Using the Generalization Properties of a Neural Network to infer the Number and Shape of its Hidden Units

Terence D. Sanger
Jet Propulsion Laboratory
MS 303-310
4800 Oak Grove Drive
Pasadena, CA 91109
(818) 354-9127 tds@ai.mit.edu

Abstract

The generalization ability of a neural network is a reflection of the underlying computational structure, and in multi-layer networks it depends upon the computations performed by the intermediate "hidden" units. In this paper, I show how the generalization pattern of a two-layer network can be used to infer the structure of a set of fixed hidden units, up to linear combination. The ability of the network to learn is crucial to this technique, since it will be trained on many different sets of desired outputs. Each training set yields a distinct pattern of generalization, and these patterns can be combined across trials to yield a model of the hidden units. This technique can be used to construct a network model of an unknown biological process. It can also be used "in reverse" to create a network with desired generalization properties. Two simulated examples are shown to demonstrate the method.

Category: Theory: computational learning theory
Keywords: Network structure, Network analysis, Generalization, Hidden units, Basis function approximation
Preference: Oral presentation

1 INTRODUCTION

In modelling an unknown biological system, we frequently ask whether there is a set of underlying intermediate "features" which are used in computation. The importance of such features is that they will determine the behavior of the overall system. We thus can write

$$y = \sum a_i \psi_i(x) \tag{1}$$

where x is the input, y is the output, $\psi_i(x)$ is a fixed (nonlinear) "feature" whose value depends on the input, and a_i is a weight connecting that feature to the output. If the system is adaptive so that the a_i's respond to training, then the features ψ_i will determine how it generalizes to new inputs after being trained on a different set of inputs. If we can discover the form of the features, then we can predict the response to training and the class of functions that can be learned accurately.

It is important to realize that the actual unknown system may not, in fact, have the form (1). Nevertheless, this form is sufficiently general that it can be used as a model, and its computational properties can be used to make testable predictions about the unknown system's behavior. The importance of the class of models (1) is that gradient descent training incurs no local minima, and we can easily use the similarity to well-known models such as the Fourier or Taylor series to access a large body of theoretical results.

2 THEORY

Let N be an unknown adaptive network which maps inputs x onto scalar outputs $y = Nx$. Assume that N can be represented as a superposition of n features:

$$y = Nx = \sum_{i=1}^{n} a_i \psi_i(x)$$

where each a_i is an adaptive scalar weight. Assume further that there exists a large class of basis functions $\phi_j(x)$ such that the features in turn are linear combinations over this basis:

$$\psi_i(x) = \sum_j s_{ij} \phi_j(x) \tag{2}$$

where s_{ij} is a set of (non-adaptive) constants describing the unknown system. It is not difficult to find such a basis ϕ_j, and we can use a "standard" one such as the Fourier, Taylor, or delta-function basis.

Let X be a matrix of training values of x, and X' be a matrix of test values (possibly such that $X \subset X'$). In matrix notation we can write:

$$Y = NX = a^T \Psi X = a^T S \Phi X$$

where Y is a row vector of output values, a^T is a row vector of weights a_i, ΨX is a matrix of hidden node values for each example $[\psi_i(x_p)]$, S is the matrix $[s_{ij}]$, and ΦX is the matrix of values $[\phi_i(x_p)]$. From equation (2), the hidden unit outputs are given by the set $S\Phi X$.

Let Y_k be the vector of desired output values for each $x \in X$ for the k^{th} learning trial. At each trial, we will let the network N adapt a_k^T so that its output approximates the desired value in Y_k for each input in X. (The set of input points X is fixed for all training trials.) After training, we then test the network generalization on a new set of inputs X' and compute the corresponding outputs Y_k'. We then use Y_k' to train the weights of a larger network

$$Y_k' \approx b_k^T \Phi X$$

over the full set of basis functions ϕ. For each training trial, the weights b_k^T represent the complete behavior of the network N on both the training and test sets. Over many trials, we can analyze the set of b_k^T's and discover the form of the hidden units in terms of the basis functions ϕ_j.

To summarize:

Step 1: *Adapt* N to minimize $\|Y_k - NX\| = \|Y_k - a_k^T S \Phi X\|$

Step 2: *Generalize* and compute $Y_k' = NX' = a_k^T S \Phi X'$

Step 3: *Train* a larger net to minimize $\|Y_k' - b_k^T \Phi X'\|$

Step 4: *Repeat* steps 1-3 for multiple trials k

Now we can write $b_k^T = a_k^T S$, and the set of all vectors b_k^T lies within the plane spanned by the rows of S which determine the hidden units. If we have chosen enough different (linearly independent) values of Y_k, then the set of vectors b_k^T will span the same plane as the rows of S.

Many of these vectors will be linearly dependent, and we can eliminate this dependence by performing Gram-Schmidt orthonormalization. If we write these vectors as a matrix B, then after orthonormalization at most the first n rows will be nonzero (where n is the number of distinct features ψ). So we can determine the number of hidden units n directly. The first n rows will be a random linear combination of the rows of S, and we can reconstruct a set of basis functions $B\Phi$ which will be a linear combination of the desired basis functions Ψ. Note that since the computational properties of the original network N are unchanged by forming invertible linear combinations of the hidden units, we will not be able to distinguish which combination is the "correct" one, since this is not a well-defined concept. Continuing the summary:

Step 5: *Orthogonalize* the matrix B with rows given by b_k^T

Step 6: *Solve* by determining the number of hidden units to be the number of nonzero rows of B (after orthogonalization) and the form of the hidden units to be a linear combination of the orthogonalized features $b_k^T \Phi$.

Note that the number of trials should be greater than or equal to the number of hidden units n, and that the desired output values Y_k should be linearly independent. Since n is unknown, we must use a number of trials believed to be larger than n.

The technique rests upon the idea of choosing a large number of easily-computed basis functions (such as a Fourier series) and then noticing that any network using a

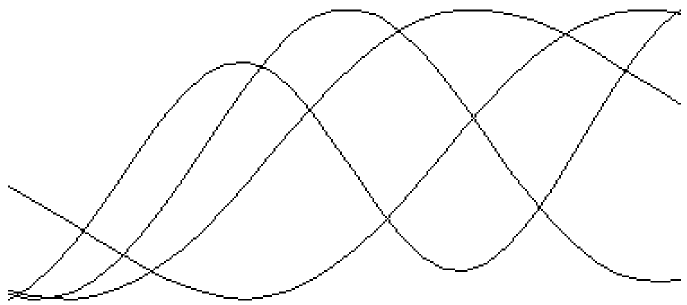

Figure 1: Linearly independent hidden units discovered for the first example.

small number of hidden units is computationally limited in a way that corresponds to a projection of the weights onto a low-dimensional subspace. Repeated training trials allow the properties of this projection to be explored, and the underlying hidden units can be analyzed (up to linear combination).

3 EXAMPLES

3.1 Smooth Hidden Units

Consider a network N that takes scalar input x in the range $[0, 1]$ and has 4 Gaussian hidden units

$$\psi_i(x) = \exp(-(x - m_i)^2/2\sigma^2)$$

where $\sigma = 0.25$ and $m_i = 0.25 \times i$. I will show the application to this example as if it were an unknown network.

Choose 32 different values of the desired inputs spread evenly over $[0, 1]$, and for each of 10 trials choose the desired scalar output values randomly for each input (in the range $[-1, 1]$). Then for **Step 1**, use the LMS rule to adapt the weights a_k^T to convergence with a rate of 0.1 and 1500 examples. Since the network only has 4 basis functions but 32 possible desired output values, the approximation always has significant error. For **Step 2** the actual outputs of the network Y_k' are then computed for each of the 32 input values.

For the "larger" basis network I use the Fourier basis, so that $\phi_i(x) = \cos(\omega_i x + \delta_i)$. This basis has the advantage that training can be accomplished using the Fast Fourier Transform. Thus in **Step 3**, b_k^T is computed from the FFT of the true output values Y_k'. Steps 1 to 3 are then repeated for each of the 10 trials.

For **Step 5**, after the sets of Fourier coefficient vectors b_k^T are orthonormalized using a sequential Gram-Schmidt procedure, only the first four are nonzero. These are then inverse-transformed to find the form of the hidden units, which is graphed in figure 1. Note that these hidden units are linear combinations of the original four Gaussian units. Also note that the form of the hidden units indicates that

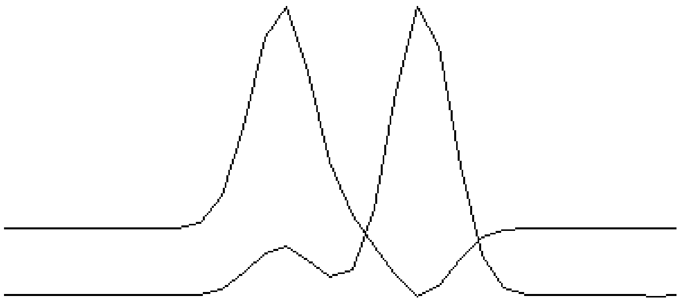

Figure 2: Hidden units for the second example using a local training set.

only smooth functions can be approximated, and this gives a clear picture of the function of the network.

3.2 Local Testing

Consider the case in which the training set X and the test set X' are different, and the points in the training set are very close together. This would happen if we were interested in studying the long-range generalization properties of the network. This case becomes particularly interesting if the hidden units are themselves local, so that a local training set will excite only a small subset of the hidden units. If this is the case, we may be able to find the form of the hidden units directly, and not just a linear combination.

In the extreme case, consider taking the training set to consist of a single point $x = X$. If this point excites only a single hidden unit, then the network will behave as if only one hidden unit existed, and the analysis technique above will find the "span" of that unit which will be the weight vector itself. Unfortunately, in general even a single input point will excite more than one hidden unit, since considerable overlap may exist. But suppose that the network incorporates some form of resource allocation method such as weight-decay. In this case, the hidden unit with the strongest initial response will "win", and weights from the others will decay to zero since only one hidden unit is needed to approximate any function at a single point. After training, we can still analyze the generalization properties at multiple points, and derive the form of the apparently single hidden unit. (An alternate resource allocation method is simply to start all the weights at 0, so that if a hidden unit is never excited it will never contribute to the output.)

For the second example, the "unknown" network now consists of 32 Gaussian hidden units with $\sigma = 1/32$. Instead of a single input point, two were chosen for illustrative purposes at values $x = 0.4$ and $x = 0.6$. A simple weight decay learning rule was used:

$$\Delta a = \gamma(y - a^T\Psi(x))\Psi(x) - 0.1\gamma a$$

with a rate of $\gamma = 0.1$ and 2000 examples. 5 complete training trials were performed. The coefficients b_k^T were calculated using a 32-point FFT, and after orthonormalization only two independent hidden units were found. These are shown in figure 2. These are not the exact hidden units, because there is a slight linear combination of the two. If only a single input point had been used, it would have found one of the hidden units exactly. The entire procedure can be repeated for different input values to extract the complete set of hidden units.

4 NETWORK DESIGN

An important application of this technique is in the design of artificial neural networks. We can reverse the analysis process and use the method to construct hidden units to have desired generalization properties. To do this, choose any set of sample responses Y_k which the network should have, and encode them as a set of vectors b_k^T (using either the FFT or a learning algorithm). Then orthonormalize these vectors to yield a set of n vectors s_i^T which can be grouped into the rows of an orthogonal matrix S. Now, note that $SS^T = I$, so that we can write $b_k^T = a_k^T S$ where $a_k^T = b_k^T S^T$. So if we let the hidden units be represented by $\phi_i(x) = s_i^T \Phi(x)$, we will obtained exactly the desired generalization behavior.

5 CONCLUSION

This apparently simple technique actually accomplishes a significant and important task. We can now model the internal structure of an unknown adaptive system by examining how it generalizes after training on a small set of examples. An immediate goal is to apply this to real neurophysiological and psychophysical data to predict the internal processing in primate motor and sensory systems. The technique can be expected to have wide applicability in the analysis of data drawn from adaptive systems. It can also be used to design adaptive systems with desired generalization properties.