Maximizing the Margin with Feed-forward Neural Networks

Enrique Romero and René Alquézar
Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya

Abstract - Feed-forward Neural Networks (FNNs) and Support Vector Machines (SVMs) are two machine learning frameworks developed from very different starting points of view. In this work a new learning model for FNNs is proposed such that, in the linearly separable case, tends to obtain the same solution that SVMs. The key idea of the model is a weighting of the sum-of-squares error function, which is inspired in the AdaBoost algorithm. The model depends on a parameter that controls the hardness of the margin, as in SVMs, so that it can be used for the non-linearly separable case as well. In addition, it allows to deal with multiclass and multilabel problems in a natural way (as FNNs usually do), and it is not restricted to the use of kernel functions. Finally, it is independent of the concrete algorithm used to minimize the error function. Both theoretic and experimental results are shown to confirm these ideas.

I. Introduction

Feed-forward Neural Networks (FNNs) and Support Vector Machines (SVMs) are two different machine learning frameworks for approaching classification and regression problems. We will consider the classification task given by a dataset $X = \{(x_1, y_1), \ldots, (x_L, y_L)\}$, with $x_i \in \mathbb{R}^N$ and $y_i \in \{-1, +1\}^C$, where C is the number of classes. Minimizing the sum-of-squares (or cross-entropy) error function and maximizing the margin are very different points of view with very interesting properties ([1], [2]). Looking at the similarities and differences between FNNs and SVMs, it can be observed that the main difference between the sum-of-squares minimization problem of an FNN and the maximization problem of a (1-Norm Soft Margin) SVM lies on the constraints related to the objective function. Since these constraints are the responsible for the existence of the support vectors, their behaviour will give the key to propose a new learning model for FNNs that, in the linearly separable case, tends to obtain the same solution that SVMs. Trying to obtain support vectors (that is, points with margin 1), a weighting of the sum-of-squares error function is proposed. This weighting is inspired in the AdaBoost algorithm [3], and it consists of modifying the contribution of every point to the total error depending on its margin. The model depends on a parameter that controls the hardness of the margin, as in SVMs, so that it can be used for the non-linearly separable case. In addition, the classical FNN architecture of the new model presents some advantages. First, it allows to deal with multiclass and multilabel problems in a natural way. This is a difficult problem for SVMs, since they are initially designed for binary classification problems. In addition, the final solution is neither restricted to have an architecture with so many hidden units as points (or support vectors) in the dataset nor to use kernel functions necessarily. Both theoretic and experimental results are shown to confirm these ideas.

Some preliminaries about FNNs, SVMs and AdaBoost can be found in Section II. In Section III, some similarities and differences between FNNs and SVMs will be discussed. The learning model and some theoretic results will be presented in Section IV. Finally, the experimental results will be shown in Section V.

This work was supported by Consejo Interministerial de Ciencia y Tecnología (CICYT), under project TAP1999-0747

II. Preliminaries

A. Feed-forward Neural Networks

The well known architecture of an FNN is structured by layers of units, with connections between units from different layers in forward direction [1]. A fully connected FNN with one output unit and one hidden layer of units computes the function:

$$f_{FNN}(x) = \varphi_0 \left(\sum_{i=1}^{Nhid} \lambda_i \, \varphi_i(\omega_i, x, b_i) + b_0 \right) \qquad (1)$$

where $\lambda_i, b_i, b_0 \in \mathbb{R}$, $x, \omega_i \in \mathbb{R}^N$ and $Nhid$ is the number of units in the hidden layer. The most used activation functions $\varphi_i(\omega, x, b)$ in the hidden units are sigmoidal for Multi-layer Perceptrons (MLPs) and radially symmetric for Radial Basis Function Networks (RBFNs), although many other functions may be used ([6], [7]). Output activation functions $\varphi_0(u)$ use to be sigmoidal or linear.

The objective of the training process is to choose adequate parameters to minimize a predetermined cost function. The sum-of-squares error function is the most usual:

$$E(X) = \sum_{i=1}^{L} \frac{1}{2} [f_{FNN}(x_i) - y_i]^2.$$

As it is well known, the sum-of-squares error function $E(X)$ is an aproximation to the squared norm of the error function $f_{FNN}(x) - y$ in the Hilbert space L^2.

The architecture (connections, number of hidden units and activation functions) is usually fixed *a priori*, whereas the weights are learned during the training process. For convenience, we will divide the weights into frequencies $(\omega_i)_{i=1}^{Nhid}$, coefficients $(\lambda_i)_{i=1}^{Nhid}$ and biases $(b_i)_{i=1}^{Nhid}$. Note that the appearance of the output function given by (1) is determined by the architecture of the trained FNN.

B. Support Vector Machines

The idea of SVMs can be stated as follows ([2], [13]): the input vectors are mapped into a (usually high-dimensional) inner product space through some non-linear mapping ϕ, chosen *a priori*. In this space (the feature space), an optimal hyperplane is constructed. Using a kernel function $K(u,v)$ the mapping can be implicit, since the inner product which defines the hyperplane can be evaluated as $\langle \phi(u), \phi(v) \rangle = K(u,v)$ for every two vectors $u, v \in \mathbb{R}^N$. In SVMs, an optimal hyperplane means a hyperplane with maximal normalized margin with respect to the dataset. The (functional) margin of a point (x_i, y_i) with respect to a function f is defined as $mrg(x_i, y_i, f) = y_i f(x_i)$. The margin of a function f with respect to a dataset X is the minimum of the margins of the points in the dataset. If f is a hyperplane, the normalized (or geometric) margin is defined as its margin divided by the norm of the orthogonal vector to the hyperplane. Using Lagrangian and Kuhn-Tucker theory, the maximal margin hyperplane for a binary classification problem turns to be

$$f_{SVM}(x) = \sum_{i=1}^{L} y_i \alpha_i K(x_i, x) + b \qquad (2)$$

where the vector $(\alpha_i)_{i=1}^{L}$ is the (1-Norm Soft Margin) solution of the following constrained optimization problem in the dual space:

Maximize

$W(X) = -\frac{1}{2} \sum_{i,j=1}^{L} y_i \alpha_i y_j \alpha_j K(x_i, x_j) + \sum_{i=1}^{L} \alpha_i$

subject to

$\sum_{i=1}^{L} y_i \alpha_i = 0$

$0 \leqslant \alpha_i \leqslant C \qquad i = 1, \ldots, L.$

The points x_i with $\alpha_i > 0$ (active constraints) are support vectors, while bounded support vectors have $\alpha_i = C$. Non-bounded support vectors have margin 1, while bounded support vectors have margin less than 1. A point is well classified if and only its margin with respect to f_{SVM} is positive. The cost function $-W(X)$ is (plus a constant) the squared norm of the error function $f_{SVM}(x) - y$ in the Reproducing Kernel Hilbert Space associated to $K(u,v)$ ([2], pag. 41). By setting $C = \infty$, one obtains the hard margin hyperplane. The most used kernel functions $K(u,v)$ are polynomial or Gaussian-like. In contrast to FNNs, note that the appearance of the solution is a consequence of the way the problem is solved.

C. AdaBoost

The AdaBoost algorithm is a particular boosting algorithm introduced in [3] and later improved in [11]. AdaBoost calls a given weak learning algorithm in a series of rounds. On each round t it computes a weak hypothesis h_t. One of the main ideas of the algorithm is to maintain a distribution D_t (a set of weights) over the training set. Initially, all weights are set equally, but on each round t, the weights are modified:

$$D_t(x_i, y_i, H_t) = \frac{e^{-mrg(x_i, y_i, H_t)}}{Z_t} \qquad (3)$$

where $H_t = \sum_{j=1}^{t} \alpha_j h_j(x_i)$ is the current hypothesis, and Z_t is a normalization factor so that D_t is a probability distribution. The effect of the distribution D_t is that the weights of incorrectly classified examples are increased so that the weak learner h_{t+1} is forced to focus on the hard (depending on D_t) examples in the training set.

III. FNNs vs SVMs

A. Comparing the output functions

As pointed out elsewhere (see, for example, [14]), the output function f_{SVM} of a SVM (2) can be implemented with a fully connected FNN with one output unit and one hidden layer of units:

1. Number of hidden units: L $(= \|X\|)$
2. Coefficients: $\lambda_i = y_i \alpha_i$
3. Frequencies: $\omega_i = x_i$
4. Biases: b_i vanishes, and $b_0 = b$
5. Activation functions:
   (a) Hidden layer: $\varphi_i(x_i, x, b_i) = K(x_i, x)$
   (b) Output layer: φ_0 linear

As in SVMs, the only parameters to be learned in such an FNN computing (1) would be the coefficients and the biases. So, the main differences between FNNs and SVMs rely both on the cost function to be optimized and the constraints, since specific learning algorithms are a consequence of the optimization problem to be solved.

B. Comparing the cost functions

Our first question has to do with the similarities between the respective cost functions. Defining

$$K_L = (K(x_i, x_j))_{i,j=1}^{N},$$

$y = (y_1, \ldots, y_L)^T$, $y\alpha = (y_1\alpha_1, \ldots, y_L\alpha_L)^T$ and considering the identifications stated in Section III-A we can express the respective cost functions as:

$$E(X) = \frac{1}{2}\, y\alpha^T \cdot K_L \cdot K_L \cdot y\alpha - \; y\alpha^T \cdot K_L \cdot y + \frac{1}{2} L$$

$$W(X) = -\frac{1}{2}\, y\alpha^T \cdot K_L \cdot y\alpha + \; y\alpha^T \cdot y$$

Regardless of their apparent similarity, is there any relationship between the minima of $E(X)$ and the maxima of $W(X)$ (or equivalently, the minima of $-W(X)$)? The next result partially answers this question.

Proposition 1. If K_L is non-singular, then the respective cost functions $E(X)$ and $-W(X)$ attain their unique minimum (without constraints) at the same point.

Proof. As $E(X)$ and $-W(X)$ are convex functions, a necessary and sufficient condition for $y\alpha^*$ to be a global minimum is that their derivative with respect to $y\alpha$ vanishes. Since K_L non-singular, both equations have the same solution. □

If K_L is singular, there will be more than one point where the optimum value is attained, but all of them are equivalent. In addition, K_L has rows which are linearly dependent among them. It indicates that the information provided (via the inner product) by a point in the dataset is redundant, since it is a linear combination of the information provided by other points. Thus, that point could probably be eliminated from the dataset and the final solution would not change.

Indeed, the optima can be very different depending on the absence or presence of the constraints. Therefore, it seems that the main difference lies on the constraints.

IV. An FNN that maximizes the margin

A. How does every point contribute to the cost function?

The existence of linear constraints in the optimization problem to be solved in SVMs has a very important consequence: only some of the α_i will be different from zero. These coefficients are associated with the so called support vectors. Thus, the remaining vectors can be omitted, both to optimize $W(X)$ and to compute the output (2). The problem is that we do not know them *a priori*. In the linearly separable case (hard margin), support vectors have margin 1 (that is, $f_{SVM}(x_i) = y_i$), while the remaining points (that will be referred to as "superclassified" points) have a margin strictly greater than 1.

In contrast, for FNNs minimizing the sum-of-squares error function every point makes its contribution to the total error. The greater is the squared error, the greater will be the contribution, independently of whether the point is well or wrongly classified. With linear output units, there may be points (very) well classified with a (very) big squared error. "Superclassified" points are a clear example of this type. Sigmoidal output units can help to solve this problem, but they can also create new ones (in the linearly separable case, for example, the solution is not bounded). An alternative idea to sigmoidal output units could be to reduce the contribution of "superclassified" points and reinforce those of misclassified points, as explained in the next section.

B. Weighting the contribution

Indeed, we do not know *a priori* which points will be finally "superclassified" or misclassified. But during the FNN learning process it is possible to treat every point in a different way depending on its error (or, equivalently, its margin). In order to simulate the behaviour of a SVM, the learning process could be guided by the following heuristics:

1. Any well classified point contributes less to the error than any misclassified point.
2. Between well classified points, the contribution is larger for smaller errors in absolute value (or equivalently, smaller margins).
3. Between misclassified points, the contribution is larger for larger errors in absolute value (or equivalently, smaller margins).

These guidelines reinforce the contribution of misclassified points and reduces the contribution of well classified ones. As can be seen, this is exactly the same idea than the distribution (3) for AdaBoost. Similarly, the contribution of every point to the error can be modified simply by weighting it individually as a function of the margin with respect to the output function f_{FNN}. In order to allow more flexibility to the model, two parameters $\alpha^+, \alpha^- \geqslant 0$ can be introduced into the weighting function as follows ($mrg = mrg(x_i, y_i, f_{FNN})$):

$$D(x_i, y_i, \alpha^+, \alpha^-) = \begin{cases} e^{-|mrg|^{\alpha^+}} & \text{if } mrg \geqslant 0 \\ e^{+|mrg|^{\alpha^-}} & \text{if } mrg < 0 \text{ and } \alpha^- \neq 0 \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

This weighting can be applied at least at two levels ($E_p = E_p(f_{FNN}(x_i), y_i, \alpha^+, \alpha^-)$):

1. Weighting the sum-of-squares error:

$$E_p = \frac{1}{2}[f_{FNN}(x_i) - y_i]^2 \cdot D(x_i, y_i, \alpha^+, \alpha^-) \tag{5}$$

2. Weighting the sum-of-squares error derivative (only if the derivative is involved in the learning process):

$$\frac{\partial E_p}{\partial f_{FNN}} = (f_{FNN}(x_i) - y_i) \cdot D(x_i, y_i, \alpha^+, \alpha^-) \tag{6}$$

Graphically (see figure 1), the right branch of the squared error parabola is bended to a horizontal asymptote. Weighting the sum-of-squares error derivative also implies a kind of weighting the sum-of-squares error, although in a slightly different way.

The following result justifies that the previously suggested weighting functions are well founded. In addition, it allows to construct new error functions in order to simulate the behaviour of a SVM.

0-7803-7278-6/02/$10.00 ©2002 IEEE

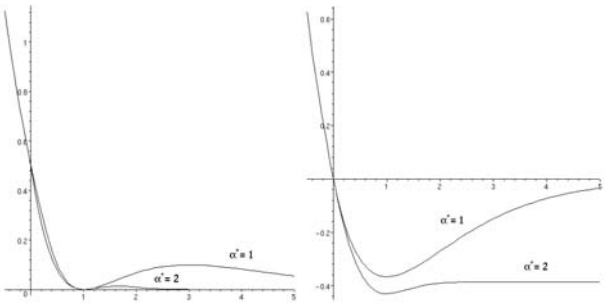

Fig. 1. Individual error for the weighted sum-of-squares error (5) (left) and the weighted sum-of-squares error derivative (6) (right) for several values of α^+ ($\alpha^- = 0$).

Theorem 1. Let $f \in \mathbb{R}$, $y \in \{-1, +1\}$, $\alpha^+, \alpha^- \geqslant 0$ and $E_p(f, y, \alpha^+, \alpha^-)$ an error function satisfying:

1. For every $\alpha^+, \alpha^- \geqslant 0$, E_p attains its (absolute) minimum value when $yf = 1$, and this value A does not depend on α^+.
2. For every $\alpha^- \geqslant 0$ and every y, f satisfying $yf > 1$ we have

$$\lim_{\alpha^+ \to \infty} E_p(f, y, \alpha^+, \alpha^-) = A$$

Then, if $X = \{(x_1, y_1), \ldots, (x_L, y_L)\}$ is a linearly separable dataset, the hyperplane $h(x)$ that maximizes the normalized margin also minimizes asymptotically ($\alpha^+ \to \infty$) the weighted sum-of-squares error function

$$E_P(X) = \sum_{i=1}^{L} E_p(h(x_i), y, \alpha^+, \alpha^-). \quad (7)$$

Remarks.

- The theorem holds true independently of whether the dataset X is linearly separable either in the input space or in the feature space.
- The previously suggested weighting functions (5) and (6) satisfy the hypothesis of the theorem.

Proof. Since X is linearly separable, $h(x)$ satisfies that the margin $y_i h(x_i) = 1$ for the support vectors, whereas $y_i h(x_i) > 1$ for the non-support vectors. Since A is the absolute minimum of E_p, regardless of α^+, the minimum value that E_P could attain is $L \cdot A$. The first hypothesis implies that for support vectors, $E_p(h(x_i), y, \alpha^+, \alpha^-) = A$. For non-support vectors, this value is asymptotically attained when $\alpha^+ \to \infty$, because of the second hypothesis. □

The reciprocal may not be necessarily true, since there can be many different hyperplanes which asymptotically minimize $E_P(X)$. However, the solution obtained by minimizing $E_P(X)$ is expected to have a similar behaviour that a SVM. In particular:

1. It is expected that a larger α^+ will be related to a harder margin.
2. Points with margin less or equal than 1 are expected to be support vectors. For the linearly separable case, the margin expected for every support is 1.
3. Theoretical results for SVMs (for example, generalization bounds) are expected to be easily applicable or adapted to the obtained solutions.

C. Practical considerations

Some benefits can be obtained by minimizing an error function (7) as the previously defined.

First, the minimization of (7) does not assume the existence of any predetermined architecture in the FNN:

1. There is no need to have, in the final solution, as many hidden units as points (or support vectors) in the dataset, nor the frequencies must be the points in the dataset.
2. There is no need to use kernel functions, since there is no inner product to compute in the feature space.

In addition, it presents a number of advantages over the classical SVM model:

1. There is no limit on the number of classes to deal with. For C-class problems, it is enough to construct an architecture with C output units. The individual squared error of every point is defined as usual [1]:

$$E_P(X) = \sum_{i=1}^{L} \sum_{c=1}^{C} E_p(f^c_{FNN}(x_i), y^c_i, \alpha^+, \alpha^-). \quad (8)$$

2. The same error function (8) allows to deal with multilabel problems without restrictions.

Finally, it is independent of the concrete algorithm used to minimize the error function.

D. Related work

In [4] an equivalence between Sparse Approximation and SVMs is shown, with some relationships to RBFNs. A single-layer perceptron learning algorithm that asymptotically obtains the maximum margin classifier is presented in [8]. In order to work, the dataset must be necessarily linearly separable, and the learning rate should be increased exponentially, leading to weights arbitrarily large. In [12], a modified SVM approach for training an MLP with a fixed number of hidden units is described. An estimation of an upper bound of the Vapnik-Chervonenkis dimension is iteratively minimized over the frequencies and the biases. The SVM method inspires the calculation of the coefficients, but it is not the same one. The work in [15] investigates learning architectures in which the kernel function can be replaced by more general similarity measures that can have internal parameters. Although the frequencies are forced to be the points in the dataset, the cost function $E(X) = \sum_{i=1}^{L} [0.65 - \tanh(mrg(x_i, y_i, f))]^2$ is used.

0-7803-7278-6/02/$10.00 ©2002 IEEE

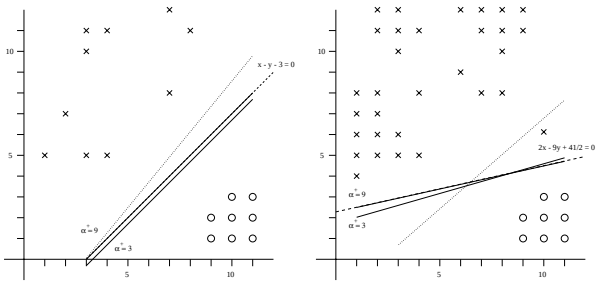

Fig. 2. Separating hyperplanes (solid lines) after minimizing the weighted sum-of-squares (7) for different values of α^+ in the two-class linearly separable problems $L2$ (left) and $R2$ (right).

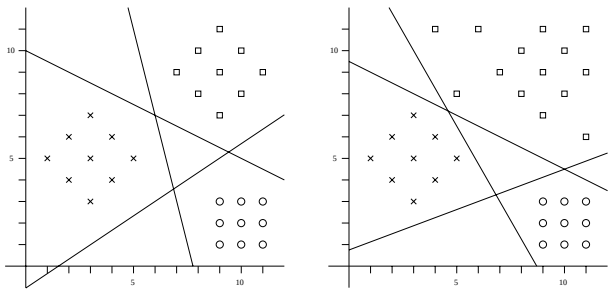

Fig. 3. Separating hyperplanes when minimizing the weighted sum-of-squares (8) for $\alpha^+ = 9$ in the three-class linearly separable problems $L3$ (left) and $R3$ (right).

V. Experiments

We performed some experiments on artificial data in order to test both the validity of the new model and the predictions made in Section IV. In particular, we were interested in testing:

1. Whether learning is possible or not with a standard FNN architecture when a weighted sum-of-squares error $E_P(X)$ is minimized with standard methods.
2. The effect of α^+ on the hardness of the margin.
3. The identification of the support vectors, simply by comparing their margin value with 1.
4. The behaviour of the model in multiclass problems minimizing the error function (8).
5. The behaviour of the model in non-linearly separable cases, when a non-linear activation function in the hidden layer is needed.
6. Whether the use of non-kernel functions can lead or not to a behaviour similar to that of kernel functions.

We set $\alpha^- = 0$, so that misclassified points had all of them a weight of 1. All experiments were performed with FNNs trained with standard Back-propagation [9] weighting the sum-of-squares error derivative (6). We will call this method BPW. Every architecture had linear output units, and was trained in batch mode.

A. Two linearly separable classes

Our first experiment consisted of learning the maximal margin hyperplane of two linearly separable classes. We constructed two different linearly separable datasets ($L2$ and $R2$), shown in figure 2. Despite of their apparent simplicity, there is a big difference between the maximal margin hyperplane (dashed line) and the minimum sum-of-squares hyperplane (dotted line), used as the initial weights for BPW in an MLP without hidden layers. Solid lines in figure 2 show the resulting hyperplanes after the training for different values of α^+. As can be observed, the maximal margin hyperplane was obtained for $\alpha^+ = 9$, so that the effect of α^+ on the hardness of the solution margin was confirmed. When looking at the output calculated by the network, we could see that every point had functional margin strictly greater than 1 except:

- For $L2$, points $\{(9,2),(10,3),(4,5),(7,8)\}$.
- for $R2$, points $\{(10,3),(1,4),(10,6)\}$.

which had margin ≈ 1. These points are, respectively, the support vectors of the maximal margin hyperplane of the datasets, confirming the prediction about the support vectors just by looking at their margin value.

B. Three linearly separable classes

Our second experiment consisted of trying to learn three linearly separable classes. As previously, we constructed two different linearly separable datasets ($L3$ and $R3$), shown in figure 3. In this case, the constructed MLPs had three output units, and BPW minimized (8). In the same conditions that in the previous section, solid lines in figure 3 show the resulting hyperplanes (the output function for every output unit) after the minimization with BPW for $\alpha^+ = 9$. We looked at the output calculated by the network for every point in the dataset, in order to identify the support vectors. Splitting the resulting network into one network for every class, we observed that every output unit of every network, as in the two linearly separable case, had functional margin strictly greater than 1 for every point in the dataset except

- For $L3$:
  - $\{(9,3),(3,3),(9,7)\}$ for the circled points class.
  - $\{(9,1),(5,5),(7,9)\}$ for the crossed points class.
  - $\{(11,3),(3,7),(9,7)\}$ for the squared points class.
- For $R3$:
  - $\{(9,3),(3,3),(11,6)\}$ for the circled points class.
  - $\{(9,1),(5,5),(5,8)\}$ for the crossed points class.
  - $\{(11,3),(3,7),(5,8)\}$ for the squared points class.

which had margin ≈ 1. These vectors are those which would have been obtained as support vectors if we had binarized the problem, solving the three respective SVM optimization problems. It confirms our hypothesis about the goodness of the model for multiclass problems.

0-7803-7278-6/02/$10.00 ©2002 IEEE

Fig. 4. Generalization obtained by SVM-Light (left), BPW with gaussian functions (center) and BPW with sine functions (right) for the *Two Spirals* problem. The results for BPW are the mean over 10 runs.

C. The *Two Spirals* Problem

The well known *Two spirals* problem consists of identifying the points of two interlocking spirals, with a training set of 194 points. A SVM with gaussian kernels and standard deviation 1 was constructed using the SVM-Light software [5] (for polynomial kernels we did not obtain satisfactory results). The hard margin solution contained 176 support vectors (0 bounded). In order to make a comparison with an FNN with the same activation functions and the same frequencies, we constructed an RBFN with 194 hidden gaussian units (also with standard deviation 1). The frequencies were fixed to be the points in the dataset, and the initial range for the coefficients was 0.001. As it was a separable problem with gaussian kernels, we set $\alpha^+ = 9$. After 10 runs of a training with BPW, the mean of the number of points with functional margin less than 1.05 (support vectors in our model) was 168. These points were always a subset of the support vectors obtained with the SVM-Light software. None of them had functional margin less than 0.95.

We also constructed an MLP with a hidden layer of 24 sinusoidal units, as in [10]. Initial frequencies for BPW were assigned randomly to an interval $[-3.5, 3.5]$, and the initial range for the coefficients was 0.0001. We set again $\alpha^+ = 9$. After 10 runs of a training with BPW, the mean of the number of points with functional margin less than 1.05 was 101.6, and none of them had functional margin less than 0.95. These experiments confirm that there is no need to use either a "SVM architecture" or kernel functions (the sine is not a kernel function).

The generalization obtained by these models can be seen in figure 4, where the corners are $(-6.5, -6.5)$ and $(+6.5, +6.5)$. It is worth knowing that all the points in the training set are radially equidistant inside a disk of radius 6.5. Therefore, while gaussian functions are expected to have a good behaviour for this problem, it is not so clear *a priori* for sine functions.

VI. Conclusions and Future Work

A new learning model of FNNs that maximizes the margin has been presented. The key idea of the model is a weighting of the sum-of-squares error function, which is inspired in the AdaBoost algorithm. The hardness of the margin can be controlled by a parameter, as in SVMs. The proposed model allows to deal with multiclass and multilabel problems in a natural way (as FNNs usually do), and it is not restricted to a "SVM architecture" nor to the use of kernel functions, independently of the concrete training algorithm. Both theoretic and experimental results have been shown confirming these ideas.

The weighting functions proposed in this work are not the only ones that can be used to weight the sum-of-squares error function. In this way, the approach must be tested in real world problems, and compared with both FNNs and SVMs. Although in this work we have only considered classification problems, the same idea can be applied to regression problems, just by changing the condition of the weighting function (4) from $mrg(x_i, y_i, f_{FNN}) \geqslant 0$ to $|f_{FNN}(x_i) - y_i| \leqslant \varepsilon$, where ε is a new parameter that controls the resolution at which we want to look at the data. This idea is similar to the $\varepsilon-$insensitive cost function proposed in [13].

References

[1] Bishop, C.M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press Inc., New York.

[2] Cristianini, N and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines*. Cambridge University Press, UK.

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

[4] Girosi, F. (1998). An Equivalence Between Sparse Approximation and Support Vector Machines. *Neural Computation* 10, 1455-1480.

[5] Joachims, T. (1999) Making large-Scale SVM Learning Practical. In *Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf and C. Burges and A. Smola (Ed.), MIT-Press.

[6] Leshno, M., Lin, V.Y., Pinkus, A and Schocken, S. (1993). Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximate Any Function. *Neural Networks* 6, 861-867.

[7] Park, J. and Sandberg, I.W. (1993). Approximation and Radial-Basis-Function Networks. *Neural Computation* 5 (2), 305-316.

[8] Raudys, S. (1998). Evolution and generalization of a single neuron: I. Single-layer perceptron as seven statistical classifiers. *Neural Networks* 11, 283-296.

[9] Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986). *Parallel Distributed Processing*, Vol 1. MIT Press.

[10] Sopena, J.M., Romero, E. and Alquézar, R. (1999). Neural Networks with Periodic and Monotonic Activation Functions: A Comparative Study in Classification Problems. *Proc. 9th Int. Conf. Artificial Neural Networks*, 323-328.

[11] Schapire, R.E. and Singer, Y. (1999). Improved Boosting Algorithms Using Confidence-rated Predictions. *Machine Learning* 17 (3), 297-336.

[12] Suykens, J.A.K and Vandewalle, J. (1999). Traning Multilayer Preceptron Classifiers Based on a Modified Support Vector Method. *IEEE Trans. on Neural Networks* 10 (4), 907-911.

[13] Vapnik, V. (1995). *The Nature of statistical learning theory*. Springer-Verlag, New York.

[14] Vapnik, V. (1998). The Support Vector Method of Function Estimation. In C. Bishop (Ed.), *Neural Networks and Machine Learning*, 239-268, Springer-Verlag, Berlin.

[15] Vincent, P. and Bengio, Y. (2000). A Neural Support Vector Architecture with Adaptive Kernels. *Int. Joint Conference on Neural Networks* 5, 187-192.

0-7803-7278-6/02/$10.00 ©2002 IEEE