ANALYSIS OF TRAINED NEURAL NETWORKS

K. Wojtek Przytula[1] and Don Thompson[2]

[1]Hughes Research Laboratories, 3011 Malibu, Cyn. Rd
Malibu, CA 90265
[2]Pepperdine University, Department of Mathematics
Malibu, CA 90263

Abstract. Neural Networks are typically thought of as black boxes trained to a specific task on a large number of data samples. In many applications it becomes necessary to "look inside" of these black boxes before they can be used in practice. This is done in case of high risk applications or applications with a limited number of training samples. This paper describes several techniques of analysis of trained networks, which can be used to verify that the networks meet requirements of the application. The two main approaches advocated are sensitivity analysis and analysis by means of graphs and polytopes. The algorithms developed for the neural network analysis have been coded in form of a Matlab toolbox.

Keywords. Artificial neural networks, analysis, sensitivity, graph theory

1 Introduction

Neural Networks have been used as an effective method for solving engineering problems in a wide range of application areas. In most of the applications it is essential to be able to ensure that the networks perform as desired in real-life situations. This is typically done by providing a rich training set that represents well the data encountered in practice, and by adhering to proper training and testing procedures. In some applications there exists an additional need to verify that the network has been properly trained and will behave as desired. This need may be caused by a relatively small training set, as a result of high cost of data acquisition, or very high liability in case of system error. One example of such an application is deployment of airbags in automobiles. Here, both the cost of data acquisition (crashing cars) and the liability concerns (danger to human lives) are very significant. The paper proposes a range of techniques for analysis and evaluation of trained neural networks. These techniques provide a clear and compact interpretation of how individual inputs affect the network output, how the

network partitions its input space into class regions, and how well it generalizes beyond the training samples.

Our techniques are intended to be used jointly to provide a clear picture of the trained network. We present two basic methods of analysis: sensitivity analysis and graph analysis. They are described briefly in the two sections that follow this introduction. Our work is captured in a collection of approximately 50 algorithms, which have been coded as a library of routines - a toolbox - for Matlab, a commercial package for mathematical computations. They have been tested on a large number of networks proposed for engineering applications and provided a very good insight into the networks.

The techniques are applicable to all multilayer neural networks without feedback, trained as classifiers. However, for simplicity of presentation, we will assume that the networks have only one output and that both the input and output values have been normalized to a [0, 1] interval. We will view the neural network as a multivariate function N, which maps input vector $x \in X$, X - network input space, into $y \in Y$, $Y = [0,1]$ - network output space, such that $y = N(W,x)$, where W is a parameter matrix (the weight matrix). The parameter matrix is fully defined once the network is trained. We will also assume that the single network output is used to distinguish between two classes of inputs, so that output values in the interval [0.5, 1] represent one class, called the high-value class, and the outputs from the interval [0, 0.5) represent the other class, called the low-value class. Thus, the boundary between the classes in input space is formed by all the points that map into output value 0.5. This boundary is determined by the network in the training process.

2 Sensitivity Analysis

This class of algorithms is characterized by observation of changes of network output, including crossing of the threshold 0.5 value, as a result of varying the input values around selected critical points, along particular lines and line segments and on selected planes in the input space. The choice of the critical points as well as the 1-D and 2-D regions is made using several sources of information. They include: training data, e.g. input points from the training set, a priori statistical information about the distribution in the input space, e.g. centroids, clusters, critical outliers etc., physical properties of the problem, and finally analysis of the network function itself. The discussion of the choice of the points and regions in the input space other than by the analysis of the function is beyond the scope of this paper.

There are several results known in literature, which are related to some of the sensitivity algorithms discussed in this paper. See Goh and Wong, 1991, as well as Oh and Lee, 1995. They are, however, rather fragmentary in their analysis of

neural networks and therefore mostly ineffective. Our experience with analysis of real networks points to a need for more comprehensive and multifaceted approach akin to that presented in this paper.

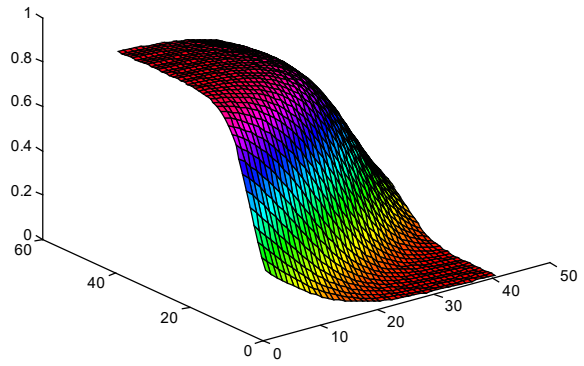

Figure 1. Well-trained and well-generalizing network

The neural network function is a highly nonlinear one defined over a region of high dimensionality. As a result of training it is forced to take on high values (close to 1) near the points from one class of inputs and low values (close to 0) near the other class. In the domain outside of the training points it transitions from high values to low, and it contains the boundary separating the classes. The generalization properties of the network depend greatly on the behavior of the function near the training points and in the transition domain.

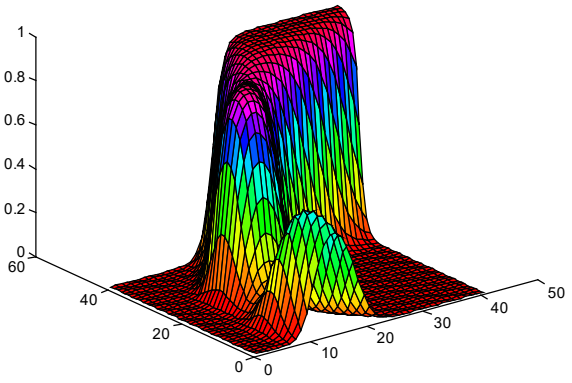

Figure 2. Insufficiently trained network

Figure 3 shows an overtrained network with poor capability for generalization. The slopes are very steep and high plateaus very small. The network created small, separate islands for clusters of points from high-value points. Small perturbation of the input values for some of these points leads to an abrupt change in the classification.

The algorithms used in sensitivity analysis apply Monte Carlo methods, various approximate measures of variability of the function value, and visualization techniques. Here, for illustration, we will show only how some results of the analysis highlight the characteristics of trained networks, see figures 1, 2 and 3. These figures present 2-D projections of the neural network function onto planes identified as important in the process of analysis. Each figure shows a different network developed for the same problem using the same training set. Figure 1 is typical for well-trained and generalizing network. The network function transitions smoothly and gradually from a high plateau of one class to a low plateau of the other class. Figure 2 presents a network that has not been trained sufficiently. Here we see a "hesitant" transition: steep slopes and oscillation.

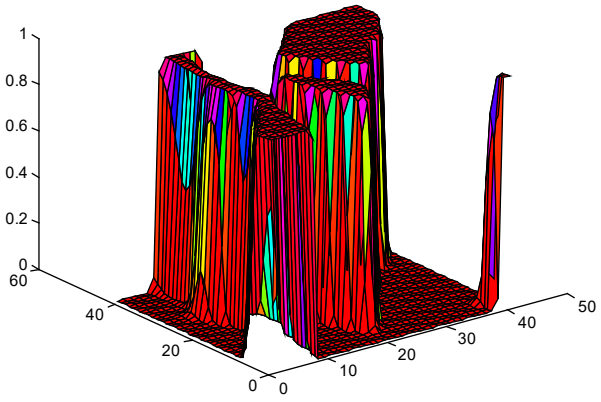

Figure 3. Overtrained and poorly generalizing network

3 Convex Polytopes and Graph Analysis

This category of algorithms helps describe the location and shape of the boundary between the input classes of the network using convex polytopes, which are

generated from training exemplars. Convex polytopes in input space are of interest because of the following theorem:

Theorem 1: Let C be a convex polytope in the input space of a neural network having one output neuron. Then N(C) = [a,b].

Proof: Since N is a continuous function, it maps the connected set C into a connected subset of [0,1], which is therefore an interval (or a single point). See Royden, 1968.

We are interested in convex polytopes belonging wholly to one or the other input class. Such polytopes help define the structure and boundaries of these input classes. We find them as follows: given the set, E, of training exemplars of a given input class, we select a maximal subset of exemplars whose convex closure is also of the same input class. That is, in the case of low output, for example, we select a vertex set $C = \{x_1, x_2, \ldots, x_m\}$ so that both $0 \leq N(x_i) < 0.5$, for all i, and $0 \leq N(\Sigma\lambda_i x_i) < 0.5$ for all $\{\lambda_1, \lambda_2, \ldots, \lambda_m\}$ where $\Sigma\lambda_j = 1$, $0 \leq \lambda_j \leq 1$ for all j, and so that no other vertices in E can be added to C while preserving these properties. Figure 4 depicts five such sets in 2-D.

In order to find such convex polytopes, we start with a related structure: that of a graph in input space. The vertices in such a graph are training exemplars which belong to the same input class. Two vertices of the graph are joined by an edge if all points on the line segment between them also belong to the same class.

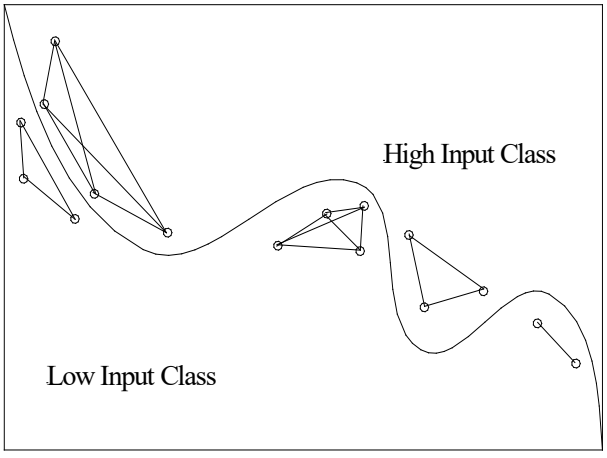

Figure 4. Maximal Convex Polytopes in 2-D

In particular, we are interested in maximal "complete" graphs. A complete graph has the property that every pair of vertices is joined by an edge, (Even, 1979). It is called maximal if no additional vertices (exemplars) can be appended without spoiling the completeness property. From the maximal complete graph we obtain the convex polytope as a convex closure of the graph's vertices. All the points of the polytope belong to the same class as the vertices if the network function N(x) is sufficiently smooth and close to linear in the vicinity of the polytope.

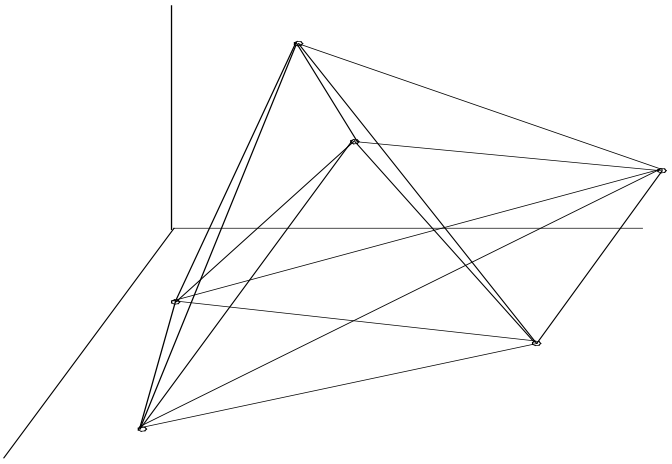

Figure 5. Complete Graph Built on Six Exemplars in 3-D.

Figure 5 depicts a three dimensional version of such a complete graph. We have developed methods of finding all maximal complete graphs of a given neural network by a series of Matlab algorithms. From them we can compute the maximal convex polytopes in each of the input classes.

Furthermore, having found a maximal complete graph, we seek to enlarge it beyond the known exemplar configuration and thus generate a larger portion of input space as well as an improved approximation of the class boundary. We begin by finding all maximal complete graphs in a given neural network for a given input class. The graph to be enlarged is the one having greatest average edge length. We choose average edge length for two reasons. First, we are able to single out the graph with the greatest dispersion of vertices as measured by overall edge girth, and second, we ensure that all edges play a role in the graph selection, especially those edges internal to the convex polytope. Because vertices internal to the convex polytope contribute to a larger average edge length value, we select the complete graph that is most likely to produce a convex polytope belonging to the same class.

We then select one edge $e=(x_i,x_j)$ from all of the edges in this graph and extend it at one endpoint x_i while preserving the completeness nature of the resulting graph. That is, the edge is stretched as far as possible as long as the remaining vertices and the expanded vertex remain a complete graph. The edge choice, e, is made as follows: for each vertex x in G, we locate the closest exemplar y in the other class. Then, we locate the point on segment xy that belongs to the boundary, from which we determine the normal vector n_{xy}. Next, we let $d = \Sigma n_{xy}/m$. (Thus, d represents the average normal direction from the boundary to the graph.) Finally, we choose the edge e having the largest projection onto d.

This approach extends a normal vector technique discussed in Lee and Landgrebe, 1997.

References

1. Even S., (1979) *Graph Algorithms*, Rockville, Maryland, Computer Science Press.
2. Goh T.H., Wong F., (1991) Semantic Extraction Using Neural Network Modeling and Sensitivity Analysis, *Proceedings IEEE International Joint Conference on Neural Networks*, 18-21.
3. Lee C., Landgrebe D., (1997) Decision Boundary Feature Extraction for Neural Networks*, IEEE Transactions on Neural Networks,* **8**, 1, 75-83.
4. Oh S., Lee Y., (1995) Sensitivity Analysis of Single Hidden-Layer Neural Networks, *IEEE Transactions on Neural Networks,* **6**, 4, 1005-1007.
5. Royden H.L., (1968) *Real Analysis*, New York, MacMillan.