

Researches into Neural Network Architecture

James J. Ewell III

University Undergraduate Fellow, 1989-90

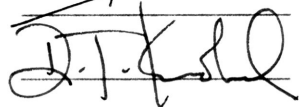
Texas A&M University

Department of Electrical Engineering

APPROVED

Fellows Advisor

Honors Director

~~AD/AD/AD~~


ABSTRACT

New insight into the choice of the number of hidden nodes for three layered neural networks is presented. I have conceived a method to reduce the total complexity and size of a network by testing for linear dependence of nodal relationships. Limitations imposed by choice of the training algorithm and training set is introduced. A discussion of training and its effects on neural net performance completes the discussion of neural networking. These evaluation procedures are very important for the physical construction of neural networks.

I. Introduction

Neural nets are linear combinations of nodes which have nodal output related to the weighted sum of the nodal input. Conglomerations of these nodes can recognize complex linear rules from multi-dimensional input patterns and apply these rules to input patterns which have not been explicitly taught to the network. In general, non-linear patterns, rules, and functions cannot be implemented by neural nets.

The original impetus for development of neural nets came from the known biological structure of human brain neurons. Simplistic models of neurons were devised and eventually refined for use as the nodes of a neural network. These nodes have a response to their input called an activation function. This

function is arbitrary. Generally, increasing the total nodal size of the network while keeping the number of layers constant (and greater than two) allows a neural network to map more complex data; however, additional size and consequent complexity can overpower a working solution and impair network performance.

In the 1960s, M. Minsky and S. Papert of MIT¹ thoroughly analyzed the neural networks of the time, particularly the two layered (input and output) network of perceptrons invented by Frank Rosenblatt. These networks were simple systems and their analysis demonstrated that such two layered networks could only solve problems that were absolutely linearly separable. Absolutely linearly separable means each linear division of the input space perfectly slices the output into the same number of different areas. No two regions can have the same output on an absolutely linearly separable input space.

Restricting input to be absolutely linearly separable removes too many useful functions (exclusive-or for example) for this to be a tenable architecture. This problem has begun to be overcome by adding a third layer of additional complexity between the input and output layers. With the additional middle layer, the input space requirements are reduced to any combination of linearly separable regions which is an obvious improvement. Different linearly separated regions may have the same output (note: their regional definitions are still linear). This allows many problems of pattern matching to be implemented, including the exclusive-or paradigm.

Research in neural nets continues in many diverse areas. Obtaining poles and zeros of data sets by adaptive filters with

and without feedback, system modeling of observable input and output signals, statistical prediction to estimate future values of time correlated digital signals from present and past input data, noise cancellation in signal processing of medical equipment, adaptive echo cancellation in long distance telephone lines, and channel equalization for non-flat frequency response and non-linear phase response in telephone signal passbands are examples of modern problems which neural networking techniques hope to solve.

Rumelhart networks (R-nets)² constitute a fundamental configuration of neural networking. They predominately have three layers of nodes (figure 1): input, middle, and output with the number of nodes in the input and output layer constant and known for a particular application. Each node is connected to every node on the following layer. These connections may have differing strengths modeled by a linear multiplier called the weight of a particular connection. The complexity of the middle layer is the only mutable parameter and is normally chosen eclectically. This is obviously an inefficient process, and methods need to be devised to intelligently choose the degree of complexity of the middle layer and to pare any redundancy in existing networks.

II. Choosing the Complexity of the Middle Layer

In general, R-nets can be represented as a series of array processes upon an input data set (see Appendix A). These array processes are linear matrices of known dimension. The network output depends upon a weighted linear sum of the result of the middle layer's interpretation of the input (which is a weighted linear sum dependent upon the original set of input data). The individual weights connecting each node to every node of the preceding layer may vary widely, but, once established, remain constant for all inputs, allowing this matrix interpretation of the network activity.

Currently, methods of choosing the complexity (size) of the middle layer and training are sorely lacking. Several methods are discussed in this paper with reference to a three layered neural network where every node is connected by a differing set of constant weights to every node on the following layer. All nodes are assumed to have the same sigmoidal activation function.

In order to choose the complexity of the middle layer, a knowledge of the performance of these nodes must be made. A simple geometric proof³ gives the number of regions an arbitrary number of nodes can represent. In n-dimensional space, the maximum number of regions that are linearly separable using M nodes is given by:

$$M(H,n) = \sum_{k=0}^n \binom{H}{k}$$

where

$$\binom{H}{k} = 0 \text{ when } H < k.$$

The calculation of the number of nodes required to separate a linearly separable problem is perfunctory if an exact relationship for the n-dimensional input space is given. Simply find the minimum number of nodes to represent the requisite number of input spaces. Since the input and output layer are immutable, the middle layer will be the location of these nodes necessary to distinguish all areas on the input.

If the input space is not known in absolute detail, approximations may be made over the region of points known. These approximations (such as for sonar return data⁴) from groups of data points to input data regions exist, but are not well defined in the neural networking community.

The number of hidden layer nodes is a function only of the number of input spaces, and not of the training patterns. Proper training is dependent upon the architecture of the input space regions and has no real relationship to the choice of the number of middle layer nodes. This distinction needs to be made to prevent confusion of considerations in construction architecture, proper training, and final operation.

III. Removing Unnecessary Complexity in the Middle Layer

If a network exists, it would be useful to identify excess complexity of the middle layer. I will note here that a reduction in complexity will result in an improved design. Speed will not increase due to the massively parallel nature of the constructs, but total nodal size and complexity will decrease. Viewing the network's finite number of inputs as a n-dimensional space, one quickly assesses whether any of the array processes implemented by the neural network can be reduced. A reduction can be made if the array processes are linearly dependent upon each other.

The reasoning behind reduction is intuitive. If we are dealing with a weighted linear combination, combinations which duplicate function (by being non-orthogonal) may be removed. The overall network is benefited by the simplification. Again, the simplification comes in terms of complexity and total nodal count, not in terms of speed.

Since the network can be modeled by matrices (see Appendix A), we can write a number of equations equal to the number of middle layer nodes times the number of input nodes and analyze the result. We can also write a number of equations equal to the number of nodes on each layer and analyze them separately because they should all be linearly independent. This latter method will be used for simplicity.

If a system of n homogeneous equations in n unknowns has a non-trivial (that is not all zeroes) solution, then necessarily the determinant of the coefficient matrix is zero. The Gramm Determinant⁵ may be calculated for further reinforcement of the linear independence of the network's nodes.

Given p vectors $\mathbf{x}_v (v=1 \dots p)$ in an inner product space E , the Gram determinant $G(\mathbf{x}_1 \dots \mathbf{x}_p)$ is defined by:

$$G(\mathbf{x}_1 \dots \mathbf{x}_p) = \det \begin{bmatrix} (\mathbf{x}_1, \mathbf{x}_1) & \dots & (\mathbf{x}_1, \mathbf{x}_p) \\ \vdots & & \vdots \\ (\mathbf{x}_p, \mathbf{x}_1) & \dots & (\mathbf{x}_p, \mathbf{x}_p) \end{bmatrix}.$$

where the parenthetical operator is the inner measure. It will be shown that

$$G(\mathbf{x}_1 \dots \mathbf{x}_p) \geq 0$$

and that equality holds if and only if the vectors $(\mathbf{x}_1 \dots \mathbf{x}_p)$ are linearly dependent.

To prove this assertion, assume first that the vectors $\mathbf{x}_v (v=1 \dots p)$ are linearly dependent. Then the rows of the matrix are also linearly dependent whence

$$G(\mathbf{x}_1 \dots \mathbf{x}_p) = 0.$$

If the vectors $\mathbf{x}_v (v=1 \dots p)$ are linearly independent, they generate a p -dimensional subspace E_1 of E . E_1 is an inner product space. Denote by del_1 a normed determinant function in E_1 . Then it follows

$$G(\mathbf{x}_1 \dots \mathbf{x}_p) = \text{del}_1(\mathbf{x}_1 \dots \mathbf{x}_p)^2.$$

The linear independence of the vectors $\mathbf{x}_v (v=1 \dots p)$ implies that $\text{del}_1(\mathbf{x}_1 \dots \mathbf{x}_p)$ is not 0, whence

$$G(\mathbf{x}_1 \dots \mathbf{x}_p) > 0.$$

Once a determination of linear dependence has been made, the linearly dependent terms can be excised from the net. These dependent terms manifest themselves during triangularization. They are rows (nodes) with zeros along the diagonal or constants under the triangularization line after triangularization. Each

row represents a specific node's equation. Error in triangularization for a particular row (node) discloses the dependence of that row (node) upon another row (node). In order to complete triangularization to find these interdependencies, remaining constants below the triangularization line should be chosen instead of zeros along the diagonal. Removal of a node will usually necessitate a recalculation of the Gramm determinant and a new triangularization of the Gramm determinant.

Due to the linear nature of the weights, replacement of a single pair of dependent nodes may be done analytically. Both of the dependent nodes are removed and a new node is added. Each weight of the new node to the preceding and following layer is the sum of the dropped nodes weights to the preceding and following layers:

$$w_{\text{new}} = w_{\text{I,old}} + w_{\text{II,old}}$$

This will not necessarily force a retraining of the network. If more than one dependency exists in a particular set of nodes, it will be necessary to recalculate the triangularization before an additional substitution may be made.

There is a need for a slight redundancy of about ten percent of the total middle layer size in which to train. This is necessary due to the imperfect nature of the training algorithm (gradient descent). For small networks (number of nodes less than eight or ten) at least one node is usually required and for very large networks (number of nodes greater than fifty or sixty) the additional nodal requirement becomes relatively static at four or five additional nodes (see Figure 2).

IV. Training Methods and Their Importance

Training is second in importance only to the choice of network structure. Poor training algorithms or training data will make even a well designed network ineffective. Deficient training data can leave blank spaces in the network output and inferior training algorithms can make the training inordinately lengthy.

During training, the internodal weights are altered according to a training algorithm⁶ (e.g. backpropagation as shown in Appendix B). The ability to optimize the network's weighting is dependent upon the particular methodology, but certain absolutes do exist.

If the network does not receive a proper training data set, the output may have regions of untrained space where the network has never had a known output with which to compare. These areas can be dangerous due to their unpredictability. A set of proper training data incorporates at least one training example for each region on the input space that the network is supposed to recognize and also one training example for every linearly separable area that the network would not normally receive an input upon. This last addition protects against the possibility that a garbled or otherwise spurious input may produce apparently valid output.

As an example, simple logic gate neural network models have these blank spaces if the training data does not include supposedly invalid input. For logic gate operation, the input is binary and the neural network approximation to this falls within

an error bound, E , of one or zero (for the sigmoid nodal activation function). If the network is supplied an input in the forbidden range, $0+E$ to $1-E$, a valid output may result. This is obviously not desirable because the input had no meaning for the logic gate model (fuzzy logic models are not considered).

Methods for allocating additional training patterns to eliminate valid output from undesirable input have not been presented up to now. Quickly, they must satisfy two broad conditions. They must not overlap with areas of valid input (linear separability) and they must provide training in a region normally not associated with valid output (perhaps a particular region for error control in the normed output space or a region outside of the normed output space for error detection alone).

Considerations for choosing the location of error control (error trapping) and methods of the mathematical definition of any region need to be developed. Considerable difficulty arises here due to the possible inability of the network to distinguish between an input generating valid output and an invalid input pattern generating "good" output. This follows usually from an improper training algorithm or improper training set, but there are limitations imposed upon all neural networks.

The network's inability to guarantee that each combination on the input space will be mapped into an exact and unique point on the output space also raises problems. Because of this limitation, there is no way of analytically extrapolating the values of the inputs from a known output for at least one input value (e.g. zero input versus zero weighted sum) and possibly more. Indeterminate mapping can limit reconstruction of the

network as a single multiple variable function and restricts our ability to gain analytical insight into the network.

A large number of training patterns increases the training time and is undesirable. Allowing one pattern for each definable region boundary on the input (including error trapping regions) compensates for the needed definition on the input space and keeps the total training size fairly low for most applications. Giving more than one training example for each region is not absolutely necessary as the information content of the additional training patterns is redundant to other training information.

In some cases, a reduction of the broadest training set is possible. Certain contiguous areas of the input space with equivalent outputs may not need to be explicitly noted as long as all input region boundaries are specified in some way by the training set. This reduction of the training set by analysis of the input space regions is easily realized, particularly when, the input space is well defined. Very little has been published on analyzing the input space, but common sense and simple analytic tools will suffice to analyze these multi-dimensional linear regions in many instances.

V. Conclusions

Neural networks are useful models for linearly separable pattern matching problems. Their construction and optimization is not complex and follows mainly intuitive rules. The number of linearly separable regions on the input space dictates the choice of training data and R-net middle layer complexity. Network nodes and training examples are needed to cover all regions of input including any regions for error trapping. The final network may need only one node for groups of contiguous input regions and a method to test for simplification was discussed. Recommendations to not trim a network to a minimum level of complexity came from the non-ideal considerations of the backpropagation training algorithms. Overall, this paper has covered construction, training, and optimization of general Rumelhart nets (R-nets).

Appendix A --- Matrix equations for R-nets.

Suppose R^K is the finite dimensional input space, R^L is the finite dimensional middle layer space, and R^M is the finite dimensional output layer space. Suppose x_K belongs to space R^K , y_L belongs to space R^L , and $y_L = {}^L[C]_K * x_K$. ${}^L[C]_K$ is the mapping from R^K to R^L . That is each row of $[C]$ is the set of weight values connecting the K inputs to a middle layer node.

The expansion for ${}^L[C]_K$ is:

$${}^L[C]_K = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1K} \\ w_{L1} & w_{L2} & \dots & w_{LK} \end{bmatrix}$$

where there are K input nodes, L middle layer nodes, and w_{KL} is the weight between them respectively.

Suppose $g_M = {}^M[P]_L * y_L$ where ${}^M[P]_L$ is the mapping from R^L to R^M . That is each row of $[P]$ is the set of weight values connecting the L middle layer inputs to an output layer node.

The expansion for ${}^M[P]_L$ is:

$${}^M[P]_L = \begin{bmatrix} p_{11} & p_{12} & \dots & p_{1L} \\ p_{M1} & p_{M2} & \dots & p_{ML} \end{bmatrix}$$

The mapping from input space to output space is, by substitution:

$$g_M = {}^M[P]_L * {}^L[C]_K * x_K = {}^M[T]_K * x_K$$

where ${}^M[T]_K = {}^M[P]_L * {}^L[C]_K$.

A number of conclusions may be drawn from the matrix representation of the input to output mapping. A zero row in $[T]$ implies the output node is always zero and independent of all possible input vectors. This is intuitive since the value of that node is always zero in the output. A zero column in $[P]$ shows that the corresponding mid-layer node contributes nothing toward any output node value (the network solution) and could be eliminated. Similar arguments hold for $[C]$.

Appendix B --- Backpropagation Networks.

Backpropagation networks are usually hierarchical; that is, they consist of at least three layers of nodes. An input layer, middle layer, and output layer exist with each layer fully connected to the next layer. Before beginning training, all nodal interconnection weights undergo a normed randomization.

During training, the network is repeatedly presented with a set of input patterns and a set of correct output patterns from a previously identified static training set. For each input pattern of the training set, the internodal weights are adjusted according to a given teaching algorithm. Eventually, if the teaching algorithm is a good algorithm, the network's performance will correctly generate the desired output.

The summed nodal input, I , is determined by multiplying each input signal by the weight on that interconnection:

$$I = f\left(\sum_{i=1}^n w_i * x_i\right)$$

where the sum is taken over all nodes in the previous layer. The terms w and x are the weight of interconnection and the magnitude of a particular input signal, respectively.

The function $f(x)$ is called the activation function of the node. It determines the excitation level generated as a result of the input signal. For a backpropagation network, this function is taken to be a sigmoid. Any continuous and monotonically increasing function asymptotically approaching fixed values as the input approaches plus or minus infinity may be used, but a sigmoid is the standard.

The general sigmoid function is:

$$f(x_i) = 1/[1+e^{-(x_i+T)}]$$

where T is a constant threshold shifter, x_i is a single nodal input, and e is the mathematical exponential constant. To make a network even more understandable, the threshold is usually set to zero (figure A1).

This function generates the node's output as a function of the summed input calculated as I . This output is propagated to every connection a particular node has with the next layer of the neural net (usually all following layer nodes).

The output is not binary since x would have to be infinite for a 0 or a +1 output. Since the weights must be able to be positive or negative, an output of zero will be termed negative output and an output of one will be termed positive output for binary problem discussion. Due to the infinite amount of input strength needed to reach these values; they will be arbitrarily changed to two regions: greater than $1-E$ for positive and less than E for negative. E is generally referred to as the output error bound.

Once the nodal outputs are all defined by the functions above, input signals may be applied. A comparison may be made between the network output generated from the input signal and the desired output. The nodal weights are changed according to a gradient descent technique using the desired output, actual output, previous iteration change, and current weighting:

$$W_{new} = W_{old} + \beta * E * X / (|X|^2) + a * (W_{new} - W_{old})_{prev}$$

where W_{new} is the new value for the weight, W_{old} is the old value for the weight, β is an arbitrary constant whose increasing value

speeds the arrival at a solution while making it more unlikely to stabilize (usually between 0 and 1), E is the error of the nodal output to the desired output, X is the input, $|X|$ is the magnitude of the input, and a is a constant for weighting the momentum term to allow the algorithm to escape from local minima on the route to optimization.

On the output layer, the weight changes can be readily calculated because the desirable output is provided by the training set. To assign blame on the middle layer, backpropagate the errors for each output layer node to the middle layer using the same interconnections and weights as the middle layer used to transmit outputs to the output layer. Compute the error term for each node in the middle layer based on their portion of the blame for the output layer's error. This is computed as:

$$e_i = f'(I) * \left(\sum_{j=1}^n w_{ij} * E_j \right)$$

where e_i is the error in the i th middle layer node and the sum is taken over j , where j indicates the j th output layer node. The remaining term is the derivative of the activation function of the middle layer node for the net input it received.

It is relatively easy to show that:

$$f'(I) = f(I) * [1-f(I)]$$

Applying this derivative serves two purposes. First, it contributes to the stability of the network since it ensures that, as the outputs approach 0 and 1, only very small changes can occur. Second, it helps compensate for excessive blame attached to a middle layer node. When the connection between a

middle layer node and an output node is very strong (extreme values for the weight on the interconnection) and the output node has a very large error, the weights of a middle layer node may be assigned a very large error also, even if that node had a very small output and thus could not have contributed much to the output node's error. By applying the derivative of the signal function, this error is moderated, and only small to moderate changes are made to the middle layer node's weights.

The weights are not actually changed until after the error has been propagated back to the previous layer. Once the activation has flowed forward through the network and the error has flowed backward through the network, a single iteration of a pattern in the training is complete and the next pattern in the training set is administered.

BIBLIOGRAPHY

- [1] M. Minsky and S. Papert, *Perceptrons*, MIT Press, reissued 1987.
- [2] J. L. McClelland and D. E. Rumelhart, *Explorations in Parallel Distributed Processing*, Cambridge, MA: MIT, 1988.
- [3] G. Mirchandani and W. Cao, "On Hidden Nodes for Neural Nets," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 5, pp. 661-664, May, 1988.
- [4] R. P. Gorman and T. J. Sejnowski, "Analysis of hidden units in a layered network trained to classify sonar targets," *Neural Networks*, vol. 1, pp. 75-89, 1988.
- [5] W. H. Greub, *Linear Algebra, Die Grundlehren Der Mathematischen Wissenschaften In Einzeldarstellungen*, vol. 97, pp. 186-196. Springer-Verlag New York, 1967.
- [6] M. Caudill, "Neural Networks Primer III," *AI Expert*, pp. 53-59, June, 1988.

Figure One
Three Layer Rumelhart Network
(Typical Backpropagation Network)

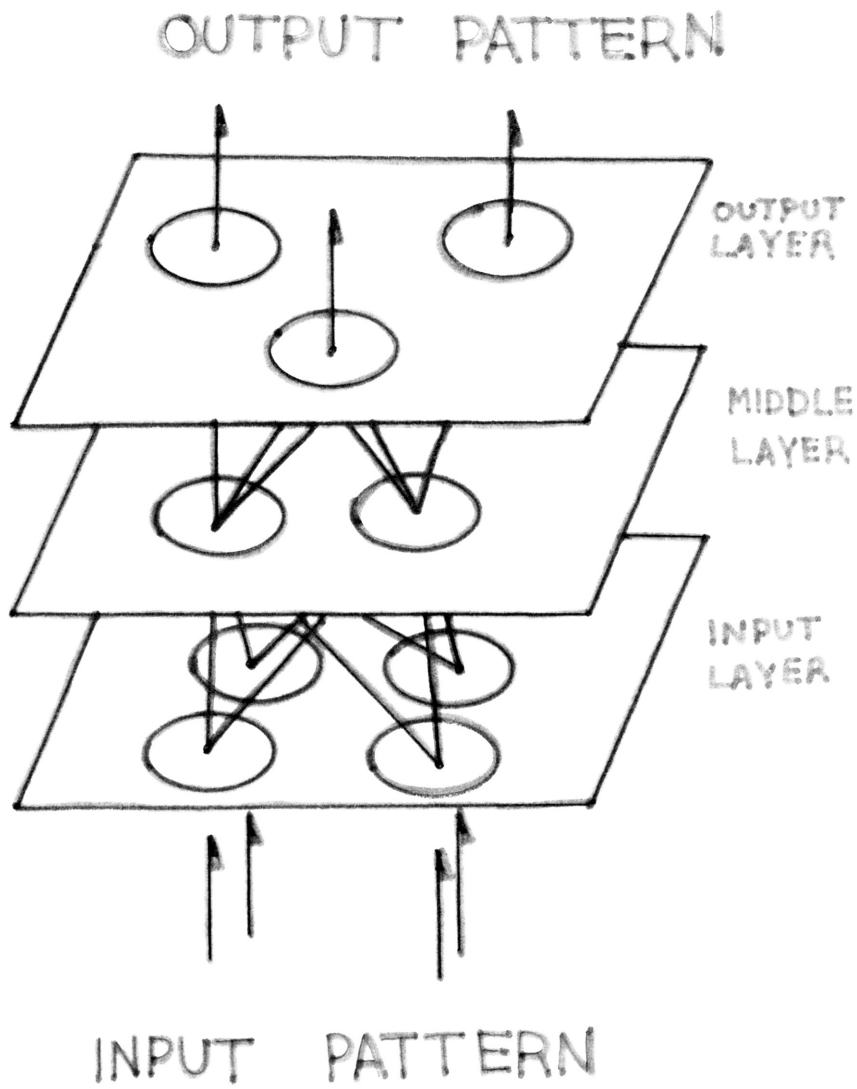
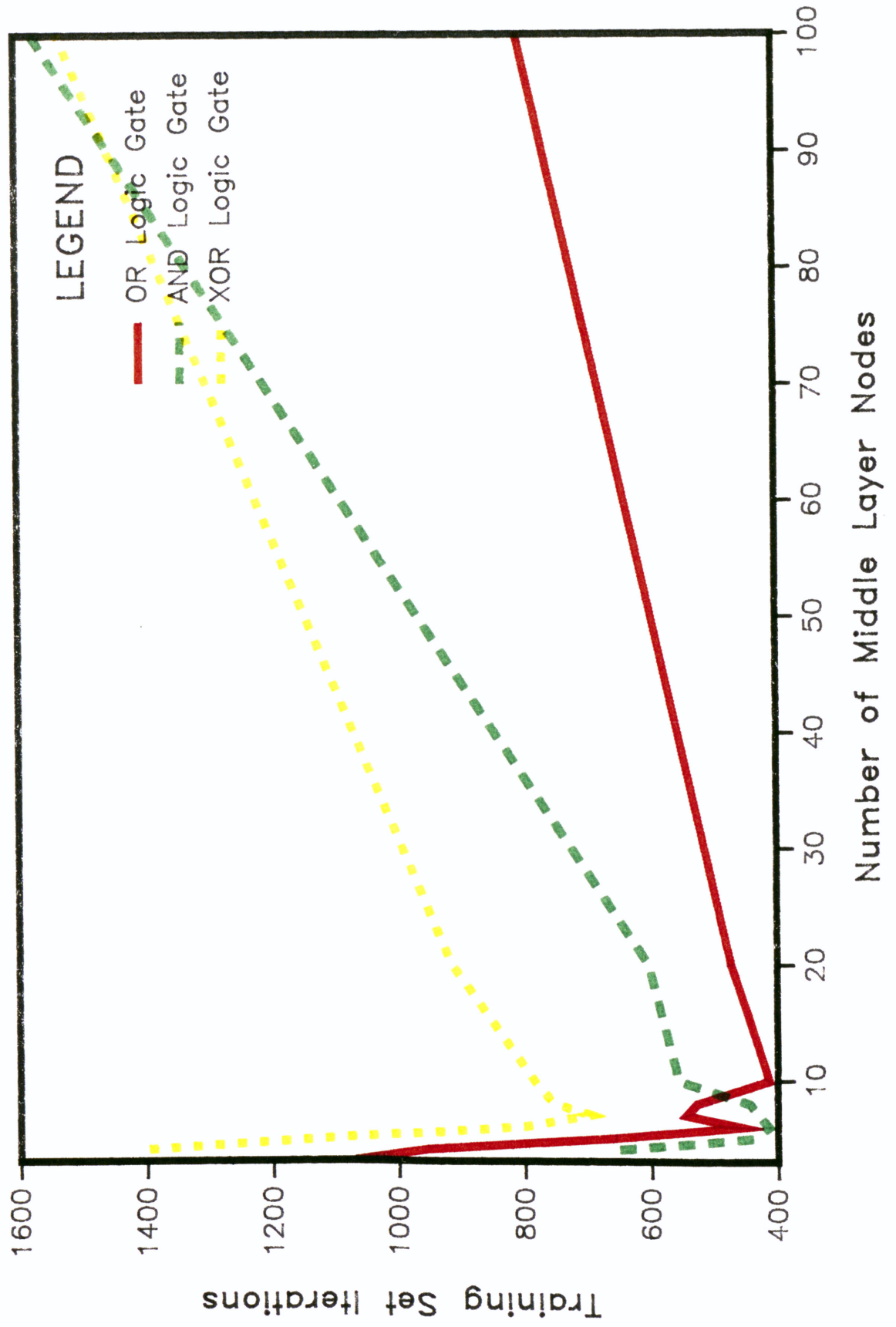
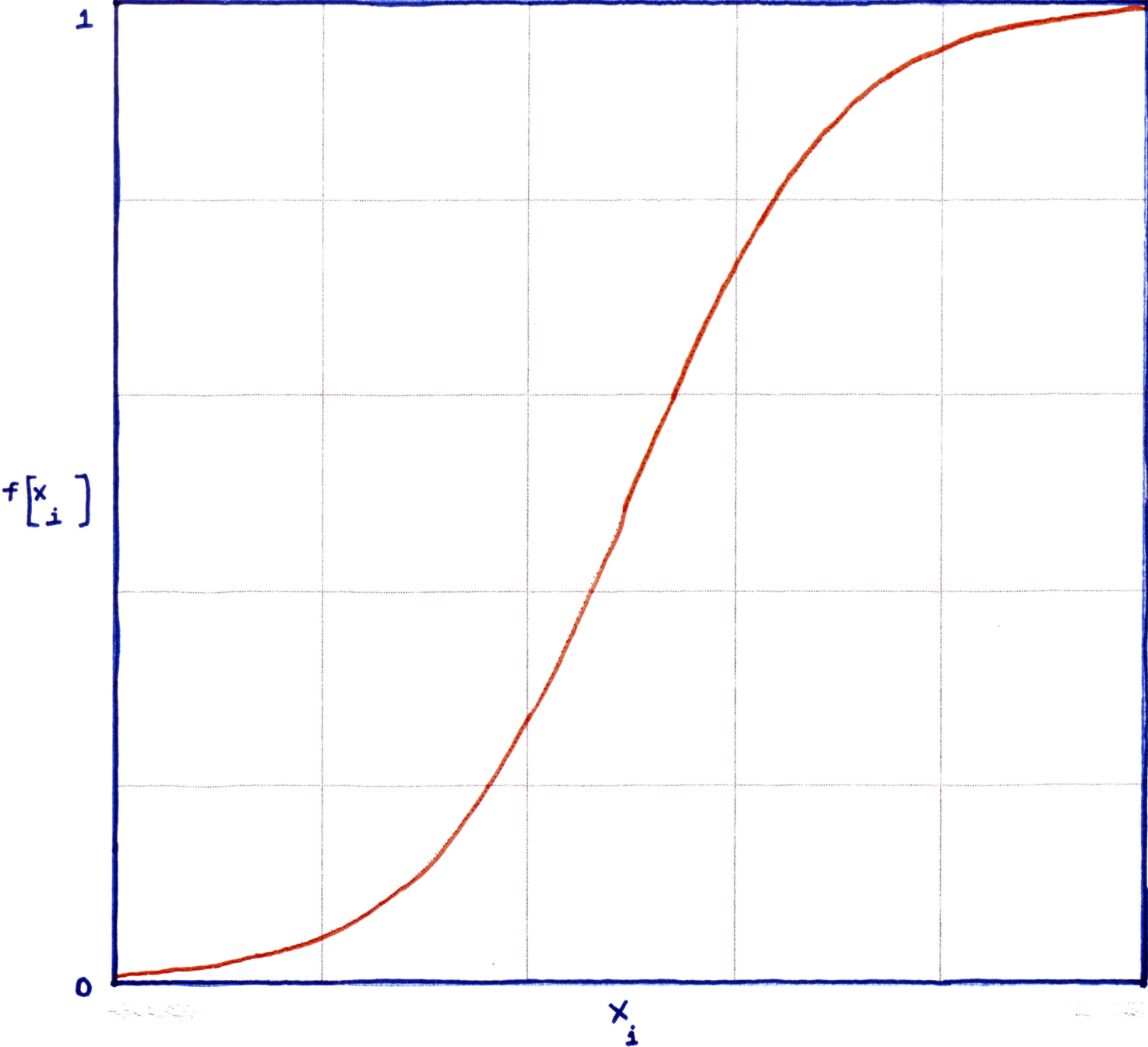


Figure 2



NEURAL NETWORK SIGMOIDAL ACTIVATION FUNCTION
FIGURE A1



EXAMPLE ONE

Calculate the Gramm matrix and its determinant given the following matrix (set of simultaneous equations):

$$A = \begin{bmatrix} 3 & 2 \\ 6 & 3 \end{bmatrix} \quad \text{Gramm } (A) = \begin{bmatrix} 9+4 & 18+6 \\ 18+6 & 36+9 \end{bmatrix}$$

$$\text{Det [Gramm } (A)] = 13 \cdot 45 - 24 \cdot 24 = 9$$

The Det [Gramm (A)] is not zero, therefore the system of equations is linearly independent as is obvious from the initial vector A .

EXAMPLE TWO

The following vector represents one solution of the mapping from the input layer to the middle layer for an R-net approximation to an OR logic gate.

$$A = \text{Weights}_{\text{In-Mid}} = \begin{bmatrix} -2.5811 & -2.2012 \\ -0.0156 & 3.5977 \\ 0.5811 & -3.8408 \end{bmatrix}$$
$$\text{Gramm } (A) = \begin{bmatrix} 11.51 & -7.87 & 6.95 \\ -7.87 & 12.94 & -13.83 \\ 6.95 & -13.83 & 15.09 \end{bmatrix}$$

$$\text{Det [Gramm } (A)] = 46.18 - 178.05 + 131.86 = -0.01 \sim 0$$

Since the Det [Gramm (A)] is zero, there is linear dependence between the rows (and consequently nodes as each row represents the input to one node's activation function). One node may be removed, training repeated, and the Gramm test administered again to check for independence or triangulation may be attempted. If the triangulated matrix has rows of zeros, these zero rows are linearly dependent upon the row that zeroed them during triangulation. These two rows (the original and the newly

zeroed) can simply add their weights (before zeroing) and form a single node. Their output weights should be summed, also. If the triangulation does not produce zero rows, one recourse is to retrain the network after paring a single node.

Triangulate the matrix (A) to attempt to algebraically reconstruct an independent matrix.

$$A_{\text{TRIANGULATED}} = \begin{bmatrix} 1 & 0.58 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

Here, the final row was zeroed out by the second.

Reconstruct the original matrix as one summed row:

$$A = \text{Weights}_{\text{In-Mid}} = \begin{bmatrix} -2.5811 & -2.2012 \\ 0.5655 & -0.2431 \end{bmatrix}$$

$$\text{Gramm } (A) = \begin{bmatrix} 11.51 & -0.925 \\ -0.925 & 0.379 \end{bmatrix}$$

$$\text{Det } [\text{Gramm } (A)] = 3.51.$$

Since this is greater than zero, the matrix (A) represents a set of linearly independent equations. Replacing the original matrix by this construct results in a good match on output and a reduction of node count. The expected error is about ten percent of the error bound (E) experimentally. A rigorous proof has not been performed.

The final matrix representing the mapping from middle layer to the output node is:

$$A_{\text{Mid-Out}} = [-8.8955 \quad 2.5820 \quad 1.1963]$$

which becomes after reconstruction

$$A_{\text{Mid-Out}} = [-8.8955 \quad 3.7788].$$