

Small Models of Large Machines

Pramod Lakshmi Narasimha

FASTVDO Inc.
5840 Banneker Rd. Suite 270 Columbia, MD 21044
pramod@fastvdo.com

Sanjeev S. Malalur and Michael T. Manry

Department of Electrical Engineering,
University of Texas at Arlington
smalalur@uta.edu, manry@uta.edu

Abstract

In this paper, we model large support vector machines (SVMs) by smaller networks in order to decrease the computational cost. The key idea is to generate additional training patterns using a trained SVM and use these additional patterns along with the original training patterns to train a neural network. Results verify the validity of the technique.

Introduction

A key element in a pattern recognition system is the classifier. In nearest neighbor classifiers (NNCs) the input feature vector is compared to all the training patterns and assigned to the class to which the nearest training pattern belongs. The NNC approximates the Bayes classifier (Fukunaga 1990), but is computationally very expensive.

A multilayer perceptron (MLP) classifier approximates Bayes discriminant in a least squares sense (Ruck *et al.* 1990) and is computationally more economic than an NNC. However, it requires many training patterns in order to generalize and it is sensitive to initial values of the weights.

Support vector machines (SVMs) (Vapnik 1998) map the inputs to a large feature space, making use of classical regularization, and allowing a subset of the training patterns to be support vectors. SVMs can often develop good decision boundaries from small data sets. However, it is difficult to find the best kernel and kernel parameters (Brown *et al.* 2000), and the resulting structure is too large and redundant. Attempts have been made to reduce the size of the SVM (de Kruif & de Vries 2003) by deleting some patterns and by developing the Reduced Support Vector Machine (RSVM) (Lee & Huang 2007), pruning of the SVM (de Kruif & de Vries 2003) and the Relevance Vector Machine (RVM) (Tipping 2000). Unfortunately, networks resulting from these techniques are still large.

In this paper, we model SVMs by much smaller machines, in order to reduce computational complexity. First, we review the MLP and the SVM. Pattern memorization in SVMs is investigated next. We then present methods for compact modeling of the SVM. Finally, we provide numerical results and conclusions.

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Review Of Learning Machines

In this section we review MLP and SVM classifiers.

Multilayer Perceptron

Let $\{\mathbf{x}_p, \mathbf{t}_p\}_{p=1}^{N_p}$ be the training dataset where $\mathbf{x}_p \in \mathfrak{R}^N$ is the input vector, $\mathbf{t}_p \in \mathfrak{R}^M$ is the desired output vector and N_p is the number of patterns. Figure 1 depicts a feedforward MLP, having one hidden layer with N_h nonlinear units and an output layer with M linear units. For the p^{th} pattern, the j^{th} hidden unit's net function, $\{net_{pj}\}_{j=1}^{N_h}$, and activation, $\{O_{pj}\}_{j=1}^{N_h}$, are

$$net_{pj} = \sum_{i=1}^{N+1} w_h(j, i) \cdot x_{pi} \quad (1)$$

$$O_{pj} = f(net_{pj}) = \frac{1}{1 + e^{-net_{pj}}} \quad (2)$$

Weight $w_h(j, i)$ connects the i^{th} input to the j^{th} hidden unit. Here the threshold of the j^{th} node is represented by $w_h(j, N+1)$ and by setting $x_{p, N+1} = 1$.

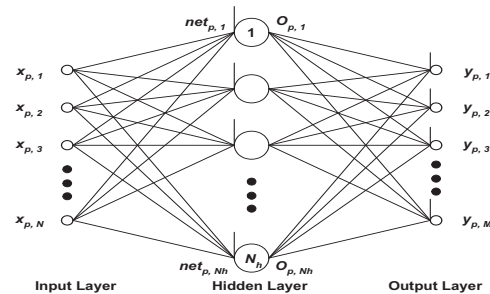


Figure 1: MLP with single hidden layer

The k^{th} output for the p^{th} pattern is given by

$$y_{pk} = \sum_{i=1}^{N+1} w_{oi}(k, i) \cdot x_{pi} + \sum_{j=1}^{N_h} w_{oh}(k, j) \cdot O_{pj} \quad (3)$$

where $1 \leq k \leq M$. Here, $w_{oi}(k, i)$ is the weight connecting the i^{th} input to the k^{th} output and $w_{oh}(k, j)$ is the weight connecting the j^{th} hidden unit to the k^{th} output.

The overall performance of a MLP network, measured as MSE, is

$$E = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{k=1}^M [t_{pk} - y_{pk}]^2 \quad (4)$$

In classification applications, $t_{pk} = \delta(k - d_p)$, where d_p is the correct class number for the p^{th} pattern.

In many neural net training algorithms, the goal is to minimize E . Example algorithms include back-propagation (Rumelhart & McClelland 1986), Levenberg Marquart (Fun & Hagan 1996), cascade correlation (Fahlman & Lebiere 1990) and constructive backpropagation (CBP) (Lehtokangas 1999).

Support Vector Machines

SVMs have the inherent ability to solve pattern-classification problem in a manner close to the optimum, using no problem domain knowledge (Haykin 1999). Here, we briefly review relevant properties of SVMs. Let the dimension of feature space be h_{svm} , which is also the number of Support Vectors (SVs). Note that h_{svm} is equivalent to number of hidden units in an MLP, as seen in figure 2.

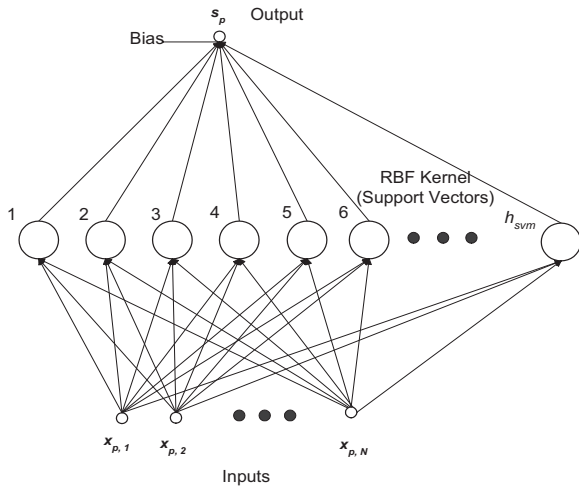


Figure 2: Support Vector Machine with RBF Kernel

Let us denote the SVM's training dataset by $\{\mathbf{x}_p, d_p\}_{p=1}^{N_v}$, where d_p is the class number for the p^{th} example. Let the set $\{\phi_j(\mathbf{x})\}_{j=1}^{h_{svm}}$ denote a nonlinear transformation from the input space to the feature space. In our case, the elements of the set are radial basis functions (RBF)

$$\phi_j(\mathbf{x}) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x} - \mathbf{x}_j\|^2\right) \quad (5)$$

for $1 \leq j \leq h_{svm}$. These $\phi_j(\mathbf{x})$ are equivalent to the hidden unit activation functions in the MLP. For the p^{th} pattern, the output of the SVM is

$$s_p = \sum_{j=0}^{h_{svm}} w_j \phi_j(\mathbf{x}_p) \quad (6)$$

where $\{w_j\}_{j=0}^{h_{svm}}$ denotes a set of linear weights connecting the feature space to the output space, $\phi_0(\mathbf{x}) = 1$ for all \mathbf{x} , so that w_0 denotes the bias. A decision is made on the given input depending on the output of the SVM. The pattern is classified as class one for s_p greater than zero, class two otherwise.

Structural Risk Minimization (SRM) Principle SVMs realize good generalization performance by finding the VC dimension (Vapnik 1998) where the minimum of the guaranteed risk (sum of training error and confidence interval) occurs. The main structural differences between MLPs and SVMs are

- SVMs do not have bypass weights connecting inputs directly to outputs, but MLPs can.
- There is only one output and two classes for SVMs whereas MLPs can handle multiple outputs and classes.

Memorization in Feedforward Networks

The storage capacity of a feedforward network is the number (N_v) of distinct input vectors that can be mapped, exactly, to the corresponding desired output vectors, resulting in zero error. The lower bound on memorization (or the upper bound on number of hidden units) is stated in the following theorem which is due to Sartori and Antsaklis (Sartori & Antsaklis 1991).

Theorem 1: For a feedforward network with N inputs, M outputs and N_h hidden units with arbitrary bounded nonlinear activation functions, at least N_h distinct patterns can be memorized. \square This can be expressed as

$$N_v \geq N_h \quad (7)$$

for networks having no bypass weights or output thresholds. For networks having bypass weights and the output thresholds, we generalize this to get

$$N_v \geq (N + N_h + 1) \quad (8)$$

Researchers are generally aware of the upper bound on pattern storage reflected in the following theorem.

Theorem 2 For a feedforward network with N inputs, M outputs and N_h hidden units with arbitrary bounded nonlinear activation functions, the number of distinct patterns that can be memorized obeys the inequality

$$N_v \leq \left\lfloor \frac{N_w}{M} \right\rfloor \quad (9)$$

where N_w is the number of weights in the network. \square

A proof for Theorem 2 is given in (Narasimha, Manry, & Maldonado 2007). The upper bound on the number of hidden units, derived by Elisseff and Moisy (Elisseff & Paugam-Moisy 1996) is agrees with (9).

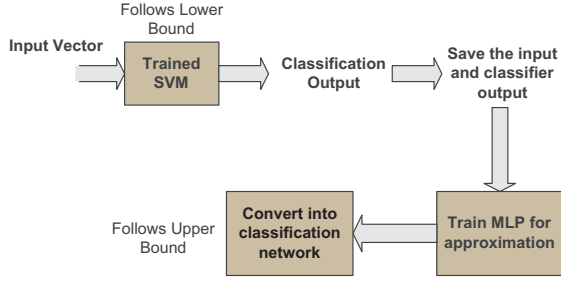


Figure 3: SVM modeling block diagram

Modeling of SVMs Through Memorization

For an SVM, patterns that are SVs generate zero squared error. Therefore the SVM memorizes h_{svm} patterns. This corresponds to the lower bound on memorization of Theorem 1, since h_{svm} is equivalent to N_h in an MLP. One can view the SVM training as a search for a subset of the training data that should be memorized. Unfortunately, h_{svm} tends to be almost as large as N_v .

Kruif and de Vries (de Kruif & de Vries 2003) have used pruning to reduce the training set size for function approximation using SVMs. However, this technique does not guarantee that the number of SVs is minimized. Also, the SVM’s ability to generate a good decision boundary is often thinly spread over thousands of SVs, so pruning of SVs does not help much. It is clear that SVMs make inefficient use of their weights.

One method for remedying this problem is to train a network that follows the upper bound on memorization. This results in a smaller network and results in faster processing. Figure 3 shows the block diagram of a modeling procedure. The idea is to save the output values of the SVM on the training dataset to obtain a new function approximation dataset. This new dataset has the same inputs as the original training file and the SVM outputs as the desired outputs $\{\mathbf{x}_p, s_p\}_{p=1}^{N_v}$. We now train a feedforward network to approximate the new dataset and the network obtained is denoted by $\mathcal{W}^{map} = \{w_{oi}^{map}, w_{oh}^{map}, w_{hi}^{map}\}$. In order to convert the network obtained into a classification network, with one output per class, all the weights are copied to the classification network without change. Then we add a second output unit and all the weights connecting to it are made equal to the negative of the corresponding weights connecting to the first output. That is

$$w_{oi}^{cls}(2, i) = -w_{oi}^{map}(1, i) \quad (10)$$

$$w_{oh}^{cls}(2, j) = -w_{oh}^{map}(1, j) \quad (11)$$

for $1 \leq i \leq N + 1$ and $1 \leq j \leq N_h$. In this way we obtain a feedforward classification network denoted by $\mathcal{W}^{cls} = \{w_{oi}^{cls}, w_{oh}^{cls}, w_{hi}^{cls}\}$. We now prune (Maldonado, Manry, & Kim 2003) this network to the desired number of hidden units and compare its performance to that of the original SVM.

Using SVM^{light} (Joachims 2004) which implements Vapnik’s SVM (Vapnik 1998), we have trained SVMs for

several two class problems. Comf18, is an image segmentation dataset (Bailey *et al.* 1993) with 18 inputs and 4 classes. Classes 1 and 2 are extracted from this dataset in order to get a binary classification problem. A prognostics dataset - F17C (Gore *et al.* 2005) has 17 inputs and 39 classes. Classes 3 and 6 are extracted from the dataset to get a binary classification problem. Gongtrn (Gong, Yau, & Manry 1994) has 16 inputs and 10 classes and corresponds to numeral recognition. The third two class problem is formed by extracting classes 5 and 10 from the numeral recognition dataset. The fourth dataset is from a speech phoneme recognition problem, which has 39 inputs and 34 classes. We extracted classes 1 and 4 from this dataset for our experiments. The fifth dataset is the Mushroom dataset (Iba, Wogulis, & Langley 1988) which consists of descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms. Each species is identified as edible or not.

Training and validation results for the SVMs and the MLPs are shown in Table 1. For the segmentation and prognostics datasets, the MLP validation errors are smaller than for those of the SVM. Sometimes, good training performance does not carry over to the validation case, as seen in the numeral and speech datasets. Observe that the number of support vectors needed for the SVM is much larger than the number of hidden units needed for the MLP. The number of hidden units shown in the table is calculated using Theorem 2.

Compact Modeling Of Large Classifiers

Although modeling through memorization works sometimes, it is unreliable. Good performance on training data is almost never a guarantee of good performance on validation data. Now, in order to minimize the validation error, the decision boundary of the MLP should be forced to converge to that of the SVM. One intuitive way of reaching the goal is by obtaining more training patterns for the MLP. Although new input vectors \mathbf{x}_p can be generated, desired outputs t_p usually cannot be. There is a way to sidestep this problem, however.

Bias-variance Decomposition Theory In (4), let $t_p \in \{+1, -1\}$, for the two class case. Let s_p be the output of the SVM, as before. Using Geman’s bias-variance decomposition theory (Geman, Bienemstock, & Doursat 1992), we can split the training error, E , as $E = E_1 + E_2$, where

$$E_1 = \frac{1}{N_v} \sum_{p=1}^{N_v} (t_p - s_p)^2 \quad (12)$$

$$E_2 = \frac{1}{N_v} \sum_{p=1}^{N_v} (s_p - y_p)^2 \quad (13)$$

Term E_1 is the variance of the expectational error of the considered model and is a constant with respect to the MLP weights. Rather than training a network on E , we can train it using E_2 . There may seem to be no advantage to this. However, note that N_v is fixed when we train with E , since we cannot generate correct desired outputs t_p for arbitrary new input vectors. For E_2 , however, we can generate as

Table 1: Modeling SVMs with MLPs.

Dataset	Patterns	SVM			MLP		
		SVs	% Training Error	% Validation Error	Hidden units	% Training Error	% Validation Error
Segmentation	4265	1095	7.43%	7.68%	33	6.27%	5.16%
Prognostics	238	190	5.88%	14.49%	19	0.42%	0.0%
Numeral	600	179	3.5%	5.17%	32	0.67%	8.5%
Speech	236	87	2.12%	2.63%	4	0.85%	7.89%
Mushroom	8124	228	0.0%	0.0%	20	0.0%	0.0%

many patterns as we want as follows. First, random vectors \mathbf{z}_p are generated, which may come from the joint pdf $f_{\mathbf{X}}(\mathbf{x})$ or some function of it. Then the vectors \mathbf{z}_p are processed by the SVM, producing scalar outputs s_p . So the limitation on the number of training patterns for the MLP is circumvented. Let N_{va} denote the number of randomly generated data patterns (\mathbf{z}_p, s_p) . E_2 can now be rewritten as

$$E_2 = \frac{1}{N_v + N_{va}} \sum_{p=1}^{N_v + N_{va}} (s_p - y_p)^2 \quad (14)$$

As we increase N_{va} , the MLP can be trained to mimic both the SVM's validation and training performances. Thus, small MLPs can mimic the performance of far larger SVMs. Next we present two methods for generating the additional patterns required for the MLP.

Volumetric Method

Here, the basic idea is to form a Voronoi tessellation of the training data's feature space and generate uniformly distributed random data points in this space. The number of data points generated for each cell is therefore made proportional to the volume of that cell. Let the training data be represented by $\{\mathbf{x}_p, d_p\}_{p=1}^{N_v}$. Using Self-Organizing Maps (SOM), we can group the inputs into K clusters. Let \mathcal{V}_k be the volume of the k^{th} cluster. The probability of generating the additional data point in cluster k is calculated by normalizing the cluster volume as

$$p_k = \frac{\mathcal{V}_k}{\sum_{n=1}^K \mathcal{V}_n} \quad 1 \leq k \leq K \quad (15)$$

Then the joint conditional probability density function, $f_{\mathbf{Z}}(\mathbf{z}|k)$, for new input vectors, \mathbf{z} , is set to be uniform, so

$$f_{\mathbf{Z}}(\mathbf{z}|k) = \begin{cases} \frac{1}{\mathcal{V}_k}, & \text{for } \mathbf{z} \in C_k; \\ 0, & \text{elsewhere.} \end{cases} \quad (16)$$

where C_k represents the k^{th} cluster. The overall pdf of the newly generated input vectors can be expressed as piecewise uniform distribution over different clusters. Using the probability of generating \mathbf{z} in cluster k , p_k computed in (15), we can write the overall pdf as

$$f_{\mathbf{Z}}(\mathbf{z}) = \sum_{k=1}^K f_{\mathbf{Z}}(\mathbf{z}|k)p_k \quad (17)$$

Additive Noise Method

Here, we generate new data by adding noise to the original input vectors. First, we calculate the standard deviation, σ_n , of each input. For each training vector \mathbf{x}_p , we add random vectors \mathbf{r}_p with zero mean ($\mathbf{m}_r = \mathbf{0}$) and element standard deviations given by $\sigma_{rn} = \sigma_n/10$. We generate new input vectors as $\mathbf{z}_p = \mathbf{x}_p + \mathbf{r}_p$. Passing them through the SVM, we obtain the output s_p . The new input pattern (\mathbf{z}_p, s_p) is thus formed. Let the pdfs of the original dataset, noise and the new dataset be represented by $f_{\mathbf{X}}(\mathbf{x})$, $f_{\mathbf{R}}(\mathbf{r})$ and $f_{\mathbf{Z}}(\mathbf{z})$. The pdf of \mathbf{Z} is obtained as

$$f_{\mathbf{Z}}(\mathbf{z}) = \int_{-\infty}^{\infty} f_{\mathbf{X}}(\mathbf{x}')f_{\mathbf{R}}(\mathbf{z} - \mathbf{x}')d\mathbf{x}' \quad (18)$$

Numerical Results and Analysis

Here we consider two class problems from different datasets and analyze the performances of SVMs and MLPs trained using constructive backpropagation (Lehtokangas 1999).

Methodology

First we generate additional patterns and combine them with the original training data. Then an MLP with one output is trained, using (14) to approximate the SVM and then converted into a classifier. Negated weights connecting to the first output are used for a second output. The behavior of the MLP is analyzed as the number of additional patterns increases. Depending on the size of the dataset, we split it into k parts and perform inverted k -fold cross-validation, with one part for training and $k - 1$ parts for validation. Hence the percentage error values we get here are different from the results of Table 1. We use CBP for training the MLP networks and the number of hidden units (N_h) is fixed from the upper bound of Theorem 2. In terms of complexity, if h_{svm} is the number of SVs in a trained SVM, then the number of multiplies required for processing one pattern is $(N + 4) \cdot h_{svm}$. Similarly, for a trained MLP with N_h hidden units and one output ($M = 1$), the number of multiplies is $((N + M + 2) \cdot N_h + MN) = (N + 3)N_h + N$. Although the expressions for the number of multiplies for processing a single pattern are linear functions of the number of SVs or the number of hidden units, from table 1 we know that $h_{svm} \gg N_h$. Thus the SVMs redundant structure has a significant overhead in processing as compared to the MLP.

Results

Figures 4 and 5 show percentage classification errors (average cross-validation error) versus N_{va} for the Numeral and Segmentation datasets respectively. Here, we have used the volumetric technique to generate additional patterns.

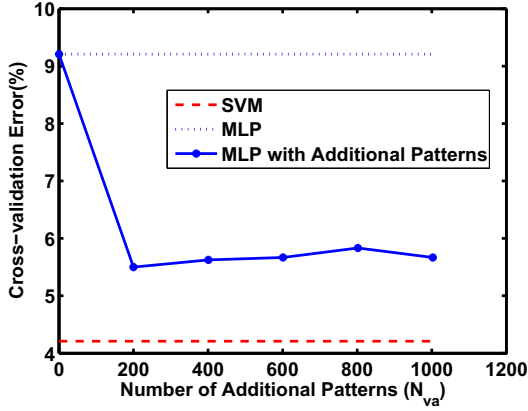


Figure 4: Volumetric Method, Numeral dataset

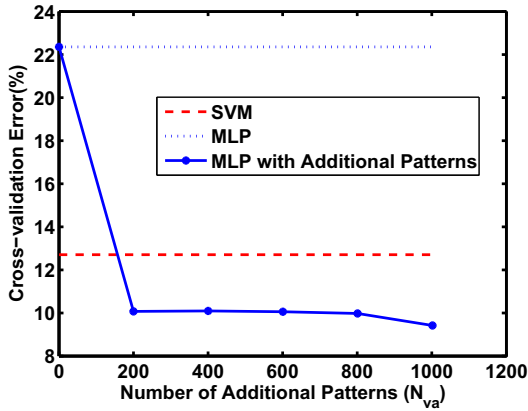


Figure 5: Volumetric Method, Segmentation dataset

Figures 6 and 7 show percentage classification errors (average cross-validation error) versus N_{va} for the Numeral and Segmentation datasets respectively. Here, we have used additive noise technique to generate the additional patterns.

We see that the SVM performs better than the MLP on the original datasets. As we increase the number of additional patterns included for training, the MLP's error decreases significantly. This indicates that the information extracted from the SVM in the form of additional patterns is being utilized constructively.

Table 2 shows the average cross-validation error percentage for the different datasets. The value of k , for a given dataset is indicated in the table. Both data generation techniques reduce the MLP error most of the time.

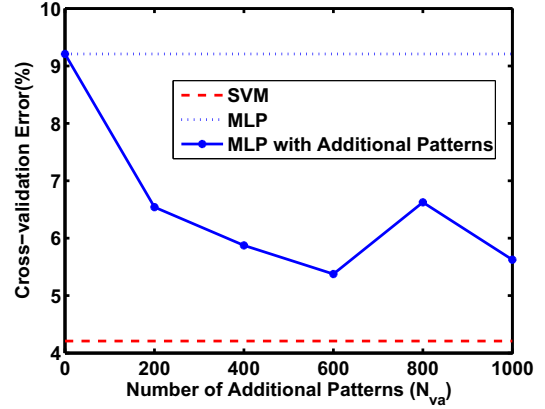


Figure 6: Additive Noise Method, Numeral dataset

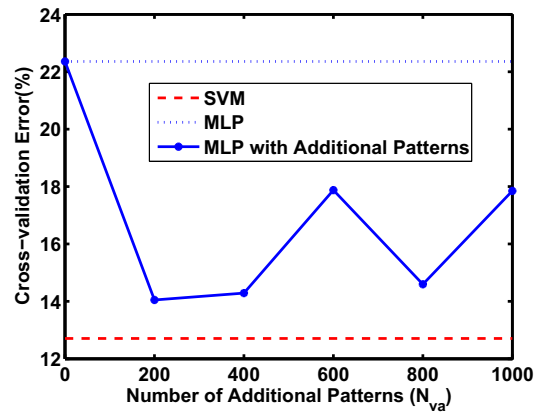


Figure 7: Additive Noise Method, Segmentation dataset

Effect of Incorrect Model

Let us consider two learning machines \mathcal{A} and \mathcal{B} where, \mathcal{A} performs better than \mathcal{B} on a particular dataset. If we include the additional patterns generated using \mathcal{B} to train \mathcal{A} , we are providing bad data for the better performing machine. This shows up as increased validation error. In such cases the additional patterns do not work constructively. This problem is discussed in detail with examples in (Sebe *et al.* 2005), where the authors discuss the theory behind this phenomenon and relate it to the Bias-Variance effect.

Discussion and Conclusion

We have developed a technique to obtain small models of large SVMs. We can attribute the success of our technique to the following:

1. SVMs are designed to follow the lower bound on pattern storage, so that memorization is not likely. This yields very large networks that exhibit small validation error and sometimes a near-optimal decision boundary. Few training patterns are required.
2. Efficient MLP training algorithms follow the upper bound

Table 2: Average validation error percentage.

Dataset	k fold	N_h	SVM Val Error %	MLP Val Error %	MLP Val Err% Additive Noise	MLP Val Err% Volumetric
Numeral	3	22	4.2	9.2	5.37	5.5
Segmentation	15	5	12.7	22.35	14.049	9.41
Flight Sim	3	20	6.34	7.8	7.8	5.85
Speech	5	2	9.76	18.8	18.8	18.08
Mushroom	50	6	4.91	8.20	7.15	4.89

on pattern storage, so that memorization is possible. This yields small networks that may exhibit large validation error. Avoiding large validation error is easier if there are many training patterns.

- Given an SVM with small validation error, random input vectors are put through it and combined with the SVM output decisions to give additional training patterns. Then following 2, we train a small MLP to mimic the near-optimal decision boundary of the SVM.

Two methods have been proposed for generating additional training data for an MLP which mimics the SVM. The small MLPs model the large SVMs fairly well.

References

Bailey, R. R.; Pettit, E. J.; Borochoff, R. T.; Manry, M. T.; and Jiang, X. 1993. Automatic recognition of usgs land use/cover categories using statistical and neural network classifiers. In *Proceedings of SPIE OE/Aerospace and Remote Sensing*.

Brown, M. P. S.; Grundy, W. N.; D. Lin, N. C. C. W. S.; Furey, T. S.; Jr., M. A.; and Haussler, D. 2000. Knowledge-based analysis of microarray gene expression data by using support vector machines. In *Proceedings of the National Academy of Science*, volume 97 of 1, 262–267.

de Kruif, B. J., and de Vries, T. J. A. 2003. Pruning error minimization in least squares support vector machines. *IEEE Transactions on Neural Networks* 14(3):696–702.

Elisseeff, A., and Paugam-Moisy, H. 1996. Size of multilayer networks for exact learning: Analytic approach. In *Neural Information Processing Systems*, 162–168.

Fahlman, S. E., and Lebiere, C. 1990. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems 2*, 524. Morgan Kaufmann.

Fukunaga, K. 1990. *Statistical Pattern Recognition, 2nd Ed.* Academic Press.

Fun, M. H., and Hagan, M. T. 1996. Levenberg-marquardt training for modular networks. In *IEEE International Conference on Neural Networks*, volume 1, 468–473.

Geman, S.; Bienemstock, E.; and Doursat, R. 1992. Neural networks and bias/variance dilemma. *Neural Computation* 4:1–58.

Gong, W.; Yau, H. C.; and Manry, M. T. 1994. *Progress in Neural Networks*, volume 2. Ablex Publishing Corporation. chapter Non-Gaussian Feature Analysis Using a Neural Network, 253–269.

Gore, R. G.; Li, J.; Manry, M. T.; min Liu, L.; Yu, C.; and Wei, J. 2005. Iterative design of neural network classifiers through regression. *International Journal on Artificial Intelligence Tools* 14(1-2):281–302.

Haykin, S. 1999. *Neural Networks A comprehensive Foundation, 2nd Edition.* Pearson Education.

Iba, W.; Wogulis, J.; and Langley, P. 1988. Trading off simplicity and coverage in incremental concept learning. In *In Proceedings of the 5th International Conference on Machine Learning*, 73–79.

Joachims, T. 2004. *SVMlight: Support Vector Machine.* Software available at <http://svmlight.joachims.org/>.

Lee, Y.-J., and Huang, S.-Y. 2007. Reduced support vector machines: A statistical theory. *IEEE Transactions on Neural Networks* 18(1):1–13.

Lehtokangas, M. 1999. Modelling with constructive backpropagation. *Neural Networks* 12:707–716.

Maldonado, F. J.; Manry, M. T.; and Kim, T.-H. 2003. Finding optimal neural network basis function subsets using the schmidt procedure. In *Proceedings of the International Joint Conference on Neural Networks*, volume 1 of 20-24, 444 – 449.

Narasimha, P. L.; Manry, M. T.; and Maldonado, F. 2007. Upper bound on pattern storage in feedforward networks. In *Proceedings of International Joint Conference on Neural Networks*, 1714 – 1719.

Ruck, D. W.; Rogers, S. K.; Kabrisky, M.; Oxley, M. E.; and Suter, B. W. 1990. The multilayer perceptron as an approximation to a bayes optimal discriminant function. *IEEE Transactions on Neural Networks* 1(4):296 – 298.

Rumelhart, D. E., and McClelland, J. L. 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. Cambridge, MA: MIT Press.

Sartori, M. A., and Antsaklis, P. J. 1991. A simple method to derive bounds on the size and to train multilayer neural networks. *IEEE Transactions in Neural Networks* 2(4):467–471.

Sebe, N.; Cohen, I.; Garg, A.; and Huang, T. 2005. *Machine Learning in Computer Vision*, volume 29. Springer.

Tipping, M. 2000. The relevance vector machine. In *Advances in Neural Information Processing Systems, San Mateo, CA.* Morgan Kaufmann.

Vapnik, V. N. 1998. *Statistical Learning Theory.* New York, Wiley.