3-22-1996

Function approximation using a sinc neural network

Wael R. Elwasif

Laurene V. Fausett

PROCEEDINGS OF SPIE

SPIEDigitalLibrary.org/conference-proceedings-of-spie

Function approximation using a sinc neural network

Wael R. Elwasif
Laurene V. Fausett

SPIE.

Invited Paper

Function approximation using a *sinc* neural network

Wael R. Elwasif and Laurene V. Fausett

Applied Mathematics Program, Florida Institute of Technology
Melbourne, Florida 32901-6988

ABSTRACT

Neural networks for function approximation are the basis of many applications. Such networks often use a sigmoidal activation function (e.g. *tanh*) or a radial basis function (e.g. gaussian). Networks have also been developed using wavelets. In this paper, we present a neural network approximation of functions of a single variable, using *sinc* functions for the activation functions on the hidden units. Performance of the *sinc* network is compared with that of a *tanh* network with the same number of hidden units. The *sinc* network generally learns the desired input-output mapping in significantly fewer epochs, and achieves a much lower total error on the testing points. The original *sinc* network is based on theoretical results for function representation using the Whittaker cardinal function (an infinite series expansion in terms of *sinc* functions). Enhancements to the original network include improved transformation of the problem domain onto the network input domain. Further work is in progress to study the use of *sinc* networks for mappings in higher dimension.

Keywords: Neural networks, Function approximation

1 INTRODUCTION

The use of feedforward neural networks for continuous function approximation is one of the areas of neural network research that has a rich history of theoretical results that can be traced back to the work by Kolmogorov.[6] It was not until the development of the backpropagation training algorithm[8] that practical applications began to emerge. In most of these implementations the networks used some form of sigmoidal activation function mainly due to biological and historical reasons. Much research was directed towards studying the properties of sigmoidal functions and their representation capabilities when used as activation functions in feedforward neural networks. The study of the conditions under which a general function $f(x)$ can be used as the activation function in a feed forward neural network has also received attention. Considerations such as hardware implementation, speed of convergence of training algorithms, and required number of neurons to achieve a certain degree of accuracy were prime motives behind the search for alternative activation functions that match or outperform sigmoidal functions while reducing the overall cost of the network (storage and training time.)

In this paper, we consider the performance of a feedforward neural network that uses the *sinc* function as the hidden layer activation function. The *sinc* function has many computationally advantageous properties that render it extremely useful in different areas of numerical analysis. Furthermore, the *sinc* function can be viewd as a radial basis function or as the mother wavelet in a wavelet expansion of functions. This paper is organized as follows: in section 2 some aspects of the theoretical foundations for the use of various functions in a feedforward architecture are discussed. The basic representaion properties of the *sinc* function are outlined and the network based on such properties is presented. In section 3, simulation results are presented. A modified preprocessing

 0-8194-2141-3/96/$6.00

mapping function is proposed and network performance using this modified mapping is compared to that of the basic network for some test functions. Conclusions and suggestions for further research are given in section 4.

2 THEORETICAL BACKGROUND

2.1 Neural networks as function approximators

The theoretical basis for the use of artificial neural networks for function approximation can be traced back to the Superposition Theorem of of Kolmogorov.[6] Sprecher,[9,10] extended the results to use a single continuous function ψ that is independent of the number of variables n in the sum. However, the functions in the theorem are highly nonsmooth[5] which limits the practical application of these results. Furthermore, in neural networks, emphasis is placed more on the approximate representation of a particular mapping as opposed to its exact representation. Of the functions used in that area the (generalized) sigmoid function $\sigma(x) : \mathbb{R} \rightarrow (0,1)$ such that $\lim_{x\rightarrow\infty} \sigma(x) = 1$ and $\lim_{x\rightarrow-\infty} \sigma(x) = 0$ received considerable attention. Cybenko[2] demonstrated that sums of the form $\sum_{i=1}^{N} \alpha_i \sigma(y_j^T x + \theta_i)$ are dense in the space of continuous functions on the unit hypercube if σ is any continuous sigmoidal function (not necessarily monotone.) Chen et al.[1] argued that the boundedness of the generalized sigmoidal function is the most important factor in using it in neural network function approximators.

Other types of functions were also considered for use in neural network function approximation. Kreinovich[7] studied the use of arbitrary smooth (at least three times differentiable) general function $g(x)$ along with linear elements to approximate continuous mapping from $[-X, X]^m$ to $\mathbb{R}^n$ to an arbitrary degree of accuracy. Logarithmic activation functions have also been used.[4] Hidden units with localized receptive fields have been considered as an alternative activation function. The *sinc* function used in this paper can be viewed as a radial basis function or as a general activation function in a traditional feedforward network. In this paper only its latter use is considered and compared with networks based on the sigmoidal *tanh* function.

2.2 The Whittaker cardinal function

The Whittaker cardinal function[12,13] uses a family of *sinc* functions as a basis for expanding functions in the real as well as complex domain. This function has played a major role in numerical algorithms and in approximation theory. In this section, a brief discussion of the relevant properties of the function is presented.[11]

The Whittaker cardinal function $C(f,h)$ is defined by

$$C(f,h)(x) = \sum_{j=-\infty}^{\infty} f(jh)\, S(j,h)(x), \tag{1}$$

whenever this series converges, where h is the step size and

$$S(j,h)(x) = \frac{\sin[(\pi/h)(x-jh)]}{(\pi/h)(x-jh)} \tag{2}$$

is a *sinc* function. The parameter j controls the position of the *sinc* function while the parameter h controls its steepness as can be seen in Figure 1 and Figure 2.

Function approximation with the Whittaker cardinal function uses the function $S(j,h)(\phi(x))$ as a basis function, where $\phi(x)$ is a suitable transformation of the input interval onto $\mathbb{R}$. The accuracy obtained is roughly the

same whether or not the interval on which approximation is required has a singularity at an endpoint, which is not true of other expansions. Using the basis function $S(j,h)(\phi(x))$, let

$$\begin{aligned} z_j &= \psi(jh), \qquad j = 0, \pm 1, \pm 2, \ldots, \text{where} \\ \psi(w) &= \phi^{-1}(w). \end{aligned}$$

For example, for the interval $(0,1)$, the mapping can take the form

$$\phi(x) = \log \frac{x}{1-x}, \qquad \psi(x) = \frac{e^w}{1+e^w}, \qquad z_j = \frac{e^{jh}}{1+e^{jh}}. \tag{3}$$

The approximation of f over $(0,1)$ takes the form

$$f(x) \approx \sum_{j=-N}^{N} f(z_j)\, S(j,h)(\phi(x)) \tag{4}$$

It is worth noting that under some conditions, the approximating properties of the Whittaker function can be extended to the complex plane (see[11] for details).

2.3 Feedforward network using Sinc functions

The architecture for approximation of functions of one variable is directly derived from (4). This approximation relation corresponds to a feedforward network. The weights connecting the input unit to all units in the hidden layer are unity. Each hidden unit receives a bias weight equal to jh where j is the index of the hidden unit, $(j = -N, \ldots, N)$ and h is a fixed parameter. The weights connecting the hidden units to the output unit (w_j) are the only trainable weights in the net.

The input to the network is the function $\phi(x)$, where $\phi(x)$ maps the input domain onto $\mathbb{R}$. For approximation over the interval $(0,1)$, we can use $\phi(x) = \log \frac{x}{1-x}$. This means that the network implements the function

$$y = \sum_{j=-N}^{N} w_j \, \frac{\sin[(\pi/h)(\phi(x) - jh)]}{(\pi/h)(\phi(x) - jh)} \tag{5}$$

The output unit performs summation on the incoming signals (identity activation function). In the following section, results obtained using the above architecture are presented. Several functions $\{f(x) : f : (0,1) \rightarrow \mathbb{R}\}$ are considered. The procedure can be expanded to incorporate general functions defined on interval (a,b) by using a simple linear transformation.

3 NETWORK PERFORMANCE

The network is trained using plain vanilla backpropagation with online weight update. Network performance is compared with that using a sigmoid ($tanh$) function. The network is trained and tested using samples from different functions in addition to noisy training data sets. The standard mapping $\phi(x)$ is used to transform the input from $(0,1)$ onto $\mathbb{R}$ as given in (3).

3.1 Results for square function

The network is trained using samples from the function $f(x) = (10(x-0.5))^2$ on the interval $(0,1)$. This function is symmetric about the line $x = 0.5$ and the network input (the function $\phi(x)$) is symmetric about the

line $\phi(x) = 0.0$. The following results were obtained using a training set of 51 equally spaced points with the leftmost point at $x = 0.001$ and the rightmost point at $x = 0.999$. The function $\phi(x)$ maps those input values onto values in the interval $[-6.90675, 6.90675]$ which are then used as input to the net. The value of h is fixed at 1 and a learning rate $\alpha = 0.1$ is used to adjust the hidden-to-output weights. In Figure 3 the network output is plotted against network input ($\phi(x)$). As seen in the figure, for $h = 1$, the network approximates the function fairly well except for regions near the two end points. This is partly due to the fact that with this choice of h, the set of *sinc* functions implemented by the hidden layer does not properly "cover" the interval $[-6.90675, 6.90675]$. The separation between the centers of any two consecutive *sinc* functions in the hidden layer is h; this means that for $N = 5$, the leftmost unit has its center at -5 while the rightmost unit has it at 5.

In what follows, a method is proposed to select h such that the network input domain is covered by the hidden units. Let $[a, b]$ denote the domain of the function $f(x)$, where $[a, b] \subset (0, 1)$. Then $\phi(x)$ maps $[a, b]$ onto the interval $[c, d]$. To guarantee complete covering of $[c, d]$ using the available number of hidden units $(2N + 1)$, let

$$h = \frac{\max\{|c|, |d|\} * 2 * K}{2N} \tag{6}$$

where K is a constant greater than one that serves to extend the coverage an arbitrary distance beyond the boundary points c and d. In this work, K is taken to be 1.1. The result of using the value of h from the previous formula with the same number of hidden units is shown in Figure 3. Network performance using this value of h is greatly superior to that using $h = 1$, particularly near the end points. The network output was obtained using patterns with separation of 0.001 in the interval $(0, 1)$. The graphs show that the network generalizes nicely between training points (except for regions near the end points.)

The training history of the network is depicted in Figure 4. It is noted that the error drops rapidly for both values of h. However for $h = 1$, the steady state of the error is much larger than that for the computed value of h. The speed of convergence of the network is a major factor especially when compared with the (usually slow) convergence of a network based on sigmoidal functions.

As shown in Figure 5, the performance of the *sinc* and *tanh* networks is almost identical for points in the input domain not close to the end points. On the other hand, the *sinc* network converged faster than the *tanh* network which experienced some oscillations in the error early during learning as seen in Figure 6. It should be noted that the learning rate for the *tanh* network was $\alpha = 0.01$ since higher values lead to network instability and overshooting.

3.2 Function with Singularities at End Points

It has been suggested[11] that the performance of the *sinc* function is superior to that of other basis functions for approximating functions having singularities at the end points. The function $f(x) = \frac{1}{x\,(1-x)}$ defined on $(0, 1)$ has singularities at the two end points. A training set of 51 equally spaced points between 0.001 and 0.999 was used to train a *sinc* network with $N = 5$ and a learning rate $\alpha = 0.1$. The parameter h was computed using (6). The network was then tested using 1000 points equally spaced within the same interval. As shown in Figure 7, the network was successful in learning all training points (the leftmost and rightmost points are not shown on the graph since the values of the output at those points are too high). The net was also capable of successfully interpolating between the training points, except for regions close to the boundaries where the error between the desired ouput and network output becomes significantly large.

The same training set was used to train a single hidden layer feedforward network using the *tanh* activation function. There were 11 hidden units (as in the *sinc* network) and a learning rate $\alpha = 0.001$ (higher rates caused network instability). The training error history for both networks is shown in Figure 8. The performance of the *sinc* network is greatly superior to that of the *tanh* net. The error for both nets dropped with learning, however

the steady state value for the *tanh* network was much larger than that for the *sinc* net.

3.3 Function with Added Noise

The ability of the *sinc* network to extract a function from noisy input is investigated. To test this property, and compare it to that of the *tanh* network, the function $f(x) = (6(x - 0.3))^3 + S$ was used, where S takes random values uniformally distributed in the interval $[-5, 5]$. A training set of 101 points equally distributed in the interval $[0.001, 0.999]$ was used to train both networks. Each network used 11 hidden units; h was determined using (6). A learning rate α of 0.1 was used for the *sinc* network while the *tanh* network was trained using a learning rate of 0.01. Both networks were then tested using 1000 patterns uniformally distributed in the interval $[0.001, 0.999]$. As shown in Figure 9, the *sinc* network learned the underlying function without being overly distracted by the added noise. Performance degraded near the end points (due to absence of training points) but otherwise was satisfactory. The performance of the *tanh* network is shown in Figure 10. As can be seen, the added noise had some influence on the network output (see the oscillations around $\phi(x) = 3$. The performance near the end points was better than that for the *sinc* network, although the output "leveled" for negative inputs and did not track the training data. The learning error history for both networks is depicted in Figure 11.

3.4 Function with oscillations

The two networks were also trained to approximate a function that contains oscillations to test their capability to capture small details in the input/output mapping. For this purpose, the following function was used:

$$\begin{aligned} f(x) &= (1 - g(x))^3 + 6\ \sin(4g(x)) + 5\ \cos(10g(x)) + 40\ \sin(0.1g(x)), \\ \text{where } g(x) &= 6(x - 0.5). \end{aligned} \tag{7}$$

A training set of 101 patterns equally distributed in the interval $[0.001, 0.999]$ was used for training (after preprocessing by the function ϕ). For this function a network having 31 hidden units ($N = 15$) was used, with the parameter h evaluated using (6). For the aforementioned number of units, h was calculated to be 0.506495. The learning rate was $\alpha = 0.1$. The network was then tested using 1001 points equally distributed in the interval $[0.001, 0.999]$. The results in Figure 12 show that the network was able to capture the general trends in the mapping and the small details (to some extent). Also the performance degraded near the two end points (as was the case in previous functions) due to the absence of training points in those regions.

A *tanh* network with the same number of hidden units was also trained using the same training data and a learning rate of $\alpha = 0.01$. The learning error history of both networks is shown in Figure 13. As seen in the figure, the steady state error for the *tanh* network is much larger than that for the *sinc* network, indicating that the *sinc* network outperformed the *tanh* network in terms of learning error.

3.5 Improving Performance Near End Points

As seen from results presented earlier, the interpolating capability of the *sinc* network deteriorates for inputs near the boundary points of the input interval. This is due in part to the non-linear nature of the mapping $\phi(x) = \log(\frac{x}{1 - x})$. For regions of the input sufficiently close to 1 or 0, the separation between network input points generated through the mapping ϕ is considerably greater than the separation for other regions in the input interval (original input interval is (0,1) before preprocessing via $\phi(x)$). This means that although the original input points may be equally spaced in the interval $(0, 1)$, the separation between network input points is not uniform and increases as the distance from 0 or 1 decreases.

To improve the network interpolation near the boundary points and improve the distribution of network input points, a modified mapping $\phi_m(x)$ is introduced. This mapping assumes that the input interval is slightly larger than (0,1). A linear mapping is performed on this larger interval onto (0,1) and the output of this linear mapping is processed through $\phi(x)$. Let $(-\Delta, 1+\Delta) \supset (0,1)$. The mapping $g(x) = \frac{1}{1+2\Delta}(x + \Delta)$ maps the interval $(-\Delta, 1+\Delta)$ onto $(0,1)$. Then, the modified mapping $\phi_m(x)$ is given by $\phi_m(x) = \log(\frac{g(x)}{1-g(x)})$.

However, the use of the modified mapping $\phi_m(x)$ shrinks the size of the input interval to the network. This reduction in the size of the input interval reduces the value of h when computed using (6). We introduce a multiplying factor $k > 1$, where k is chosen such that the value of h is not too small. The modified mapping $\phi_m(x)$ will have the form $\phi_m(x) = k \log(\frac{g(x)}{1-g(x)})$.

Using very large or very small values of h should be avoided. High values of h lead to "flat" *sinc* units that fail to capture details in the input space (and the learning is usually very slow for such values of h) while low values of h lead to sharp *sinc* units with increased possibility of learning instability.

3.5.1 Function with Singularities at End Points

The *sinc* network was trained using the data set described in section 3.2 but using the modified mapping $\phi_m(x)$. Several attempts were made to train the network using different values of k while keeping the number of hidden units fixed at 11. In all trials, the network was unable to learn the training set to any acceptable degree of accuracy. Better learning was achieved using more hidden units but, for comparison purposes, those results are irrelevant. In most situations, the network had the hardest time learning points where the slope of the function changes suddenly from an almost infinte slope to almost zero. In this case, the original mapping $\phi(x)$ was superior to the modified one $\phi_m(x)$.

3.5.2 Function with Oscillations

The *sinc* network using the modified mapping $\phi_m(x)$ was trained using the data set described in section 3.4 with the same number of hidden units (31 units). For the modified mapping $\phi_m(x)$, k was set to 4. The network using the modified input data showed significant improvement in learning over the one using the original $\phi(x)$ as can be seen from Figure 14.

Both networks were then tested for generalization using 1000 points equally distributed in the interval $[0.001, 0.999]$. The results of such tests are depicted in Figure 15. As can be seen from the figure, the network using the unmodified mapping generated a fitting curve that captured the basic trends in the input data but failed to learn the small details (oscillations) therein. Furthermore, it is noted that the output had large oscillations near both end points. On the other hand, the network using the mapping $\phi_m(x)$ was able to reproduce the learned mapping with greater accuracy while eliminating the oscillations near the end points observed in the other net.

4 CONCLUSION

The *sinc* network was used to approximate functions defined over the interval $(0,1)$ and the performance compared to that of the conventional sigmoidal network using the *tanh* activation function. Networks based

directly on the representational properties of the *sinc* network as well as slightly modified networks were discussed. In the functions used, the *sinc* network's performance was superior to that of the sigmoidal network in terms of the number of epochs required to achieve satisfactory learning. The actual time required to perform the same number of epochs in both networks was almost identical although the *sinc* network uses half as many weights as the sigmoidal net. This is due to the difference in computation time for the activation function and its derivative. A modified mapping function ϕ_m was introduced to overcome problems with network interpolation near endpoints.

This work could be extended to accomodate function representation in the complex domain since the representation theory of the *sinc* function applies to the complex domain as well. Extending the network repesentation to accomodate the general mapping $\mathbb{R}^m \rightarrow \mathbb{R}^n$ is worthy of study. Also using the *sinc* function in an RBF network in higher dimensions and evaluating its performance as opposed to the more traditional RBF functions (e.g. the Gaussian function) would involve both theoretical as well as practical issues. The versatility of the *sinc* function suggests its potential for use as the building block in an adaptive architecture network that changes its configuration based on some features of the combined input/output space.

5 REFERENCES

[1] T. Chen, H. Chen, and R-W. Liu. "Approximation capabilities in $C(\overline{\mathbf{R}}^n)$ by multilayer feedforward networks and related problems." *IEEE Trans. Neural Net.*, 6(1):25–30, January 1995.

[2] G. Cybenko. "Approximation by superposition of sigmoidal function." *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[3] W. R. Elwasif. *Function approximation using a sinc neural network.* Master thesis, Florida Institute of Technology, Melbourne, Florida, August 1995.

[4] L. V. Fausett. *Fundamentals of neural networks.* Prentice-Hall, 1994.

[5] V. Kůrková. "Kolmogorov's theorem and multilayer neural networks." *Neural Networks*, 5:501–506, 1992.

[6] A. N. Kolmogorov. "On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition." *Doklady Akademii Nauk, USSR*, 114(5):953–956, 1957.

[7] V. Kreinovich. "Arbitrary nonlinearity is sufficient to represent all functions by neural networks : A theorem." *Neural Networks*, 4:381–383, 1991.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning internal representation by error propagation." In David E. Rumelhart, James L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, volume 1, chapter 8. MIT Press, 1986.

[9] D. A. Sprecher. "On the structure of continuous functions of several variables." *Transactions of The American Mathematical Society*, 115(3):340–355, 1965.

[10] D. A. Sprecher. "A universal mapping for kolmogorov's superposition theorem." *Neural Networks*, 6:1089–1094, 1993.

[11] F. Stenger. "Numerical methods based on Whittaker cardinal, or sinc functions." *Siam Review*, 23(2):165–224, April 1981.

[12] E. T. Whittaker. "On the functions which are represented by the expansions of the interpolation theory." In *Proc. Roy. Soc. Edinburgh*, volume 35, pages 181–194, 1915.

[13] J. M. Whittaker. "On the cardinal function of interpolation theory." In *Proc. Edinburgh Math. Soc.*, volume 2 of *1*, pages 41–46, 1927.

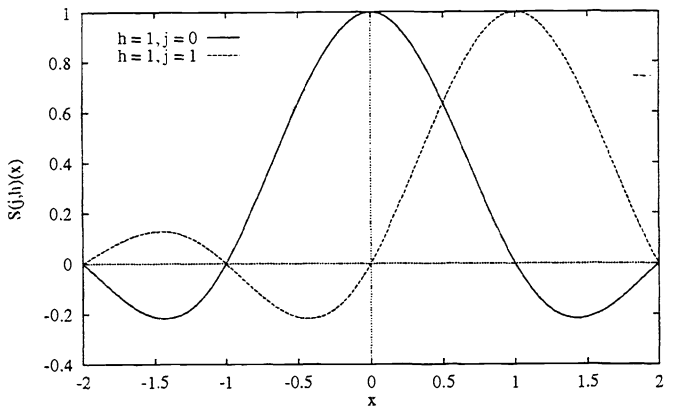

Figure 1: *Effect of Parameter j on the Function $S(j,h)(x)$*

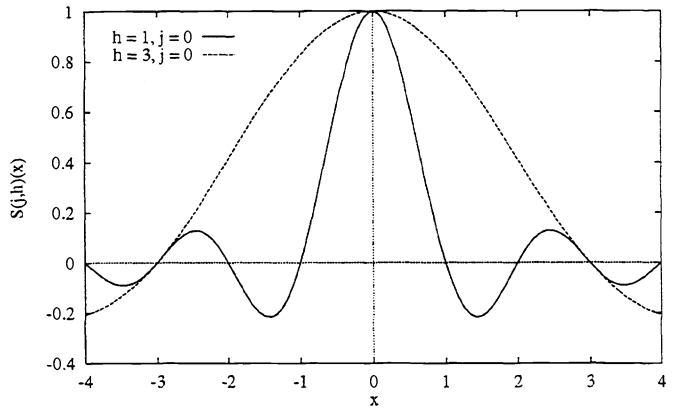

Figure 2: *Effect of Parameter h on the Function $S(j,h)(x)$*

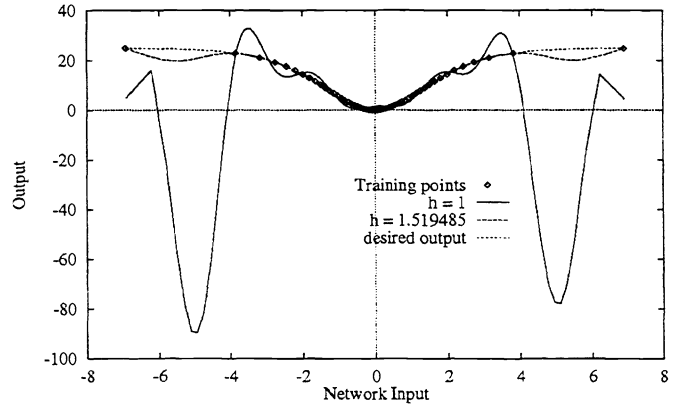

Figure 3: *Network Performance on $f(x) = (10(x - 0.5))^2$*

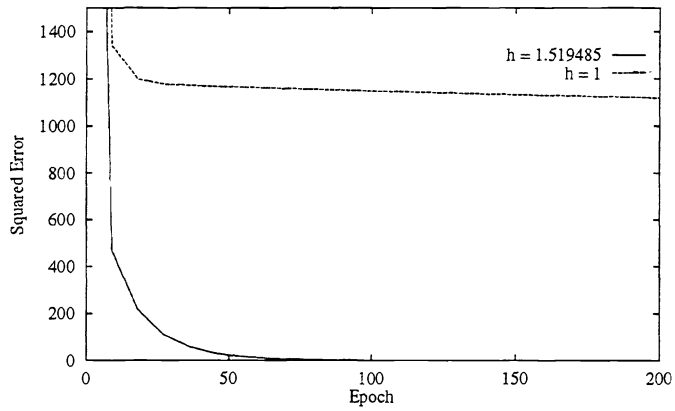

Figure 4: *Training Error for* $f(x) = (10(x - 0.5))^2$

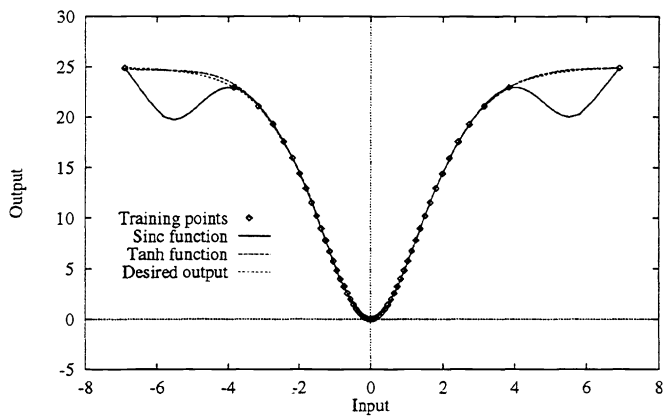

Figure 5: *Tanh vs Sinc Network for* $f(x) = (10(x - 0.5))^2$

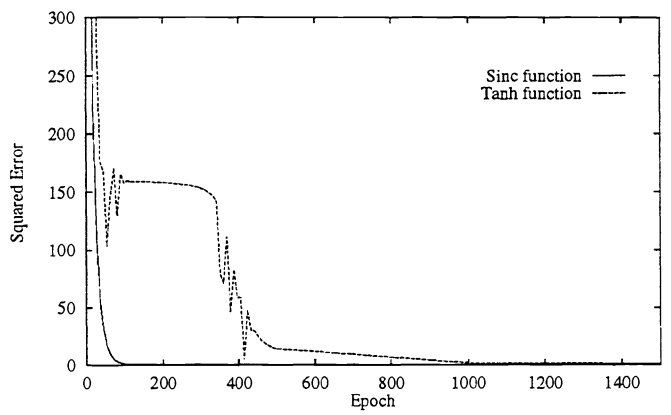

Figure 6: *Error for Tanh and Sinc Networks for* $f(x) = (10(x - 0.5))^2$

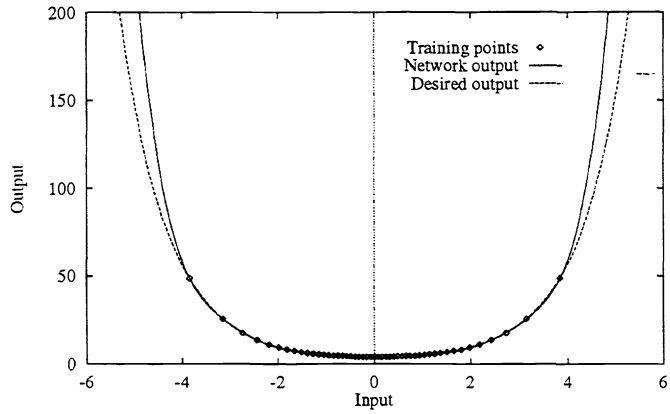

Figure 7: *Sinc Network Performance for Function with Singularities*

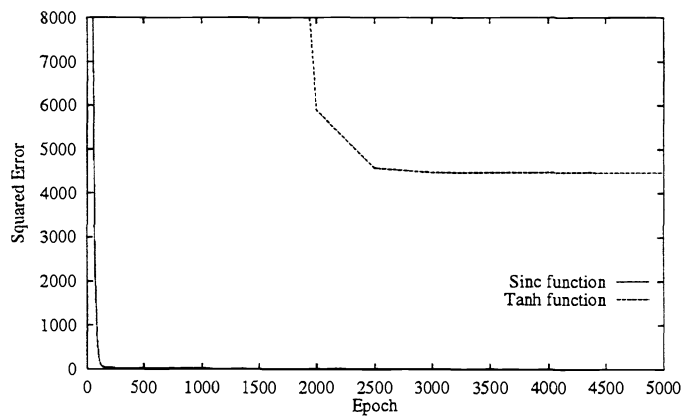

Figure 8: *Error Comparison for Function with Singularities*

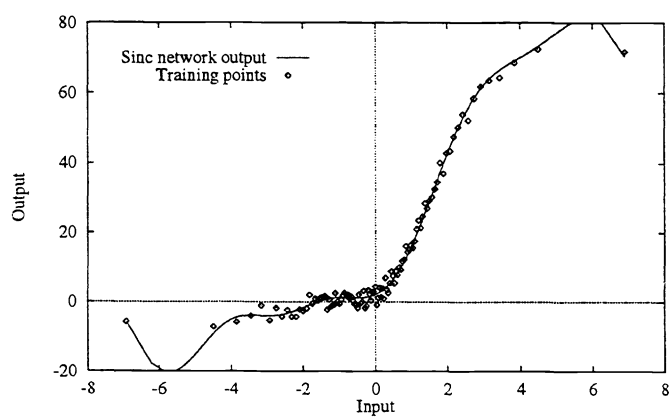

Figure 9: *Sinc Network Performance on Noisy Training Data*

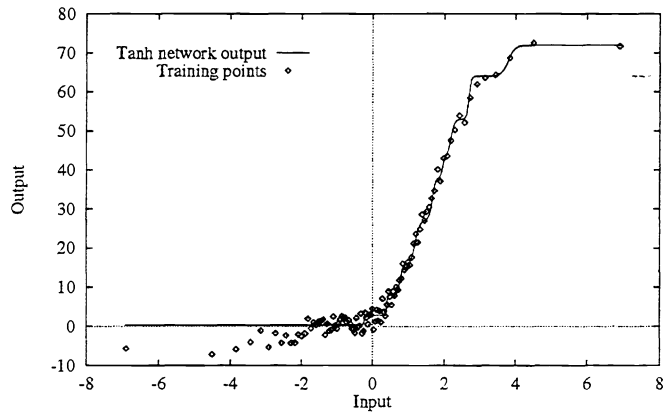

Figure 10: *Tanh Network Performance on Noisy Training Data*

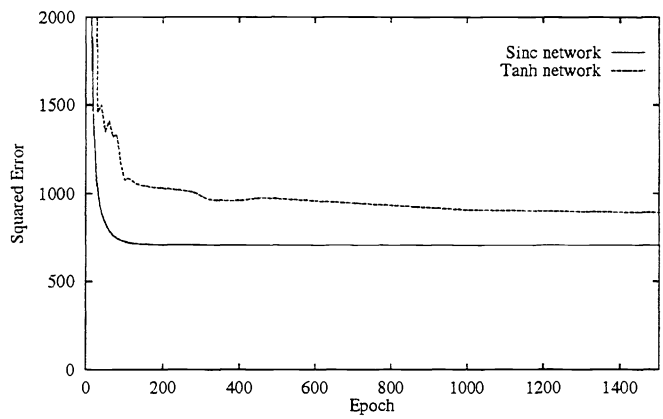

Figure 11: *Error Comparison for Function with Added Noise*

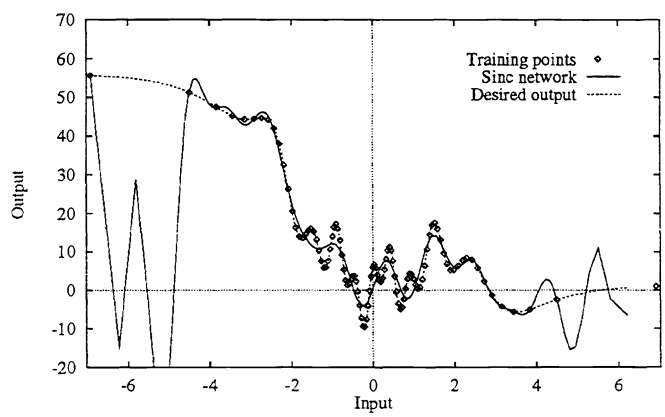

Figure 12: *Sinc Network Performance on an Oscillatory Function*

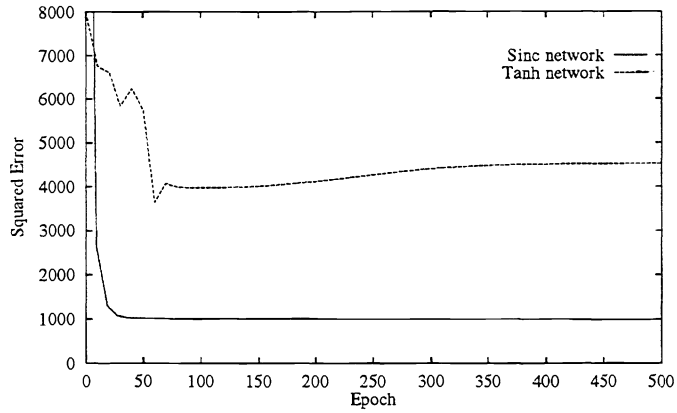

Figure 13: *Error Comparison for Oscillatory Function*

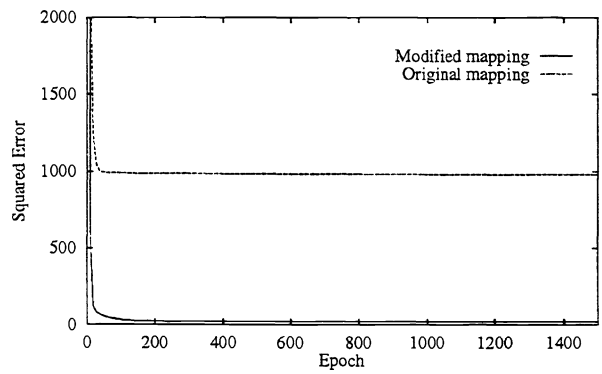

Figure 14: *Learning Error Using Modified Mapping $\phi_m(x)$ for Function with Oscillations*

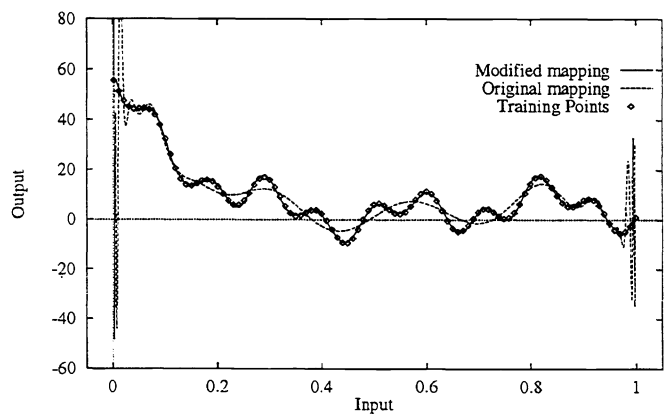

Figure 15: *Network Generalization Performance using the Modified Mapping $\phi_m(x)$ for Function with Oscillations*