

Upper Bound on Pattern Storage in Feedforward Networks

Pramod L. Narasimha, Michael T. Manry and Francisco Maldonado

Abstract—Starting from the strict interpolation equations for multivariate polynomials, an upper bound is developed for the number of patterns that can be memorized by a nonlinear feedforward network. A straightforward proof by contradiction is presented for the upper bound. It is shown that the hidden activations do not have to be analytic. Networks, trained by conjugate gradient, are used to demonstrate the tightness of the bound for random patterns. Based upon the upper bound, small multilayer perceptron models are successfully demonstrated for large support vector machines.

I. INTRODUCTION

Pattern memorization in nonlinear networks has been studied for many decades. The number of patterns that can be memorized has been referred to as the information capacity [2] and storage capacity [18]. Equating network outputs to desired outputs has been referred to as strict interpolation [5], [20], [7]. It is important to understand the pattern memorization capability of feedforward networks for several reasons. First, the capability to memorize is related to the ability to form arbitrary shapes in weight space. Second, if a network can successfully memorize many random patterns, we know that the training algorithm is powerful [16]. Third, some useful feedforward networks such as Support Vector Machines (SVMs), memorize large numbers of training patterns [10], [11].

Upper bounds on the number of distinct patterns P that can be memorized by nonlinear feedforward networks are functions of the number of weights in the network, N_w , and the number of outputs, M . For example, Davis [5] has shown that for any P distinct, complex points there exists a unique $(P-1)$ degree polynomial, with complex coefficients, that strictly interpolates (memorizes) all the points. In other words, breaking up the complex quantities into separate real and imaginary parts, he has derived a bound for the $M = 2$ case. An upper bound on the number of hidden units in the Multilayer Perceptron (MLP) for the $M = 1$ case, derived by Elisseff and Moisy [6], agrees with the bound of Davis. Suyari and Matsuba [21] have derived the storage capacity of neural networks with binary weights, using minimum distance between the patterns. Cosnard et al. [4] have derived upper and lower bound on the size of nets capable of computing arbitrary dichotomies. Ji and Psaltis [13] have derived upper and lower bounds for the information capacity of two-layer feedforward neural networks with binary interconnections, using an approach similar to that of Baum [3]. Moussaoui [1] and Ma and Ji [17] have

pointed out that the information capacity is reflected in the number of weights of the network.

Unfortunately, most recent research on pattern memorization in feedforward networks focuses on the one output case. In this paper, partially building upon the work of Davis [5], we investigate an upper bound for $M \geq 1$ and arbitrary hidden unit activation functions. In section II, we introduce our notation. A straightforward proof of the upper bound is given in section III. An example which indicates the validity of the bound is presented in section IV. In section V, we use the upper bound to predict the size of MLPs that can mimic the training behavior of SVMs.

II. NOTATION AND PRELIMINARIES

A. Notation

Let $\{\mathbf{x}_p, \mathbf{t}_p\}_{p=1}^P$ be the data set where $\mathbf{x}_p \in \mathbb{R}^N$ is the input vector and $\mathbf{t}_p \in \mathbb{R}^M$ is the desired output vector and P is the number of patterns. Let us consider a feedforward MLP, having N inputs, one hidden layer with h nonlinear units and an output layer with M linear units. For the p^{th} pattern, the j^{th} hidden unit's net function and activation are respectively

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

$$O_{pj} = f(net_{pj}) \quad (2)$$

Here, the activation $f(net)$ is a nonlinear function of the net function. The 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 is handled by fixing $x_{p, N+1}$ to one. The k^{th} output for the p^{th} pattern is given by

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

where $1 \leq k \leq M$. For the p^{th} pattern, the N input values are x_{pi} , ($1 \leq i \leq N$) and the M desired output values are t_{pk} ($1 \leq k \leq M$). w_{oi} are the weights connecting inputs to outputs and w_{oh} are the weights connecting hidden units to outputs.

B. Review

A feedforward network is said to have memorized a dataset if for every pattern, the network outputs are exactly equal to the desired outputs. Storage capacity of a feedforward network is the number (P) of distinct input vectors that can be mapped, exactly, to the corresponding desired output vectors resulting in zero error.

Pramod L. Narasimha and Michael T. Manry are with the Department of Electrical Engineering, University of Texas at Arlington, Arlington, TX 76013, USA (email: pramod.narasimha@uta.edu; manry@uta.edu).

Francisco Maldonado is with Williams Pyro, Inc., 200 Greenleaf Street, Fort Worth, Texas 76107. (email: javier.maldonado@williams-pyro.com)

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 [8] and Huang [9].

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

This can be expressed as

$$P \geq h \quad (4)$$

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

$$P \geq (N + h + 1) \quad (5)$$

In the next section, we consider an upper bound on the storage capacity.

III. AN UPPER BOUND

Researchers are generally aware that the number of distinct patterns P that can be memorized by a feedforward nonlinear network satisfies

$$P \leq \frac{N_w}{M} \quad (6)$$

where N_w is the number of weights in the network and M is the number of outputs. For example, the upper bound on the number of hidden units, derived by Elisseeff and Moisy [6] is based upon (6). They assume that the activation functions have continuous derivatives and have finite limits L^- in $-\infty$ and L^+ in $+\infty$.

In this section, we describe a functional link net approach for modeling the memorization process. Then an upper bound on memorization and a proof by contradiction are presented.

A. Monomial Activation Case

Let us assume monomial activation functions of degree d for the hidden units. In this case,

$$O_{pj} = (net_{pj})^d \quad (7)$$

$$= \left(\sum_{i=1}^{N+1} w_h(j, i) \cdot x_{pi} \right)^d \quad (8)$$

The above equation can be written as a power series in \mathbf{x}_p as in the Volterra filter. Let $\mathbf{X}_p = [X_{p1}, X_{p2}, \dots, X_{pL}]^T$ contain the multinomial combinations of the input variables for the p^{th} example. Here, L is the number of polynomial basis functions (PBFs) of the form $x_1^n \cdot x_2^m \cdot \dots \cdot x_N^q$. For N inputs and maximum degree d , the number of multivariate basis functions that can be derived is given by

$$L = \frac{(N + d)!}{N!d!} \quad (9)$$

The output in (3) can be written as a linear combination of these multinomial bases

$$y_{pk} = \sum_{i=1}^L a_{ik} \cdot X_{pi} \quad (10)$$

for $1 \leq k \leq M$ and $1 \leq p \leq P$, where P is the number of patterns. Note that the right hand side of (10) is a Gabor polynomial and is thus equivalent to the k^{th} output of a functional link net [19].

Our aim is to find the exact solution for the network weights. This requires replacement of y_{pk} with desired outputs t_{pk} and making P equal to L . Let us define the following matrices.

$$\begin{aligned} \mathbf{X} &= [\mathbf{X}_1^T, \mathbf{X}_2^T, \dots, \mathbf{X}_L^T]^T \\ &= \begin{pmatrix} X_{11} & X_{12} & \dots & X_{1L} \\ X_{21} & X_{22} & \dots & X_{2L} \\ \vdots & \vdots & \vdots & \vdots \\ X_{L1} & X_{L2} & \dots & X_{LL} \end{pmatrix} \end{aligned} \quad (11)$$

$$\begin{aligned} \mathbf{A} &= [\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_M] \\ &= \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1M} \\ a_{21} & a_{22} & \dots & a_{2M} \\ \vdots & \vdots & \vdots & \vdots \\ a_{L1} & a_{L2} & \dots & a_{LM} \end{pmatrix} \end{aligned} \quad (12)$$

$$\begin{aligned} \mathbf{T} &= [\mathbf{t}_1^T, \mathbf{t}_2^T, \dots, \mathbf{t}_L^T]^T \\ &= \begin{pmatrix} t_{11} & t_{12} & \dots & t_{1M} \\ t_{21} & t_{22} & \dots & t_{2M} \\ \vdots & \vdots & \vdots & \vdots \\ t_{L1} & t_{L2} & \dots & t_{LM} \end{pmatrix} \end{aligned} \quad (13)$$

Each row of \mathbf{X} stores the basis vector \mathbf{X}_p^T for one pattern. Similarly, each row of \mathbf{T} stores the desired output vector for one pattern. For all patterns and outputs, equation (10) can be written in a compact form as

$$\mathbf{X}\mathbf{A} = \mathbf{T} \quad (14)$$

where \mathbf{A} is an FLN coefficient matrix. We find the MLP network weights in two steps. In the first step, we solve the above equation for \mathbf{A} . If the inputs are all distinct, the columns of \mathbf{X} may be linearly independent so the rank of \mathbf{X} equals L . The coefficients a_{ki} are found by solving M sets of L equations in L unknowns,

$$\mathbf{A} = \mathbf{X}^{-1}\mathbf{T} \quad (15)$$

where the solutions are exact. The coefficients of the matrix \mathbf{A} can be expressed as polynomial functions of the unknown weights of the network. In the second step, we solve for the actual network parameters. Let $\mathbf{f}(\mathbf{w}) = \mathbf{A}$ where \mathbf{w} is a vector that contains all the network weights. The total number of network weights is N_w , hence $\mathbf{w} \in \mathbb{R}^{N_w}$. Let $\mathbf{f}_1(\mathbf{w}), \mathbf{f}_2(\mathbf{w}), \dots, \mathbf{f}_M(\mathbf{w})$ be the columns of the matrix $\mathbf{f}(\mathbf{w})$. Then for the i^{th} output we have

$$\mathbf{f}_i(\mathbf{w}) = \mathbf{a}_i \quad (16)$$

for $1 \leq i \leq M$. These equations may be solvable by using back substitution. Let us assume that we eliminate one

unknown weight by equating one element of the vector $\mathbf{f}_i(\mathbf{w})$ to the corresponding element of \mathbf{a}_i . Thus for each i we have

$$\begin{aligned} f_{1i}(\mathbf{w}) &= a_{1i} \\ f_{2i}(\mathbf{w}) &= a_{2i} \\ &\vdots \\ f_{Li}(\mathbf{w}) &= a_{Li} \end{aligned}$$

Note that it is not possible to solve the i^{th} set of equations in (16) separately from the rest. There is only one set of equations, since some weights affect more than one output. For each of the $L \cdot M$ elements of \mathbf{A} , we have one equation in N_w unknowns, which is

$$f_{ki}(\mathbf{w}) = a_{ki} \quad (17)$$

So if the number of unknowns N_w , is equal to $L \cdot M$, a solution that satisfies all the above $L \cdot M$ equations is possible. In other words, for an MLP, the storage capacity, that is the number of patterns that can be memorized by a network with h hidden units, is equal to the total number of weights in the network divided by the number of outputs,

$$P = \frac{N_w}{M} = \frac{h(N + M + 1) + M(N + 1)}{M} \quad (18)$$

There are several problems with this direct approach. The direct design of a memorization network would fail if the inverse of \mathbf{X} does not exist in (15). Also the validity of back substitution is questionable as closed form expressions for roots of high degree polynomials do not exist. Finally, we want a proof for activations other than monomial activations.

B. Arbitrary Hidden Activations

Here we relax the assumption that the activation function is a monomial and let it be arbitrary. Let us assume that the net functions for different finite inputs can be different but finite. Equating y_{pk} in (3) to t_{pk} , we get

$$t_{pk} = \sum_{i=1}^{N+1} w_{oi}(k, i) \cdot x_{pi} + \sum_{j=1}^h w_{oh}(k, j) \cdot O_{pj} \quad (19)$$

In order to model memorization above, we can use an exact, finite degree polynomial model for O_{pj} . This is accomplished by using Lagrange polynomials [12]. The $P - 1$ degree Lagrange polynomial that precisely satisfies $\mathcal{L}_j(\text{net}_{pj}) = O_{pj}$ is developed as

$$\mathcal{L}_j(\text{net}) = \sum_{p=1}^P O_{pj} \cdot l_p^j(\text{net}) \quad (20)$$

$$l_p^j(\text{net}) = \prod_{k=1, k \neq p}^P \frac{\text{net} - \text{net}_{kj}}{\text{net}_{pj} - \text{net}_{kj}} \quad (21)$$

The j^{th} hidden unit's activation function is precisely equal to the interpolation polynomial for our training patterns, so

$$O_{pj} = \sum_{n=0}^{P-1} \alpha_{jn} (\text{net}_{pj})^n \quad (22)$$

where α_{jn} are the Lagrange polynomial coefficients. Substituting equation (22) into equation (19), we get

$$\begin{aligned} t_{pk} &= \sum_{i=1}^{N+1} w_{oi}(k, i) \cdot x_{pi} \\ &+ \sum_{j=1}^h w_{oh}(k, j) \sum_{n=0}^{P-1} \alpha_{jn} (\text{net}_{pj})^n \end{aligned} \quad (23)$$

For the memorized training patterns, note that (23) is exact, rather than being an approximation of (19). As net_{pj} is a linear combination of the inputs, the expansion of $(\text{net}_{pj})^n$ results in polynomial combinations of the input variables of maximum degree $P - 1$. Thus, we get

$$t_{pk} = \sum_{i=1}^L a_{ki} \cdot X_{pi} \quad (24)$$

which is similar to the output equation obtained for the monomial activation case. The number of basis functions, L , will always be equal to P . With this, we can formally state our theorem on the storage capacity of feedforward networks.

Theorem 2 *For a feedforward network with N inputs, M outputs, h hidden units and arbitrary hidden activation functions, the number of patterns P that can be memorized with no error is less than or equal to number of absolute free parameters of the network, N_w divided by the number of outputs, M . \square*

In other words,

$$P \leq \left\lfloor \frac{N_w}{M} \right\rfloor \quad (25)$$

where $\lfloor \cdot \rfloor$ denotes truncation.

Proof Let us assume that (1) the network can memorize $(N_w/M) + 1$ patterns and (2) that we know the values of all the weights. There are two cases.

Case 1 (\mathbf{X} is singular for the given dataset): If the inverse of \mathbf{X} does not exist, assumption (1) is disproved and the theorem is confirmed for the given dataset.

Case 2 (\mathbf{X} is nonsingular): For this case, the proof continues to the second step. The number of nonlinear equations that needs to be solved is $PM = (N_w/M + 1)M = N_w + M$. So, we have $N_w + M$ equations in N_w unknowns.

Case 2.1 (Back-substitution eliminates exactly one equation and one unknown at a time):

Here, we solve exactly one equation at a time and substitute its solution into the remaining equations. By the end of this back substitution procedure, we will be left with M or more equations and no unknowns. Hence, the network weights that are already found must satisfy the remaining M equations in order to memorize $P = N_w/M + 1$ distinct patterns. This in general is not possible as we shall see in Case 2.2.

The last step in this sub-case is to show the validity of back-substitution in this proof. Without assumption (2), this

would be a daunting task. Note that in (17), each weight can occur many times. However, (17) can be simplified by substituting the correct value of the weight for most occurrences of the weight variable. As an example of (17), consider the equation

$$5w_1w_2^8 + 4w_3^3w_2^5 + 3w_4^8 = 43 \quad (26)$$

Solving for an occurrence of w_2 in the first term, we have

$$w_2 = \frac{43 - 4w_3^3w_2^5 - 3w_4^8}{5w_1w_2^7} \quad (27)$$

where the correct numerical value of w_2 is used on the right hand side. In back substitution, a solution of the form of (27) would be substituted into the remaining equations. Hence, we do not need a closed form expression for roots of an eighth degree equation.

Equation number N_w may be very complicated, but it will have only one unknown, and the method described above still works. For this first sub-case, there are M equations left over that cannot be solved. For this sub-case, therefore, the theorem is proved by contradiction.

Case 2.2 (In at least one back-substitution step, more than one equation is solved):

Suppose that two or more equations are solved by a back-substitution step. For the second equation solved, we have a constant on the left hand side and a FLN coefficient a_{ki} on the right hand side, in (17). We now change a_{ki} slightly so that the equation is not solved. Using equation (14), we now generate new desired outputs for our data file, without invalidating any of the equations already successfully solved. For this second sub-case, there are again M equations left over that cannot be solved. Therefore the theorem is proved by contradiction. \square

Since the proof uses only $h \cdot P$ values of the activation, its characteristics at other values do not matter. Therefore, the activations do not need to be continuous, analytic, or even bounded, in between the known points. Now that we have proved the theorem, we can investigate the tightness of the bound.

IV. DEMONSTRATION OF THE DERIVED BOUND

In this section, we experimentally demonstrate the tightness of the upper bound. We generated a dataset having fifteen inputs, two outputs and $P = 340$ patterns. All the inputs and outputs were Gaussian random numbers with zero mean and unit standard deviation. The number of hidden units required to memorize all the P patterns according to (18) and Theorem 2 satisfies

$$h \geq \frac{P \cdot M - M \cdot (N + 1)}{N + M + 1} \quad (28)$$

Plugging in $P = 340$, $N = 15$, and $M = 2$, we get

$$h \geq 36 \quad (29)$$

MLP networks of different sizes were trained using the conjugate gradient algorithm [15], [14], [16] on the data set. A plot of the MSE for different network sizes is shown in

Fig. 1. The plot shows the MSE for networks of different sizes starting from zero hidden units up to fifty at intervals of five. It is observed that the error curve changes dramatically from the linear network case (zero hidden units) to the neighborhood of $h = 35$ hidden units. After 40 hidden units, the change in error is negligible. This confirms that the network takes around $h = 36$ hidden units to memorize all the patterns, which agrees with Theorem 2. Unfortunately, the good performance of conjugate gradient on random data does not extend to the case of correlated data [16].

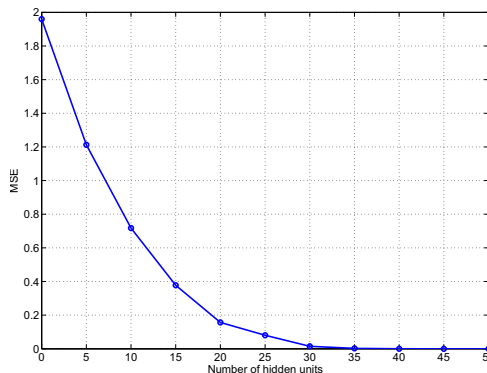


Fig. 1. MSE versus number of hidden units (h) for random data with fifteen inputs and two outputs with 340 patterns.

V. EFFICIENT MODELING OF SVMs

In this section, we investigate the memorization behavior of SVMs, and relate it to the upper and lower bounds on pattern storage.

A. SVMs

Here, we briefly review relevant properties of SVMs [10], [11]. Let the dimension of feature space be h_{svm} , which is also the number of Support Vectors (SVs). Radial Basis Function type inner-product kernels are used, which are denoted by $K(\mathbf{x}, \mathbf{x}_i)$ and defined by

$$K(\mathbf{x}, \mathbf{x}_i) = \sum_{j=0}^{h_{svm}} \phi_j(\mathbf{x})\phi_j(\mathbf{x}_i) \quad (30)$$

where

$$\phi_j(\mathbf{x}) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x} - \mathbf{x}_j\|^2\right) \quad (31)$$

for $1 \leq i \leq P$ and $1 \leq j \leq h_{svm}$. The main structural differences between SVMs and the MLPs of this paper are (1) unlike MLPs, bypass weights are not present in the SVMs (weights connecting inputs directly to outputs) and (2) there is only one output for SVMs whereas MLPs can handle multiple outputs.

The decision hyperplane constructed by the SVM for a given dataset is as follows:

$$\sum_{j=1}^{h_{svm}} w_j \phi_j(\mathbf{x}) + b = 0 \quad (32)$$

where $\{w_j\}_{j=1}^{h_{svm}}$ denotes a set of linear weights connecting the feature space to the output space, and b is the bias. A decision is made on the given input depending on sign of the SVM's output. If it is greater than zero, the pattern lies on the right side of the hyperplane and is classified to be of class one and if it is less than zero, the pattern lies on the left side of the hyperplane and is classified to be of class two. For a SVM, patterns that are SVs generate zero squared error. Therefore the SVM memorizes $P = h_{svm}$ patterns. This corresponds to the lower bound on memorization of Theorem 1 (see equation (4)).

B. Compact Modeling through Memorization

Kruif and Vries [22] have used pruning to reduce the training set size for function approximation using SVMs. They select those SVs that introduce minimum approximation error when omitted. However, this technique does not guarantee that the number of support vectors responsible for the computational complexity 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. Another approach for generating a small network is needed.

Since SVM pattern memorization follows the lower bound of Theorem 1, it is natural to try to duplicate this memorization using the upper bound as in equation (28). One approach 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 with the SVM outputs used as the desired outputs $\{\mathbf{x}_p, y_p^{svm}\}_{p=1}^P$, where y_p^{svm} is the output of the SVM for the p^{th} training pattern. 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 this network into a classification network, with one output per class, all the weights are copied to the classification network without any 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 unit. That is

$$w_{oi}^{cls}(2, i) = -w_{oi}^{map}(1, i) \quad (33)$$

$$w_{oh}^{cls}(2, j) = -w_{oh}^{map}(1, j) \quad (34)$$

for $1 \leq i \leq N + 1$ and $1 \leq j \leq h$. 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 [23] this network to the desired number of hidden units and compare its performance to that of the original SVM.

Using *LibSVM* [27], which implements Vapnik's SVM [11], we have trained SVMs on several two class problems. The first dataset, Segmentation, is an image segmentation dataset [26] with 18 inputs and 4 classes. Classes 1 and 2 are extracted from this dataset. A prognostics dataset - Prognostics [25] has 17 inputs and 39 classes. Here classes 3 and 6 are extracted from the dataset. Dataset Numeral [24] 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, which corresponds to numerals 4 and 9. The fourth dataset, Speech, is from a speech phoneme recognition problem. The speech samples are first preemphasized and converted to the frequency domain by taking the DFT. Then they are passed through Mel filter banks and the inverse DFT is applied on the output to get Mel-Frequency Cepstrum Coefficients (MFCC). Each of MFCC(n), MFCC(n)-MFCC(n-1) and MFCC(n)-MFCC(n-2) would have 13 features, which results in a total of $N = 39$ features. Each class corresponds to a phoneme. We extracted classes 1 and 4 from this dataset for our experiments.

Table I shows the results obtained. It should be noted that the training and validation errors are obtained from pruning the MLP classifier.

For the segmentation and prognostics datasets, the MLP validation errors are smaller, which indicates better generalization than SVM. For speech dataset, the validation errors for both SVM and MLP are equal, which indicates that both the networks have same generalization. Often, good training performance does not carry over to the validation case, as seen in the numeral recognition dataset.

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 in section III. In the case of dataset segmentation, we found that $h = 12$ is sufficient to get good generalization. Here we are not trying to memorize the training dataset and achieve zero training error, which would require a lot more hidden units.

VI. CONCLUSIONS

We have developed a simple proof of an upper bound on the storage capacity of feedforward networks with arbitrary hidden unit activation functions. The validity and tightness of the upper bound has been demonstrated through simulation. We have pointed out that SVMs attain a lower bound on pattern memorization. Their training performance can be modeled by much smaller networks that attain the upper bound. Examples are given that show this.

Future work includes improving the MLP model of the SVM by further training and pruning. Also, methods for improving generalization of the MLP will be developed.

REFERENCES

- [1] Y. S. A. Moussaoui, H. A. Abbassi, Hybrid hot strip rolling force prediction using a bayesian trained artificial neural network and analytical models, *American Journal of Applied Sciences* 3 (6) (2006) 1885–1889.
- [2] Y. S. Abu-Moustafa, J.-M. S. Jacques, Information capacity of hopfield model, *IEEE Transactions on Information Theory* IT-31 (4) (1985) 461–464.
- [3] E. B. Baum, On the capabilities of multilayer perceptrons, *Journal of Complexity* 4 (1988) 193–215.
- [4] M. Cosnard, P. Koiran, H. Paugam-Moisy, Bounds on the number of units for computing arbitrary dichotomies by multilayer perceptrons, *Journal of Complexity* 10 (1994) 57–63.
- [5] P. J. Davis, *Interpolation and Approximation*, Blaisdell Publishing Company, 1963.

Dataset	Patterns	SVM			MLP		
		SVs	% Training Error	% Validation Error	Hidden units	% Training Error	% Validation Error
Segmentation	4265	1422	6.53%	8.04%	12	6.59%	5.22%
Prognostics	238	212	0.0%	20.29%	19	0.0%	1.45%
Numeral	600	127	0.0%	5.67%	32	1.5%	5.83%
Speech	236	97	0.0%	5.26%	4	1.27%	5.26%

TABLE I
MODELING SVMs WITH MLPs.

- [6] A. Elisseeff, H. Paugam-Moisy, Size of multilayer networks for exact learning: Analytic approach, in: *Neural Information Processing Systems*, 1996.
- [7] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed., Pearson Education, 2004.
- [8] M. A. Sartori and P. J. Antsaklis, "A Simple Method to Derive Bounds on the Size and to Train Multilayer Neural Networks," *IEEE Transactions on Neural Networks*, vol. 2, no. 4, pp. 467-471, Jul. 1991.
- [9] G-B. Huang and H. A. Babri, "Upper Bounds on the Number of Hidden Neurons in Feedforward Networks with Arbitrary Bounded Nonlinear Activation Function," *IEEE Transactions in Neural Networks*, vol. 9, no. 1, pp. 224-229, Jan. 1998.
- [10] V. N. Vapnik, "An overview of Statistical Learning Theory," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 988-999, Sep. 1999.
- [11] V. N. Vapnik, *Statistical Learning Theory*, New York, Wiley, 1998.
- [12] H. Jeffreys, B. S. Jeffreys, *Methods of Mathematical Physics*, chap. Lagrange's Interpolation Formula, 3rd ed., Cambridge University Press, Cambridge, England, 1988, p. §9.011 pp. 260.
- [13] C. Ji, D. Psaltis, Capacity of two-layer feedforward neural networks with binary weights, *IEEE Transaction on Information Theory* 44 (1) (1998) 256-268.
- [14] E. M. Johansson, F. U. Dowla, D. M. Goodman, Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method, *Journal of Neural Systems* 2 (4) (1992) 291-301.
- [15] T. Kim, M. T. Manry, F. Maldonado, New learning factor and testing methods for conjugate gradient training algorithm, in: *International Joint Conference on Neural Networks*, vol. 3, 2003.
- [16] T.-H. Kim, J. Li, M. T. Manry, Evaluation and improvement of two training algorithms, in: *The 36th Asilomar Conference on Signals, Systems, and Computers*, 2002.
- [17] S. Ma, C. Ji, Performance and efficiency: Recent advances in supervised learning, *Proceedings of the IEEE* 87 (9) (1999) 1519-1535.
- [18] C. Mazza, On the storage capacity of nonlinear neural networks, *Neural Networks* 10 (4) (1997) 593-597.
- [19] Y.-H. Pao, Y. Takefuji, Functional-link net computing: Theory, system architecture, and functionalities, *IEEE Computer* 25 (5) (1992) 76-79.
- [20] M. J. D. Powell, Radial basis function approximations to polynomials, *Numerical Analysis 1987 Proceedings* (1988) 223-241.
- [21] H. Suyari, I. Matsuba, New approach to the storage capacity of neural networks using the minimum distance between input patterns, in: *International Joint Conference on Neural Networks*, vol. 1, 1999.
- [22] B. J. de Kruif and T. J. A. de Vries, "Pruning Error Minimization in Least Squares Support Vector Machines," *IEEE Transactions on Neural Networks*, vol. 14, no. 3, pp. 696-702, May 2003.
- [23] F. J. Maldonado, M. T. Manry and T.-H. Kim, "Finding optimal neural network basis function subsets using the Schmidt procedure," *International Joint Conference on Neural Networks*, vol. 1, pp. 444-449, Jul. 2003.
- [24] W. Gong, H. C. Yau and M. T. Manry, "Non-Gaussian Feature Analyses Using a Neural Network," *Progress in Neural Networks*, vol. 2, pp. 253-269, 1994.
- [25] R. Gore, J. Li, M. T. Manry, L. M. Liu, C. Yu and J. Wei, "Iterative Design of Neural Network Classifiers Through Regression," *International Journal on Artificial Intelligence Tools*, vol. 14, no. 1&2, 2005.
- [26] R.R. Bailey, E. J. Pettit, R. T. Borochoff, M. T. Manry, and X. Jiang, "Automatic Recognition of USGS Land Use/Cover Categories Using Statistical and Neural Network Classifiers," *Proceedings of SPIE OE/Aerospace and Remote Sensing*, 1993.
- [27] C. Chang and C. Lin. LIBSVM: A library for support vector machines. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>, 2001.