A Neural Support Vector Network Architecture with Adaptive Kernels

Pascal Vincent & Yoshua Bengio
Département d'informatique et recherche opérationnelle
Université de Montréal
C.P. 6128 Succ. Centre-Ville, Montréal, Québec, Canada, H3C 3J7
{vincentp,bengioy}@iro.umontreal.ca

Submission to IJCNN 2000

Abstract

In the Support Vector Machines (SVM) framework, the positive-definite kernel can be seen as representing a fixed similarity measure between two patterns, and a discriminant function is obtained by taking a linear combination of the kernels computed at training examples called support vectors. Here we investigate learning architectures in which the kernel functions can be replaced by more general similarity measures that can have arbitrary internal parameters. The training criterion used in SVMs is not appropriate for this purpose so we adopt the simple criterion that is generally used when training neural networks for classification tasks. Several experiments are performed which show that such *Neural Support Vector Networks* perform similarly to SVMs while requiring significantly fewer support vectors, even when the similarity measure has no internal parameters.

1 Introduction

Many pattern recognition algorithms are based on the notion of a similarity measure, and generalization is obtained by assigning the same class to similar patterns. In the Support Vector Machines (SVM) framework [3, 11], the positive-definite kernel represents a kind of fixed similarity measure between two patterns, and a **discriminant function** is obtained by taking a linear combination of the kernels computed at training examples called **support vectors**. In this paper we investigate learning architectures in which the kernel functions can be replaced by more general similarity measures with arbitrary internal parameters that may be optimized jointly with the weights assigned to the support-vectors.

Recent work studies adapting a positive-definite kernel based on geometrical considerations after a first SVM optimization run [1]. And [7] investigates ways of using a fixed but not necessarily positive-definite similarity matrix with SVMs. There is also much previous work on learning similarity measures, e.g, [8] adapts the scale of each dimension in a euclidean K-nearest-neighbor classifier, and [4, 2] use a convolutional neural network to learn a similarity measure (respectively for signature verification and thumbprint recognition). In this paper, we consider the minimization of the sum of "margin losses" over the training examples, where *margin* is the signed distance of the discriminant function output to the decision surface (as in AdaBoost [10]). We call this type of architecture "Neural Support Vector Networks" or NSVN. To allow adaptation of parameters inside the kernel, we minimize the squared loss with respect to the hyperbolic tangent of the discriminant function. This criterion, often used to train neural networks for classification, can also be framed as a criterion for maximizing this margin. Our experiments with NSVNs suggest that such architectures can perform similarly to SVMs while requiring significantly fewer support vectors, even when the similarity measure has no internal parameters.

2 Support Vector Machines and Motivations

We consider pattern classification tasks with training data $\{(x_i, y_i)\}$, with $x_i \in \mathcal{R}^n$, and $y_i \in \{-1, +1\}$ is the class associated to input pattern x_i. SVMs [3, 11] use a discriminant function with the following form:

$$f_{\alpha,b}(x) = \sum_{(x_i,y_i)\in sv} \alpha_i y_i K(x, x_i) + b \tag{1}$$

where sv, the set of **support vectors**, is a subset of the training patterns, and $sign(f(x))$ gives the class for any pattern x. Parameters α and b are learned by the SVM algorithm, which also finds the set of support vectors, in

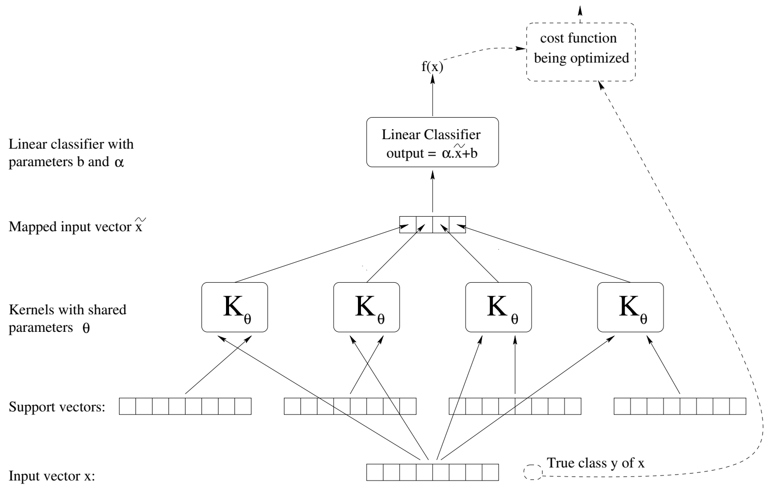

Figure 1: The Neural Support Vector Network Architecture (dotted path is used only during training)

that it brings the α_i of all non-support-vectors down to 0. The kernel K is a function from $\mathcal{R}^n \times \mathcal{R}^n$ to $\mathcal{R}$ that must verify Mercer's conditions, which amounts to being positive definite. For example we have used the Gaussian or RBF kernel $K_\sigma(x_1, x_2) = e^{\frac{-||x_1 - x_2||_2^2}{\sigma^2}}$. Mercer's conditions are necessary and sufficient for the existence of an implicit mapping Φ from the input space to an induced Hilbert space, called the kernel-feature space or Φ-space, such that the kernel actually computes a dot product in this Φ-space: $K(x_1, x_2) =< \Phi(x_1), \Phi(x_2) >$ This "kernel trick" allows the straightforward extension of the dot-product based SVM algorithm, originally designed for finding a margin-maximizing *linear* decision surface (hyperplane) in input space, to finding a margin-maximizing *linear* decision surface in Φ-space, which typically correspond to a *non-linear* decision surface in input space[3]. The *margin* that SVM learning maximizes is defined as the orthogonal Euclidean distance between the separating hyperplane and the nearest of the positive and negative examples, and is motivated by theoretical results that link it to bounds on the generalization error [11]. SVM learning amounts to solving a constrained quadratic programming problem in α, the details of which can be found in [3] and [6] for the "soft-margin" error-tolerationg extension, which adds a complexity control parameter C. Typically, a range of values for C and parameters of the kernel are tried and decided upon according to performance on a validation set. This approach is a serious limiting factor for the research on complex kernels with more than one or two parameters. Yet experiments [5] show that the choice of an appropriate K and parameters can be critical.
Unfortunately, the mathematical formulation of SVMs does not easily allow to incorporate trainable adaptive k-ernels. In particular it is not clear whether the theoretical considerations underlying SVM training still hold for kernels with parameters that are not kept fixed during the optimization. Also the positive definitess constraint limits the choice of possible kernel functions. All these considerations lead us to the design of the architecture described in the following section.

3 Neural Support Vector Networks

Figure 1 shows the general architecture of a Neural Support Vector Network (NSVN). SVMs and NSVNs yield decicion functions f of the same form (see equation 1). Consequently, when using the same fixed K, the function space $\mathcal{F}$ being considered by both algorithms is nearly[1] identical. However in NSVN training, K does not have to be a fixed positive-definite kernel, it can be any scalar function of two input objects, and may have parameters that can be learned. Computing the discriminant function f on a point x can be seen as a two-stage process:

1. map input x into $\tilde{x}$ by computing the values of all support-vector-centered similarity measures for x:
   $\tilde{x} = \Psi(x) = (K_\theta(x, x_1), K_\theta(x, x_2), ..., K_\theta(x, x_m))$ where $\{x_1, x_2, ...x_m\} = sv$ is the set of m support vectors

[1] In NSVN training we don't usually constrain the α_i's to be positive, thus we allow a support-vector-centered kernel to "vote *against* its class".

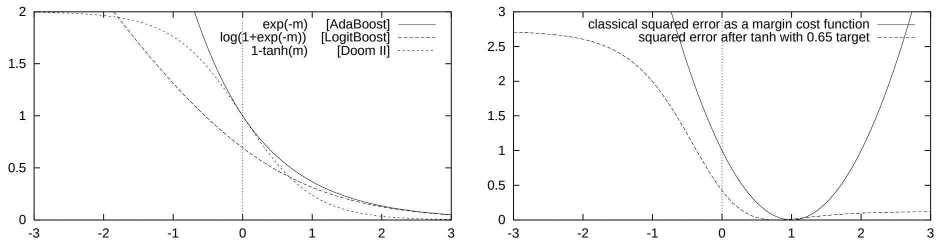

Figure 2: Loss functions of the margin $m = yf(x)$ (horizontal axis). Left: The loss functions used in AdaBoost, LogitBoost and Doom II. Right: The classical squared error criterion (with ± 1 target), and squared error after tanh squashing with $\pm .65$ target, expressed as functions of the margin m.

and θ the parameters of the similarity-measure. Let us call Ψ-space the resulting space.

2. apply a linear classifier on $\tilde{x}$: $f_{\theta,\alpha,b(x)} =< \alpha, \tilde{x} > +b =< \alpha, \Psi(x) > +b$

Suppose first that we are given a fixed set of support vectors sv. Finding a decision function $f \in \mathcal{F}$ of the form of equation 1 amounts to constructing an appropriate linear classifier in Ψ space. Notice the difference with SVM training which finds a margin maximizing linear classifier in Φ-space. While the existence of an implicit Φ-space requires a positive definite kernel, and Φ may be unknown analytically, a Ψ-space can be associated to any K (even non-symmetric) for a given set of support, and is defined, precisely, by its mapping.

There are many possible algorithms for constructing a reasonable linear classifier; for instance a linear SVM could be used (see [7]). In this paper however, we will limit our study to algorithms based on the backpropagation of error gradients. These allow error gradients to be propagated back to parameters θ of a parameterized similarity measure K_θ, and adapt them on the fly, which was one of our primary design goals. Formally, for a training set S (of input/output pairs (x, y), with $x \in \mathcal{X}, y \in \{-1, +1\}$), a gradient based training algorithm will choose the parameters of the decision function $f_{\theta,\alpha,b} \in \mathcal{F}$ that minimize an empirical error defined as the sum of the losses $Q(f_{\theta,\alpha,b}(x), y)$ incurred on each pattern $(x, y) \in S$, with $Q : \mathcal{R} \times \{-1, +1\} \to \mathcal{R}$. Training consists in searching for (θ, α, b) which minimize this average empirical loss:

$$f_{\theta^\star,\alpha^\star,b^\star} = \arg\min_{\theta,\alpha,b} \sum_{(x_i,y_i)\in S} Q(f_{\theta,\alpha,b}(x_i), y_i)$$

3.1 Margin Loss Functions

With NSVNs, we are no longer maximizing the geometrically inspired SVM margin. [10] use another definition of *margin* and relate AdaBoost's goood performance to the maximization of this margin. The support vector form of decision functions can to some extent be framed as a linear combination of weak classifiers à la AdaBoost, each application of the kernel to a support point providing one possible weak classifier. Formally, having a parameterized decision function $f(x)$ whose sign determines the decided class, we define the individual **margin** of a given sample x_i with true class $y_i \in \{-1, +1\}$ as $m_i = y_i f(x_i)$, from which we define a loss function $c(yf(x)) = c(m) = Q(f_{\theta,\alpha,b}(x), y)$. The algorithm searches for the parameters (θ, α, b) that minimize $\sum_{(x_i,y_i)\in S} c(m_i)$ where $c(m)$ is the margin loss function.

[9] compare the performance of several voting methods that were shown to optimize a margin loss function. *AdaBoost* uses an exponential (e^{-m}) margin loss function [10]. *LogitBoost* uses $log_2(1 + e^{-2m})$ and *Doom II* [9] approximates a theoretically motivated margin loss with $1 - \tanh(m)$. As can be seen in Figure 2 (left), all these functions encourage large positive margins, and differ mainly in how they penalize large negative ones. In particular $1 - tanh(x)$ won't penalize outliers to excess, and proved to work better especially in the case of label noise [9].

These margin loss functions have a problem if the parameters allow arbitrary scaling of the discriminant function f, which does not change the decision function, so the parameters could grow indefinitely to maximize margins.

For the 3 previously mentioned voting methods the parameters α_i's are constrained, so that the problem does not appear. Yet, while it makes perfect sense to constrain $\sum_i \alpha_i = 1$ for instance (AdaBoost), how to constrain kernel parameters, or even just b, is much less clear.
Now, our experiments suggest that the well known squared loss functions $(f(x) - y)^2$, and $(\tanh(f(x)) - 0.65y)^2$ often used in neural networks perform rather well, even without constrained parameters. It is interesting to express them as margin loss functions to see why:

Squared loss:	$(f(x) - y)^2 = (1 - m)^2$
Squared loss after $\tanh$:	$(tanh(f(x)) - 0.65y)^2 = (0.65 - tanh(m))^2$

Both are illustrated on figure 2 (right). Notice that the squared loss after $\tanh$ has a shape very similar to the margin loss function used in Doom II, except that it slightly increases for large positive margins, which is why it behaves well with unconstrained parameters.

3.2 Choice of the set of support vectors

So far in our discussion, we assumed that we were given an appropriate set sv of support vectors. We have not yet discussed how to choose such a set. The SVM algorithm considers all training data and automatically chooses a subset as its support vectors by driving the corresponding α_i's down to 0. On the contrary, a simple unconstrained empirical error minimization procedure that would consider all data is not very likely to lead to many zero αs. There are several ways to address this issue:

1. Add a regularization term to the loss function that would push down α_i's and let the algorithm choose its support set by itself, for instance use a penalty term $\lambda \sum_i |\alpha_i|$, where λ allows some control on the number of support vectors.
2. Use a heuristic to choose the support vectors. For instance we could use the support vectors returned by a SVM, or use geometric considerations such as points that are closest to points of the other class.
3. Pick m support points at random, m being seen as a capacity control parameter.

One difference with some classical RBF networks is that the support vectors are training examples (not free parameters). Another one is that our ultimate goal is to learn, simultaneously with the α's, a similarity measure (which may be more sophisticated than a Mahalanobis distance), that is applied to all the support vectors (whereas generally in RBFs, there may be a different variance for each cluster).

4 Experimental Results

In a first series of experiments we compared the performance of NSVN (picking sv at random) and SVM when using the same fixed kernel. The objective of the experiment was to see if the training criterion that we had set up could learn the parameters "correctly" in the sense of giving a decision function that performs comparably to SVMs. The experiments were performed with 5 data sets, four of which were obtained from the UCI Machine Learning database. All involve only 2-way classification. Each of the data set was split in three approximately equal subsets for training (choosing the parameters), validation (controlling capacity, either with the box constraint C for SVMs or with the number of support vectors m for NSVNs), and out-of-sample testing. The breast cancer (200 train + 200 validation + 283 test) and diabetes (200 train + 200 validation + 368 test) data was normalized according to the input values in the training set. The ionosphere data (100 train + 100 validation + 151 test) was used as is. For each of these these we used a Gaussian RBF-Kernel with σ chosen roughly with a few validation-runs on the validation set. The Corel data (296 train + 295 validation + 296 test) consists of 7x7x7 smoothed histogram counts (to the power $\frac{1}{2}$) for the 3 colors of images from the Corel data set[5]. The task was the discrimination of two types of images. For this dataset, we used a kernel of the form $\exp(-||x_1^{0.25} - x_2^{0.25}||_1/\sigma)$, as suggested in [5]. For the Wisconsin Breast Cancer database, we also checked how the algorithms performed when 20% label noise (labels flipped with a probability of 20%) had been added to the training set.
We run SVM optimizations for several values of the complexity-control parameter C. Similarly, we run NSVN optimizations for several values of m, each time trying 10 different random choices of m support vectors and keeping the one that minimized the average empirical loss $(\tanh(f(x)) - 0.65y)^2$. For both algorithms we retained the run with lowest error on the validation set, and the results of the comparative experiments are given in table 1.

Note that we also give the average error on the joined test and validation sets which are a bit less noisy (but slightly biased, although the bias should be similar for both SVM and NSVN).

	SVM				NSVN		
	C	# of S.V.	test error	valid. +test	#m of S.V.	test error	valid. +test
Breast Cancer	15	35	3.5%	3.5%	20	2.8%	3.7%
Noisy Breast Cancer	0.1	149	3.5%	3.7%	10	4.5%	4.1%
Diabetes	0.5	123	23.0%	24.1%	5	24.2%	24.3%
Ionosphere	3	59	6.6%	5.9%	45	7.2%	5.9%
Corel	20	197	12.8%	11.3%	150	12.5%	11.3%

Table 1: Comparison of number of support vectors and error rates obtained with SVM and NSVN. As can be seen, NSVN performs comparably to SVM, often with far fewer support vectors.

Our next experiment aimed at showing that the NSVN architecture was, indeed, able to learn a useful similarity-measure K that is not necessarily positive-definite, together with the weights of the support-vectors. For this, we used a traditional multilayer perceptron (MLP) with one hidden layer, sigmoid activation, and a single output unit, as our similarity measure $K(x_1, x_2)$. The input layer receives the concatenation of the two inputs x_1 and x_2, and the MLP performs the similarity computation, which can be stated as $K_{w0,b0,w1,b1}(x_1, x_2) = tanh(b_1 + w_1.sigmoid(b_0 + w_0.(x_1, x_2)))$

This was tried on the Breast-Cancer data, using 3 hidden units and the same training procedure as previously described (different values of m, 10 random trials each). It achieved 3.0% error on the test-set and 3.1% error error on the combined validation and test set, with 30 support vectors. This shows that, although the parameters of the similarity measure were initialized at random, NSVN training was able to find values appropriate for the requested classification task.

5 Conclusion and Future Work

We have proposed a new architecture that is inspired from SVMs but is driven by the objective of being able to *learn* a similarity function that is not necessarily a positive definite kernel.

In the process, we have uncovered a link between the loss functions typically used in neural network training and the kind of *margin* cost functions optimized by AdaBoost and similar algorithms, and outlined the differences with the geometrically-inspired *margin* maximized by SVM learning. Moreover, we have shown experimentally that both approaches perform comparably, in terms of expected errors, which may suggest that the "support-vector kind of architecture" (which determines the form of discriminant functions that are considered) may be responsible for their good performance, and not only the particular kind of margin-maximization that is used.

Several experiments on classification data sets showed that the proposed algorithm, when used with the same fixed Kernel, performs comparably to SVM, often with substantially fewer support-vectors (chosen at random!), which is in itself interesting, as it allows an equally substantial improvement in terms of speed.

But more important, we have defined a framework that opens the way to the exploration of more interesting adaptive similarity measure than the fixed positive-definitie kernels typically used with SVM. Trainable parametric similarity measures can now be used, that were designed to incorporate prior knowledge specific to the task at hand (such as those proposed in [4, 2]).

A large number of open questions remain though, in particular regarding the merits of various margin cost function, or the way to choose the set of support vectors...

Acknowledgements

The authors would like to thank Olivier Chapelle, Patrick Haffner and Léon Bottou for helpful discussions, as well as the NSERC Canadian funding agency, and the IRIS network for support.

References

[1] S. Amari and S. Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 1999. to appear.

[2] P. Baldi and Y. Chauvin. Neural networks for fingerprint recognition. *Neural Computation*, 5(3):402–418, 1993.

[3] B. Boser, I. Guyon, and V. Vapnik. An algorithm for optimal margin classifiers. In *Fifth Annual Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, 1992.

[4] J. Bromley, J. Benz, L. Bottou, I. Guyon, L. Jackel, Y. LeCun, C. Moore, E. Sackinger, and R. Shah. Signature verification using a siamese time delay neural network. In *Advances in Pattern Recognition Systems using Neural Network Technologies*, pages 669–687. World Scientific, Singapore, 1993.

[5] O. Chapelle, P. Haffner, and V. Vapnik. Svms for histogram-based image classification. *IEEE Transactions on Neural Networks*, 1999. accepted, special issue on Support Vectors.

[6] C. Cortes and V. Vapnik. Soft margin classifiers. *Machine Learning*, 20:273–297, 1995.

[7] T. Graepel, R. Herbrich, P. Bollmann-Sdorra, and K. Obermayer. Classification on pairwise proximity data. In *12th Annual Conference on Neural Information Processing Systems (NIPS'98)*, 1999.

[8] D.G. Lowe. Similarity metric learning for a variable-kernel classifier. *Neural Computation*, 7(1):72–85, 1995.

[9] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent. In S. A. Solla, T. K. Leen, and K-R. Mller, editors, *Advances in Neural Information Processing Systems 12*. The MIT Press, 2000. Accepted for Publication.

[10] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, to appear, 1998.

[11] V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New-York, 1995.