=& u\cdot w^{t-1} + \lambda\, y_{l_t}\, u\cdot x_{l_t} \\ &\geq& u\cdot w^{t-1} + \lambda \\ &\geq& t\,\lambda\,. \end{array} \qquad (16)$$

Moreover, if we consider R the radius of the data (that is, $\|x_l\| \leq R\ \ \forall l$), we have

$$\begin{array}{rcl} \|w^t\|^2 &=& \|w^{t-1}\|^2 + 2\,\lambda\, y_{l_t}\, w^{t-1}\cdot x_{l_t} + \lambda^2\|x_{l_t}\|^2 \\ &\leq& \|w^{t-1}\|^2 + 2\,\lambda + \lambda^2 R^2 \\ &\leq& t\,\lambda^2\,(\frac{2}{\lambda} + R^2)\,. \end{array} \qquad (17)$$

Using (16) and (17), we obtain with the Cauchy-Schwarz inequality:

$$\begin{array}{rcl} t\,\lambda &\leq& u\cdot w^t \\ &\leq& \|u\|\,\|w^t\| \\ &\leq& \frac{2}{\rho_{\max}}\,\sqrt{t}\,\sqrt{\frac{2}{\lambda} + R^2} \end{array} \qquad (18)$$

which leads to an upper bound on the number of updates:

$$t \leq \frac{4}{\rho_{\max}^2}\left(\frac{2}{\lambda} + R^2\right)\,. \qquad (19)$$

This shows that the Margin Perceptron converges in a finite number of updates in the separable case, which has already been demonstrated (Duda & Hart, 1973).

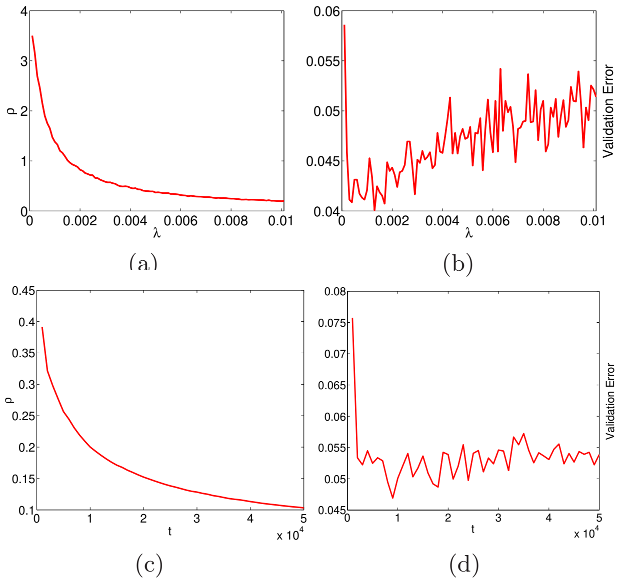


Figure 2. Practical examples of the control of the margin, with the Margin Perceptron algorithm. (a) and (b) represent respectively the margin ρ and the validation error with respect to the learning rate λ, when the number of updates is fixed. (c) and (d) represent respectively the margin ρ and the validation error with respect to the number of updates t, when the learning rate λ is fixed.

However, if we introduce (19) into (17), we are then able to compute a lower bound on the margin $\rho = \frac{2}{\|w^t\|}$ found by the Margin Perceptron, as stated in the following theorem:

Theorem 2 *If the classes are separable, the Margin Perceptron algorithm will converge in a finite number of iterations, and its final margin ρ will satisfy*

$$\rho \geq \rho_{\max} \frac{1}{2 + R^2\,\lambda}\,.$$

Therefore, the smaller is the learning rate, the larger will be the margin of the Perceptron. Note that in Graepel et al. (2001), the authors 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.

In the non-separable case, we would like to be able to control the margin (that is $\rho = \frac{2}{\|w\|}$), while minimizing the number of errors in the margin (15). SVMs control the trade-off between these two quantities with the weight decay parameter μ in the cost function (12).

By construction, the Margin Perceptron minimizes the number of errors in the margin through training iterations. Using (17), the evolution of the margin can be described using the following theorem:

Theorem 3 *After t updates, the margin ρ_t of the Margin Perceptron is bounded in the following way:*

$$\rho_t \geq \frac{2/\lambda}{\sqrt{t(2/\lambda + R^2)}}\,.$$

Thus, the smaller is the number of updates or the learning rate, the larger will be the margin. As the number of updates and the learning rate control the margin, they also control the generalization performance, as highlighted previously. To give an idea of what happens in practice, we give an example of the control of the margin in Figure 2. As early stopping limits the number of updates, a justification of early stopping is given with Theorem 3, which shows that *early stopping controls the size of the margin.* An illustration is given in Figure 3.

5. Extension to Non-Linear Models

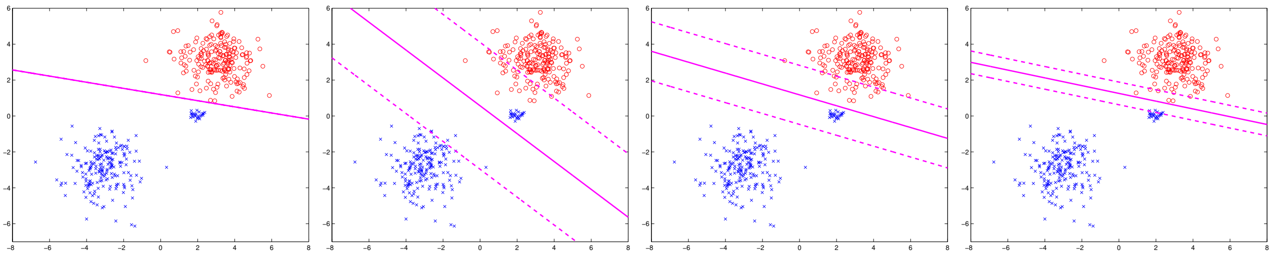
The result in Section 3 is still valid without any restriction if we choose an arbitrary Φ function which maps the input space into an arbitrary space, and if we work with examples $(\Phi(x_l),\, y_l)$ instead of $(x_l,\, y_l)$. Results in Section 4 remain valid as long as there exists a constant $R \in \mathbb{R}$ such as $\|\Phi(x_l)\| \leq R\ \forall l$. Thus, if the constraint is respected, all results apply to Φ-machines and non-linear SVMs.

6. Extension to MLPs

We consider now the minimization of (11), where function f is an MLP with one hidden layer, described by (1) and (2). For an easier understanding of what really happens in MLPs, let us perform yet another simplification: instead of using a hyperbolic tangent for the transfer function h, we consider the following "hard" version of h

$$h(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 \leq x \leq 1 \\ 1 & \text{if } x > 1\,. \end{cases} \tag{20}$$

Note that the use of this transfer function still maintains the universal approximation property of MLPs, as shown in 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 at least similar to a standard MLP, for the same number of hidden units.



(a) (b) (c) (d)

Figure 3. Visual example showing the evolution of the separating hyper-plane and its margin during training with the Margin Perceptron. The aim is to separate two classes: crosses (blue) and circles (red). We have added some noisy crosses near (just below) the circles. (a) shows the hyper-plane found with the original Perceptron algorithm (see (8) in Section 2), after one epoch over the training set; it is not possible to control the margin, and the hyper-plane is not robust to noise. (b), (c) and (d) show the hyper-plane found by the Margin Perceptron algorithm, after 1, 10 and 100 iterations. Dashed lines represents the margin of the hyper-plane. Early stopping thus allow to choose solution (b), which is robust to noise.

As for SVMs, the minimization of (11) can be rewritten as a minimization problem under constraints: (11) is equivalent to minimizing

$$(v_1, \dots, v_N, \alpha) \to -\sum_{l=1}^{L} \alpha_l + \frac{\nu}{2}\sum_{i=1}^{N} \|v_i\|^2 + \frac{1}{2}\sum_{k=1}^{L}\sum_{l=1}^{L} y_k\, y_l\, \alpha_k\, \alpha_l\, \Phi(x_k)\cdot\Phi(x_l) \qquad (21)$$

subject to the constraints (5) and

$$v_i = \frac{w^i}{\nu}\sum_{l=1}^{L} y_l\, \alpha_l\, h'(v_i \cdot x_l + d_i)\, x_l \quad \forall i\,,$$

$$\sum_{l=1}^{L} y_l\, h'(v_i \cdot x_l + d_i)\, \alpha_l = 0\,. \quad \forall i\,,$$

where w^i is the i-th coordinate of w. Once again w is given by (6). Even if we do not know how to solve this minimization problem under constraints, we can state Theorem 4, which is proven by verifying all Karush-Kuhn-Tucker conditions (see for example Fletcher, 1987), which is quite fastidious (details are given in Collobert & Bengio, 2004b).

Theorem 4 *If an MLP is minimum of (11), then it maximizes the margin in the hidden layer space. In other words, (w, b) is solution of the SVM problem (4), for the hidden layer Φ found by the MLP.*

Moreover, the i-th hidden unit is solution of a local SVM on examples x_l which satisfy $|v_i \cdot x_l + d_i| \le 1$. For this SVM, the standard separation constraints $y_l\,(v_i \cdot x_l + d_i) \ge 1$ are replaced with the constraints

$$y_l\,(v_i \cdot x_l + d_i) \ge 1 - y_l\,(b + \sum_{k \ne i} w^k\, h(v_k \cdot x_l + d_k))\,. \qquad (22)$$

Finally, if $(\Phi(x_l),\, y_l)$ is a "global" support vector for w, then $(x_l,\, y_l)$ will be a "local" support vector for the i-th unit if $|v_i \cdot x_l + d_i| \le 1$.

This theorem has several important consequences. First, it shows that an optimal MLP for the cost function (11) is an SVM in the hidden layer space. It is therefore possible to apply the large margin classifier theory (Vapnik, 1995) to MLPs. Moreover, it gives an interesting interpretation of the role of the hidden units: each of them 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 w is a sparse combination of the features $\Phi(x_l)$, whereas the hidden units weights v_i are a sparse combination of the training examples x_l.

Thus, if we now consider training an MLP by minimizing the cost function (11) using stochastic gradient descent, we can derive easily the same lower bound on the number of iterations found in section 3. If we remove regularization terms, as in section 4 for the Margin Perceptron, Theorem 3 is still valid for controlling the "global" margin $\rho = \frac{2}{\|w\|}$ with early stopping or the learning rate. For hidden units, the margin is not straightforward to derive, because constraints (22) are different for each training example; however, using derivations similar to those used for Theorem 3, it is possible to control the norms $\|v_i\|$ as well.

Note that in practice we are left with too many hyper-parameters to select with this kind of MLP: first the weight decay parameters μ and ν, and then the learning rates for the hidden and output layers. In particular, tuning the two learning rates happens to be really tricky. We therefore propose yet another simplification in the next section, already introduced in Collobert and Bengio (2004a) in an optimization framework.

7. A Very Simple MLP

It has been shown in Auer et al. (2002) that any boolean function can be approximated with a Perceptron committee $f_\theta(x) = b + \sum_{i=1}^{N} h(v_i \cdot x + d_i)$. This shows that we do not need to have output weights in MLPs for binary classification. In that case however, if we use the cost function (11), we cannot anymore control the size of the margin in the hidden layer space, as stated in section 6. We thus propose to minimize instead

$$\theta \to \frac{1}{L} \sum_{l} |\beta - y_l\, f_\theta(x)|_+ \,, \tag{23}$$

where β is a hyper-parameter which has to be tuned. As the margin ρ in the hidden layer space is fixed and satisfies

$$\rho = \frac{2\,\beta}{\sqrt{N}} \,,$$

it is very easy to control this margin with β. Moreover, note that what we developed in section 6 remains valid, providing we force $w^i = 1\ \forall i$. In particular, each hidden unit acts as a local SVM, and it is possible to control the norms $\|v_i\|$ (and thus the sparseness of the optimal v_i in the dual space) with the learning rate and early stopping.

Deriving stochastic gradient descent equations for the cost function (23) leads to the very simple Algorithm 2. This algorithm is very fast, because very quickly hidden units will focus on few examples and only a few updates will be performed. Moreover, it is very simple to implement, and as it does not require any internal storage (such as derivatives for example), nor any hyperbolic tangent computation, it is really computationally efficient. Finally, from an optimization viewpoint, it also exploits the fact that the Hessian of (23) is block-diagonal (with respect to pair of hidden units), leading to a mathematically efficient algorithm, as shown in Collobert and Bengio (2004a).

Algorithm 2 Simple MLP Algorithm

```
Initialize w and b to zero
repeat
    for l ∈ {1..L} do
        if y_l (w · x_l + b) ≤ β then
            for i ∈ {1..N} do
                if |v_i · x_l + d_i| ≤ 1 then
                    v_i ← v_i + λ y_l x_l
                    d_i ← d_i + λ y_l
                end if
            end for
            b ← b + λ y_l
        end if
    end for
until termination criterion
```

Model	Test Error (%)	
	Forest	Connect-4
SVM	10.5	11.4
Standard MLP	11.1	11.4
Simple MLP	**8.5**	**10.3**

Model	Time Factor	
	Forest	Connect-4
SVM	134.2	54.7
Standard MLP	3.1	3.0
Simple MLP	1.0	1.0

Table 1. Test errors of SVMs (with a Gaussian kernel), MLPs (trained with the Cross-Entropy criterion), and our proposed Simple MLP on two different classification tasks (Forest and Connect-4). The time factors give the training time relative to the ones of Simple MLPs. This shows that not only the Simple MLP is statistically significantly better than MLPs and SVMs (with 99% confidence), it is also significantly faster (from 3 to 100 times faster!).

To illustrate this, we performed some experiments on the two biggest datasets available on the *UCI* web site, using the Torch C++ library[4] which implements in an efficient way MLPs and SVMs. The SVM code included in the library is one of the fastest available, based on an enhanced version of the SMO algorithm using cached kernel evaluations (as described in Joachims, 1999). We used 300Mo for the kernel evaluation cache on top of the memory needed for the SVM structure, whereas MLPs did not need any additional memory. The first dataset was *UCI Forest*. We modified the 7-class classification problem into a balanced binary classification problem where the goal was to separate class 2 from the others. We used 100,000 examples for training, 10,000 for validation and 50,000 for testing. The second dataset was *UCI Connect-4*. We modified the 3-class classification problem into a balanced binary classification problem where the goal was to separate class "won" against the others. We used 50,000 examples for training, 7,000 for validation and 10,000 for testing. All hyper-parameters of MLPs and SVMs 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 1.

[4]Available at `www.torch.ch`

8. Conclusion

In this paper, we have drawn some new links between three well-known machine learning algorithms, namely Perceptrons, Multi Layer Perceptrons (MLPs), and Support Vector Machines (SVMs). In particular, after pointing out that apart from the training algorithms which are different, 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 have then shown that the margin can still be controlled by other means, namely the learning rate and the mechanism of early stopping. In the case of linear separable classes, we have even 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 are maximizing the margin in the hidden layer space, and where all hidden units are also SVMs in the input space. The results found for Perceptrons regarding methods to control the margin can also be applied to MLPs. Finally, we have proposed a new algorithm to train MLPs which is both very efficient (in time and space) and yields at least as good generalization performance as other models. In the future, we would like to explore the possibility of extending the efficient training algorithm techniques of SVMs (Joachims, 1999) to the case of MLPs.

Acknowledgments

The authors would like to thank the Swiss NSF through the (IM)2 project for financial support.

References

Auer, P., Burgsteiner, H., & Maass, W. (2002). Reducing communication for distributed learning in neural networks. *ICANN'2002* (pp. 123–128). Springer.

Bishop, C. (1995). *Neural networks for pattern recognition.* Oxford University Press.

Collobert, R., & Bengio, S. (2004a). A gentle hessian for efficient gradient descent. *IEEE International Conference on Acoustic, Speech, and Signal Processing, ICASSP.*

Collobert, R., & Bengio, S. (2004b). *Links between Perceptrons, MLPs and SVMs* (Technical Report 04-06). IDIAP.

Duda, R., & Hart, P. (1973). *Pattern classification and scene analysis.* New York: Wiley.

Fletcher, R. (1987). *Practical methods of optimization.* John Wiley & Sons.

Graepel, T., Herbrich, R., & Williamson, R. (2001). From margin to sparsity. *Advances in Neural Information Processing Systems* (pp. 210–216). MIT Press.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*, 359–366.

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

Kivinen, J., Smola, A. J., & Williamson, R. C. (2002). Online learning with kernels. *Advances in Neural Information Processing Systems* (pp. 785–792). MIT Press.

LeCun, Y. (1987). *Modeles connexionnistes de l'apprentissage (connectionist learning models).* Doctoral dissertation, Université P. et M. Curie (Paris 6).

LeCun, Y., Bottou, L., Orr, G., & Müller, K.-R. (1998). Efficient backprop. In G. Orr and K.-R. Müller (Eds.), *Neural networks: Tricks of the trade*, 9–50. Springer.

Nilsson, N. (1965). *Learning machines.* McGraw-Hill.

Novikoff, A. B. J. (1962). On convergence proofs on perceptrons. *Proceedings of the Symposium on the Mathematical Theory of Automata* (pp. 615–622).

Plaut, D., Nowlan, S., & Hinton, G. E. (1986). *Experiments on learning by back-propagation* (Technical Report CMU-CS-86-126). Department of Computer Science, Carnegie-Mellon University.

Rosenblatt, F. (1957). *The perceptron — a perceiving and recognizing automaton* (Technical Report 85-460-1). Cornell Aeronautical Laboratory.

Rosset, S., Zhu, J., & Hastie, T. (2004). Margin maximizing loss functions. In S. Thrun, L. Saul and B. Schölkopf (Eds.), *Advances in neural information processing systems 16.* Cambridge, MA: MIT Press.

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning internal representations by backpropagating errors. In D. Rumelhart and J. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition*, vol. 1, 318–362. MIT Press.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory.* Springer. Second edition.