[2] Y. Ota and B. M. Wilamowski, "CMOS architecture of synchronous pulse-coupled neural network and its application to image processing," in *Proc. 26th Ann. Conf. IEEE Ind. Electron. Soc. (IECON)*, Nagoya, Japan, Oct. 2000, pp. 1213–1218.
[3] D. Matolin, J. Schreiter, and R. Schüffny, "Implementierung eines neuronalen Netzwerkes zur Bildsegmentierung als Beispiel für die Realisierung von hochparallelen und fehlertoleranten Signalverarbeitungssystemen mittels analoger integrierter Schaltungen," in *Proc. Dresdner Arbeitstagung Schaltungs- und Systementwurf (DASS)*, Dresden, Apr. 2005, pp. 7–14.
[4] C. Koch, *Biophysics of Computation—Information Processing in Single Neurons*. Oxford, U.K.: Oxford Univ. Press, 1999.
[5] A. Heittmann and U. Ramacher, "An architecture for feature detection utilizing dynamic synapses," in *Proc. 47th IEEE Int. Midwest Symp. Circuits Syst.*, Jul. 2004, pp. II-373–II-376.
[6] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Comput.*, vol. 14, no. 11, pp. 2531–2560, 2002.
[7] C. Moore, C. Wilson, C. Mayer, S. Acquah, V. Massari, and M. Haxhiu, "A GABAergic inhibitory microcircuit controlling cholinergic outflow to the airways," *J. Appl. Physiol.*, vol. 96, pp. 260–270, 2004.
[8] R. Eckhorn, "Neural mechanisms of scene segmentation: Recordings from the visual cortex suggest basic circuits for linking field models," *IEEE Trans. Neural Netw.*, vol. 10, no. 3, pp. 464–479, May 1999.
[9] R. V. Rullen and S. J. Thorpe, "Rate coding versus temporal order coding: What the retinal ganglion cells tell the visual cortex," *Neural Comput.*, vol. 13, pp. 1255–1283, 2001.
[10] Z. Yang, A. F. Murray, F. Worgotter, K. L. Cameron, and V. Boonsobhak, "A neuromorphic depth-from-motion vision model with STDP adaptation," *IEEE Trans. Neural Netw.*, vol. 17, no. 2, pp. 482–495, Mar. 2006.
[11] C. Mayr and R. Schüffny, "Image pulse coding scheme applied to feature extraction," in *Proc. Image Vis. Comput.*, Dunedin, New Zealand, Nov. 2005, pp. 49–54.
[12] A. Linares-Barranco, G. Jimenez-Moreno, B. Linares-Barranco, and A. Civit-Balcells, "On algorithmic rate-coded AER generation," *IEEE Trans. Neural Netw.*, vol. 17, no. 3, pp. 771–788, May 2006.
[13] K. Shimonomura and T. Yagil, "A multichip aVLSI system emulating orientation selectivity of primary visual cortical cells," *IEEE Trans. Neural Netw.*, vol. 16, no. 4, pp. 972–979, Jul. 2005.

Comparing Support Vector Machines and Feedforward Neural Networks With Similar Hidden-Layer Weights

Enrique Romero and Daniel Toppo

***Abstract*—Support vector machines (SVMs) usually need a large number of support vectors to form their output. Recently, several models have been proposed to build SVMs with a small number of basis functions, maintaining the property that their hidden-layer weights are a subset of the data (the support vectors). This property is also present in some algorithms for feedforward neural networks (FNNs) that construct the network sequentially, leading to sparse models where the number of hidden units can be explicitly controlled. An experimental study on several benchmark data sets, comparing SVMs and the aforementioned sequential FNNs, was carried out. The experiments were performed in the same conditions for all the models, and they can be seen as a comparison of SVMs and FNNs when both models are restricted to use similar hidden-layer weights. Accuracies were found to be very similar. Regarding the number of support vectors, sequential FNNs constructed models with less hidden units than standard SVMs and in the same range as "sparse" SVMs. Computational times were lower for SVMs.**

***Index Terms*—Feedforward neural networks (FNNs), sparse models, support vector machines (SVMs).**

Manuscript received May 5, 2006; revised October 26, 2006 and December 27, 2006; accepted January 5, 2007. This work was supported by the Consejo Interministerial de Ciencia y Tecnología under Projects CGL2004-04702-C02-02 and TIN2006-08114.

E. Romero is with the Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Barcelona 08034, Spain (e-mail: eromero@lsi.upc.edu).
D. Toppo is with the I&C School of Computer and Communication Sciences, Swiss Federal Institute of Technology (EPFL), Lausanne 1015, Switzerland.

Digital Object Identifier 10.1109/TNN.2007.891656

I. Introduction

Support vector machines (SVMs) and feedforward neural networks (FNNs) are two alternative machine learning approaches for classification and regression problems with different inductive bias and very interesting properties (see, for example, [1] and [12]). Although both models have been developed from different backgrounds, they share a number of elements that allow to establish a direct correspondence between them. In fact, from a formal point of view, they are structurally similar, since both SVMs and FNNs induce an output function which is expressed as a linear combination of simple (basis) functions

$$f(x) = b + \sum_{k=1}^{N} \lambda_k h(\omega_k, x). \tag{1}$$

For SVMs, N is the number of support vectors, h is the kernel function, $\{\omega_k\}_{k=1}^{N}$ are the support vectors, and $\{\lambda_k\}_{k=1}^{N}$ are the coefficients found by the constrained optimization problem posed. For FNNs (fully connected with one hidden layer of units and linear output units), N is the number of units in the hidden layer, h is the activation function, $\{\omega_k\}_{k=1}^{N}$ are the hidden-layer weights, and $\{\lambda_k\}_{k=1}^{N}$ are the output-layer weights. The bias term b is common for both SVMs and FNNs.

The differences between the solutions obtained by both models lie in the way the elements of that linear combination (1) are found. This is a consequence of their respective inductive bias. The first important difference is related to the number of elements in the combination (number of support vectors for SVMs and number of hidden units for FNNs). Whereas for SVMs the number of support vectors is usually a result of the optimization problem posed, for FNNs the number of hidden units is usually fixed *a priori*. A second difference lies in the hidden-layer weights $\{\omega_k\}_{k=1}^{N}$. For SVMs, they are always a subset of the data (the support vectors), as a consequence of the optimization problem solved. For FNNs, in contrast, that property does not usually hold. Finally, the values of the output-layer weights $\{\lambda_k\}_{k=1}^{N}$ may be very different for the same training set, since different optimization problems are solved (the maximization of the margin for SVMs and the minimization of the sum-of-squares error for FNNs).

There exist, however, FNN models [4], [13], [9] that do not show all of these differences with respect to SVMs. In these models, the network is constructed sequentially, so that the number of hidden units is a result of the learning process rather than fixed *a priori* (for a review of sequential FNNs see, for example, [6]). In addition, hidden-layer weights are always a subset of the data, as usual for SVMs. These properties lead to sparse solutions where the number of elements in (1) can be explicitly controlled.

In practice, models with good performance and few basis functions are desired. One of the problems affecting SVMs is that a large set of support vectors is usually needed to form their output function, making it complex and computationally expensive for real-time applications. Recently, several alternative methods to build SVMs with a small number of basis functions have been proposed. Among them, the ν-SVM [10] and the "sparse" SVMs described in [5] maintain the property that the hidden-layer weights $\{\omega_k\}_{k=1}^N$ are a subset of the data.

This work focuses on the comparison of SVMs and the aforementioned sequential FNNs. An experimental study on several benchmark data sets for classification problems is presented. All the tested models have in common the property that their hidden-layer weights are a subset of the data. The goal is finding out whether FNNs are competitive with SVMs when both models are restricted to use similar hidden-layer weights. For SVMs, several approaches are tested, namely, the standard one-norm soft margin SVM, the ν-SVM [10], and the greedy (forward) selection of support vectors presented in [5]. The experiments were performed in the same conditions for all the models. To this end, the same kernel/activation functions were tested with the same training and test data sets, so that the set of simple functions $\{h(\omega_k, x)\}$ available to construct the output was the same. The model selection process was as similar as possible, taking into account that different methods need different parameters.

The accuracies obtained in the experiments were very similar for the different methods. Regarding the number of support vectors, sequential FNNs constructed models with less hidden units than standard SVMs (including ν-SVMs), and in the same range as the method presented in [5]. In addition, all the hidden-layer weights in the FNN models were also considered as support vectors by standard SVMs. Computational times were lower for SVMs.

II. Background

To fix notation, consider the classification task given by a data set $X = \{(x_1, y_1), \ldots, (x_L, y_L)\}$, where each instance x_i belongs to the input space $\mathbb{R}^N$, $y_i \in \{-1, +1\}^T$, and T is the number of classes. For two-class problems, usually $y_i \in \{-1, +1\}$.

A. SVMs

SVMs for classification can be described as follows [12]: the input vectors are mapped into a (usually high dimensional) inner product space through some nonlinear mapping ϕ, chosen *a priori*. In this space (the *feature space*), an optimal separating hyperplane is constructed. By using a (positive–definite) kernel function $K(u, v)$ the mapping becomes implicit, since the inner product defining 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 the SVM framework, an optimal hyperplane means a hyperplane with maximal normalized margin for the examples of every class. The normalized margin is the minimum distance to the hyperplane.

When the data set is not separable by a hyperplane (neither in the input space nor in the feature space), some tolerance to noise is introduced in the model. Using Lagrangian and Kuhn–Tucker theory, the maximal margin hyperplane for a binary classification problem given by a data set X is a linear combination of simple functions depending on the data

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

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

$$\begin{aligned} &\text{Maximize}_\alpha \ \sum_{i=1}^{L} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{L} y_i \alpha_i y_j \alpha_j K(x_i, x_j) \\ &\text{subject to} \ \sum_{i=1}^{L} y_i \alpha_i = 0 \qquad \text{(bias constraint)} \\ &\qquad 0 \leqslant \alpha_i \leqslant C, \qquad i = 1 \ldots L \end{aligned} \tag{3}$$

for a certain constant C. The points x_i with $\alpha_i > 0$ (active constraints) are named *support vectors*. An example is well classified if and only if its functional margin $y_i f_{\mathrm{SVM}}(x_i)$ with respect to f_{SVM} is positive. The cost function in (3) is (plus a constant) the squared norm of $f_{\mathrm{SVM}}(x) - y(x)$ in the reproducing kernel Hilbert space defined by $K(u, v)$.

The parameter C allows to control the tradeoff between the margin and the errors in the data set. By setting $C = \infty$, the hard margin hyperplane is obtained. The most usual kernel functions $K(u, v)$ are polynomial, Gaussian-like, or sigmoidal functions. It is worth noting that the kernel function depends on a certain parameter γ (that is, $K(u, v) := K_\gamma(u, v)$). In many implementations, b is treated separately (fixed *a priori*, for example) in order to avoid the bias constraint.

One of the problems affecting SVMs is that a large set of support vectors is usually needed to form their output, making it complex and computationally expensive for real-time applications. To overcome this problem, several models have been proposed in the literature. The ν-SVM, for instance, replaces the parameter C by a parameter $\nu \in [0, 1]$, which is an upper bound on the fraction of margin errors and a lower bound on the fraction of support vectors (that are asymptotically equal) [10]. The method described in [5] performs a greedy (forward) selection of the support vectors, together with an efficient computation of the vector (α_i) for the two-norm soft margin SVM. We will refer to this method as SVMs with reduced complexity (SVMRC). As a result of the forward selection process, the sparsity of the model is explicitly controlled.

B. Sequential FNNs Where the Hidden-Layer Weights Are a Subset of the Data

Unlike SVMs, the most usual cost function for fully connected FNNs with one hidden layer of N units and output linear units is the sum-of-squares error

$$\sum_{i=1}^{L} \frac{1}{2} (f_{\mathrm{FNN}}(x_i) - y_i)^2 \tag{4}$$

where

$$f_{\mathrm{FNN}}(x) = b_0 + \sum_{k=1}^{N} \lambda_k \varphi(\omega_k, x). \tag{5}$$

As usual, $\{\omega_k\}_{k=1}^N$ are the hidden-layer weights and $\{\lambda_k\}_{k=1}^N$ are the output-layer weights. The most common activation functions $\varphi(b, \omega, x)$ for the hidden units are sigmoidal for multilayer perceptrons (MLPs) and radially symmetric for radial basis function networks (RBFNs), although many other functions may be used [7], [8]. Biases can be seen as part of the hidden-layer weights. Similar to kernel functions, the activation functions φ usually depend on a certain parameter γ, named "gain factor" for sigmoidal functions or "width" for radial basis functions. The architecture of the network (i.e., connections, number of hidden units, and activation functions) is usually fixed in advance, whereas the weights are learned during the training process.

There exist, however, FNN models that construct the network sequentially, so that the number of hidden units is a result of the learning process rather than being fixed *a priori*. For a review of constructive FNNs see, for example, [6]. Among them, some models select the hidden-layer weights to be always a subset of the data, such as the orthogonal least squares learning algorithm [4], kernel matching pursuit with *prefitting* [13], and the sequential approximation with optimal coefficients and interacting frequencies algorithm [9]. In these models, an (implicit or explicit) orthogonalization of the output vectors of the hidden units is performed. The hidden-layer weights of the new hidden unit are selected taking into account the interactions of that hidden unit with the previously selected ones. These interactions are discovered by means of the optimal (in a least squares sense) output-layer weights, which can be analytically computed by solving a linear equations system (the normal equations). More precisely, in order to select

```
Algorithm
  repeat
    Increase by 1 the number of hidden units $N$
    for every example $\omega$ in the training set
      Assign $\omega$ to the weights of the $N$th hidden unit
      Compute the optimal output-layer weights
      Set $\omega_N := \omega$ if the output-layer weights are
        valid (their norm is bounded by a constant $C$)
        and the training error is minimized
    end for
    Assign $\omega_N$ to the weights of the $N$th hidden unit
  until the maximum number of hidden units is added
  Compute the optimal output-layer weights
end Algorithm
```

Fig. 1. Pseudocode for the SVSFNNs tested in this letter.

the hidden-layer weights of the new hidden unit, every input example in the data set is considered at every step as a hidden-layer candidate weights vector. It is then temporarily installed in the network and the optimal output-layer weights of the whole network are computed. The input example that allows a greater reduction of the whole error is selected. This selection procedure is identical to that of SVMRC. We will refer to these models as support vector sequential FNNs (SVSFNNs). Fig. 1 shows the pseudocode for the SVSFNNs used in this letter. It is equivalent to [4], [13], and [9] except for the constant C (see Section III-D).

III. Comparing SVMs and SVSFNNs

This section compares SVMs and SVSFNNs and explains the methodology followed in the experiments.

A. Motivation

Beyond the structural similarity of their respective output functions (2) and (5), several parallelisms between SVMs and SVSFNNs can be established. On the one hand, the number of terms of the approximation (support vectors or hidden units, respectively) is a result of the learning process itself (for SVSFNNs, it can be explicitly controlled). On the other hand, they share the property that the hidden-layer weights are always a subset of the data, leading to sparse solutions. Only those input examples that, according to their respective inductive bias, have some influence on their respective approximations are present in the obtained solutions. Therefore, the main difference between SVMs and SVSFNNs lies in the output-layer weights, which are found by solving different optimization problems (the maximization of the margin for SVMs and the minimization of the sum-of-squares error for SVSFNNs). Among all SVM models, SVMRC is the most similar to SVSFNNs, since their selection processes are identical.

The aim of this letter is to compare SVMs and SVSFNNs, investigating whether FNNs are competitive with SVMs when both models are restricted to use similar hidden-layer weights.

B. Compared Methods

We focused on classification tasks. For SVMs, we tested the standard one-norm soft margin SVM, the ν-SVM [10], and the SVMRC [5] (with $\kappa = 59$ and the rest of parameters to their default value). For SVSFNNs, the model described in Fig. 1 was tested.

C. Software

For the standard one-norm soft margin SVM and the ν-SVM, we used the LIBSVM software [3]. For SVMRC, we resorted to a MATLAB implementation.[1] For SVSFNNs, we developed our own implementation.

D. Methodology

- **Preprocessing**. Categorical attributes were converted to *dummy* variables. The rest of the attributes were scaled to mean zero and variance one (linear scaling in [0, 1] gave similar results).
- **Kernel and activation function**. SVM models were obtained using the Gaussian $e^{-\gamma\|\vec{u}-\vec{v}\|^2}$ and the 2° polynomial $(\gamma\vec{u}'\vec{v}+1)^2$ kernels. For SVSFNNs, equivalent activation functions (Gaussian RBF and polynomial MLP) were used to obtain the networks.
- **Parameters and model selection**. In order to perform the experiments in the same conditions for both SVMs and SVSFNNs, the following correspondence between the parameters of the respective models can be made.
  1) The γ parameter of the kernel can be considered equivalent to the MLP-gain factor or RBF-width of the activation function (see Section II).
  2) Similar to the restriction related to the C parameter for SVMs [see (3)], a hidden-layer candidate weights vector was not considered valid for SVSFNNs if the one-norm of the solution (the output-layer weights) of its associated linear equations system was greater than a certain value C (see Fig. 1). This can be seen as a form of regularization. For the ν-SVM, the parameter C is replaced by ν.

  In order to get adequate γ and C/ν parameters, a *grid search* was performed (with γ ranging from 2^{-10} to 2^5, C ranging from 2^{-5} to 2^{10} and ν ranging from 0.05 to 0.95). The parameters corresponding to the best tenfold cross-validation (CV) accuracy were kept to build the model. The same grid search was performed for all the models, and repeated for every kernel function.

  In order to select the number of support vectors/hidden units of the final models for SVMRC and SVSFNNs, we followed [5]. First, the maximum number of support vectors/hidden units $N_{\max}$ is chosen. Then, $N_{\max}$ models are constructed until $N_{\max}$ support vectors/hidden units are added. The number of support vectors/hidden units corresponding to the best CV accuracy defines the final size of the model.
- **Model training and testing**. The methods were trained and tested over 30 training-test different random partitions (90%–10% of the data) of the whole data set, except for the United States Postal Service *(USPS)* data set, where a fixed subset was used to test the models trained with the rest of the data (this test set was not used in the model selection process). In all cases, the training and test data sets were exactly the same for all the methods. The one-versus-one approach was used for multiclass problems.

E. Data Sets

Several benchmark data sets from the University of California at Irvine (UCI) repository [2] were used for the comparison, namely *Abalone*, *Australian Credit*, *Pima Indians Diabetes*, *German Credit*, *Hepatitis*, *Ionosphere*, and *Sonar*. The *USPS* data set was also tested.

For the *USPS* data set, 7291 and 2007 examples were used for training and testing, respectively, without scaling the data.

F. Experimental Results

Fig. 2 shows the results obtained using the methodology previously described for the UCI data sets. For the *USPS* data set, no overfitting was observed for sequential methods (SVMRC and SVSFNN), at least for the addition of the first 1000 support vectors/hidden units. Therefore, the results are qualitatively different from those of the rest of

[1]Available at http://www.kyb.tuebingen.mpg.de/bs/people/chapelle/primal.

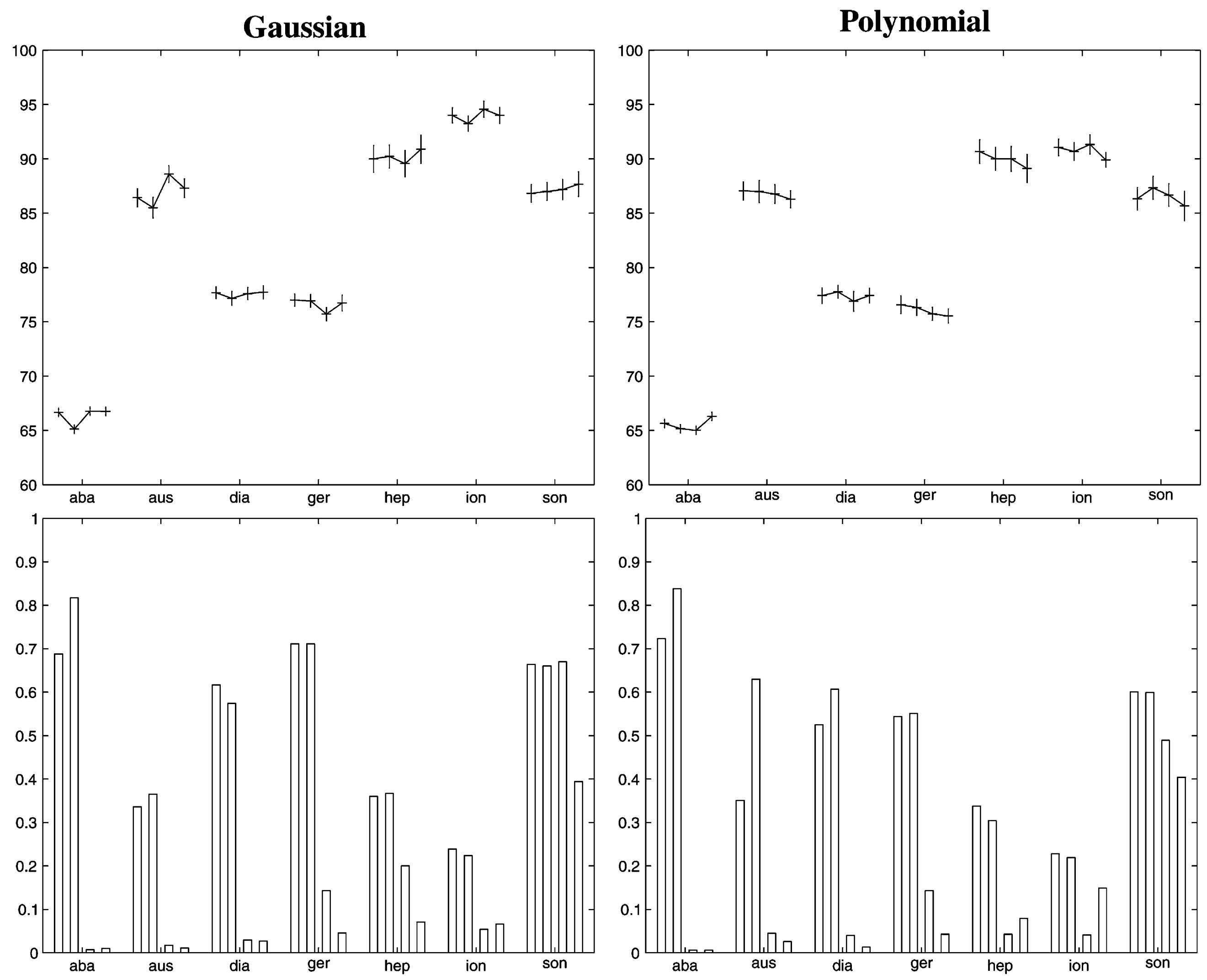

Fig. 2. SVM and SVSFNN results with the Gaussian and polynomial kernel/activation function for the UCI data sets tested: Abalone (aba), Australian credit (aus), Diabetes (dia), German credit (ger), Hepatitis (hep), Ionosphere (ion), and Sonar (son). Error bars (top) show the mean accuracies and the standard errors. Bars (bottom) show the average ratio of support vectors and hidden units (with respect to the number of training examples) for SVMs and SVSFNNs, respectively. For every data set, methods are ordered from left to right: SVM, ν-SVM, SVMRC, and SVSFNN.

the data sets. Test accuracies for standard SVMs and ν-SVMs with the Gaussian kernel were 95.57% and 95.37%, with 1777 and 1935 support vectors, respectively. To achieve similar accuracy, SVMRC and SVSFNN needed 77 and 236 support vectors/hidden units, respectively (with 1000 basis functions, accuracy was around 97.0% for both models). Fig. 3 shows the accuracies for SVMRC and SVSFNNs with respect to the first 512 basis functions.

Accuracies obtained were very similar for all the models (specially for the Gaussian kernel, which yielded the best results). Regarding the support vectors (hidden units), it is noteworthy that SVSFNNs obtained models with less hidden units than the number of support vectors found by standard and ν-SVMs, and similar to SVMRC. This number of support vectors are in the same range for every data set, regardless of the kernel/activation function used. In general, the average ratios of hidden units with respect to the number of training examples are quite low.

We observed that, for every data set and kernel/activation function, all the hidden-layer weights selected by an SVSFNN were also found among the support vectors of the corresponding standard SVM. This can be intuitively explained if we note that both the SVSFNN and SVM solutions are expressed as a sum of hyperplanes, each one defined by the inner product with an input example in the data set (the support vectors in the SVM model). Support vectors are near the decision boundary. Therefore, these examples are also likely to be selected as the most discriminating ones by SVSFNNs (recall that only the input

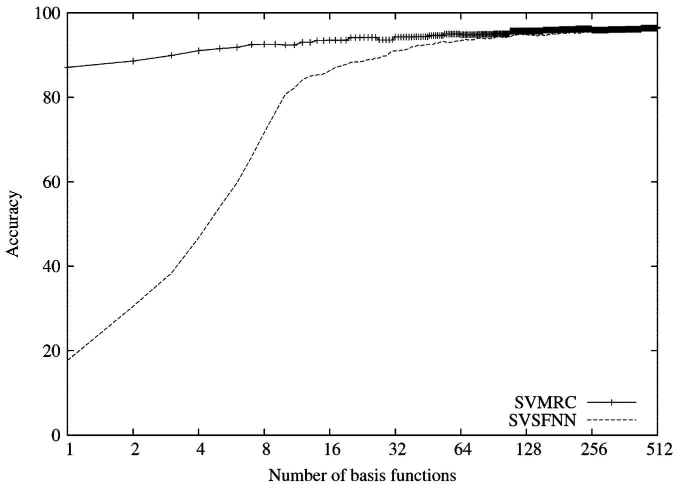

Fig. 3. Accuracies (in logarithmic scale) for SVMRC and SVSFNNs with respect to the first 512 Gaussian basis functions for the *USPS* data set.

examples in the data set are considered as the hidden-layer candidate weights vectors at every step).

For SVSFNNs, numerical problems were encountered in some cases (during the model selection step), motivated by the bad conditioning of the matrix of the associated linear equations systems. However, these numerical problems were always observed for parameter values far from the optimal ones. The same was observed for SVMRC.

In general, the computational times were lower for SVMs than for SVSFNNs. We cannot directly compare the execution times, because different methods were implemented with different programming languages and implementation optimizations. The publicly available LIBSVM software was implemented in C++ (although mainly in C). We used C to implement SVSFNNs. As previously mentioned, SVMRC was implemented in MATLAB. With these implementations, standard SVMs were the fastest method (up to ten times faster), followed by SVMRC. These figures change when the accuracy is compared in terms of the computational time (see [5]). SVMRC was approximately two times faster than SVSFNNs. As an example, the execution times of the *Australian* data set on a Pentium 4 CPU at 1.8 GHz were 13 s for LIBSVM, 97 s for SVMRC, and 140 s for SVSFNN.

IV. Conclusions and Future Work

The experiments in this letter can be seen as a comparison of the inductive bias of SVMs and FNNs when both models are restricted to use similar hidden-layer weights. In our experiments, FNNs obtained similar accuracies to SVMs, constructing models with less hidden units than standard SVMs, and in the same range as "sparse" SVMs. The hidden-layer weights associated to the hidden units in FNNs were always considered as support vectors by standard SVMs. Although computational times were lower for SVMs, they may be dependent on the particular implementation or code optimizations. Therefore, we conclude that FNNs are competitive with SVMs when both models are restricted to use similar hidden-layer weights.

Looking at the results, one might wonder if SVMs could be more influenced by the fact that hidden-layer weights are a subset of the data than by their inductive bias. To shed light on this issue, testing other models that also select their hidden-layer weights within the data, such as the relevance vector machine [11], would be worthwhile.

Note that the sequential FNNs tested in this letter can be directly extended in several ways that SVMs cannot (or at least not so easily). First, any activation function can be used, without the restriction of being a kernel function. In particular, any similarity measure could be used. Moreover, different units may have different activation functions, since the search process to obtain the parameters of the new hidden units is deterministic and independent on the others.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable suggestions.

References

[1] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York: Oxford Univ. Press, 1995.
[2] C. L. Blake and C. J. Merz, "UCI repository of machine learning databases," Univ. of California, Irvine, CA, 1998 [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html
[3] C. C. Chang and C. J. Lin, "LIBSVM: A library for support vector machines," 2002 [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm
[4] S. Chen, C. F. N. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Trans. Neural Netw.*, vol. 2, no. 2, pp. 302–309, Mar. 1991.
[5] S. S. Keerthi, O. Chapelle, and D. DeCoste, "Building support vector machines with reduced classifier complexity," *J. Mach. Learn. Res.*, vol. 7, pp. 1493–1515, 2006.
[6] T. Y. Kwok and D. Y. Yeung, "Constructive algorithms for structure learning in feedforward neural networks for regression problems," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 630–645, May 1997.
[7] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Netw.*, vol. 6, pp. 861–867, 1993.
[8] J. Park and I. W. Sandberg, "Approximation and radial-basis-function networks," *Neural Comput.*, vol. 5, pp. 305–316, 1993.
[9] E. Romero and R. Alquézar, "A sequential algorithm for feed-forward neural networks with optimal coefficients and interacting frequencies," *Neurocomput.*, vol. 69, pp. 1540–1552, 2006.
[10] B. Schölkopf, A. J. Smola, R. Williamson, and P. Bartlett, "New support vector algorithms," *Neural Comput.*, vol. 12, pp. 1207–1245, 2000.
[11] M. Tipping, "Sparse Bayesian learning and the relevance vector machine," *J. Mach. Learn. Res.*, vol. 1, pp. 211–244, 2001.
[12] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995.
[13] P. Vincent and Y. Bengio, "Kernel matching pursuit," *Mach. Learn.*, vol. 48, pp. 165–187, 2002.