Theory and scope of exact representation extraction from feed-forward networks

Ofer Melnik*

DIMACS Center
Rutgers University, Piscataway, NJ, USA

Jordan B. Pollack

Volen Center for Complex Systems
Brandeis University, Waltham, MA, USA

Abstract

An algorithm to extract representations from feed-forward threshold networks is outlined. The representation is based on polytopic decision regions in the input space– and is exact not an approximation. Using this exact representation we explore scope questions, such as when and where do networks form artifacts, or what can we tell about network generalization from its representation. The exact nature of the algorithm also lends itself to theoretical questions about representation extraction in general, such as what is the relationship between factors such as input dimensionality, number of hidden units, number of hidden layers, how the network output is interpreted to the potential complexity of the network's function.

Key words:
artificial neural networks, feed-forward networks, multi layer perceptrons, rule extraction, hyperplane arrangement, polytopes, generalization, artifacts

1 Introduction

There are important reasons to analyze trained neural networks: For networks deployed in real-world applications, which must be predictable, we want to use

* Work was conducted at Brandeis University
Email addresses: melnik@cs.brandeis.edu (Ofer Melnik), pollack@cs.brandeis.edu (Jordan B. Pollack).

network analysis as a verification tool to gauge network performance under all conditions. In *Data Mining* applications, where networks are trained in the hope that they may successfully generalize, and in so doing capture some underlying properties of the data, we want to use network analysis to extract what the network has learned about the data. Or conversely, when a network fails to generalize, we want to use network analysis to find the causes of its failure.

However, despite being prolific models, feed-forward neural networks have been ubiquitously criticized for being difficult to analyze. There are multiple reasons for this difficulty: First, since neural networks consist of a large quantity of interconnected processing units that continuously pass information between them, there is a high degree of interdependence in the model. This implies a lack of locality, where small perturbations in one location can affect the complete network. Therefore, it is not possible to modularize a network's functionality directly with respect to its architecture, rather it has to be analyzed as a whole. Second, the input dimensionality of the network implicitly limits its comprehensibility. Most people's intuition resides in two or three dimensions. Higher dimensionality impedes our ability to comprehend complex relationships between variables. Third, the processing units are non-linear. As such, they are not easily attacked with standard mathematical tools.

Different approaches have been proposed to analyze and extract representations from neural networks (covered in the next section). But other than Golea's (Golea, 1996) proof of the NP-hardness of discrete network analysis, little work has been done to address questions of theory and scope for representation extraction in general. Some of these general questions relate directly to the previously discussed reasons for network analysis, including: How does the computational cost of verifying a real-world neural network scale? What are the common properties of networks that generalize well and those that do not? The focus of this paper is to address these and other questions. Specifically, on the theory front we are interested in the following questions:

(1) How does the potential complexity of a neural network change as a function of the number of its input dimensions and its hidden units? This relationship acts as a lower bound on the computational complexity of any representation extraction algorithm.
(2) What are the computational costs of calculating the exact error between an approximate representation and the actual function that a neural network is computing? These costs act as a bound on the difficulty of verifying the correctness of an extracted representation.
(3) How does the number of hidden layers, and the interpretation of the network outputs affect what a network can do? This correspondence is related to how the quality of a representation might change across different network models.

The questions of scope address what aspects of a network's function can be inferred from an extracted representation. Some questions we address are:

(1) How can we gauge the potential generalization properties of a network by examining its extracted representation? In general, what properties might a network with good generalization exhibit that can be detected in its representation?
(2) Where can we expect the network to exhibit unpredictable behavior or artifacts? What is the form of these artifacts that can be detected in an extracted representation?

The tool that we use to address these fundamental questions of theory and scope is a new algorithm able to extract exact and concise representations from feed-forward neural networks. By being an exact algorithm its properties reflect directly on all other representation extraction algorithms. Since any properties it exhibits or illustrates are inherently related to properties of the underlying networks it analyzes, and as such delimit the performance of representation extraction in general. Such an algorithm will help address the theory questions in the following ways:

(1) The computational complexity of the representation extraction algorithm is related to the potential complexity of the network. Thus, how the resource usage of the algorithm changes with variations in the number of network inputs or hidden layers is indicative of underlying network complexity.
(2) The computational cost of performing measurements on the exact representation is related to the cost of estimating how well other representations capture a networks function.
(3) How the computational complexity of the algorithm changes when hidden layers are added, or how this computational complexity changes when the network output is interpreted differently, are both directly related to how these variations affect the network's potential complexity.

The questions of scope require that the representation will allow us to understand what features of a network's function are conducive or detrimental to good performance and generalization. Only by having an exact representation can we be sure that we are examining all the relevant aspects of a network's function, and not missing information. The scope questions are approached by example– by examining the representations of different networks. Specifically, these questions are addressed in the following ways:

(1) By having a trustworthy representation, and some intuition about what generalization entails, we can compare and contrast the representations of networks with good and poor generalization to elucidate common properties of both.

(2) By scrutinizing the representations of networks we can determine the form that deviations from desired function or artifacts take on. Ideally we would like to be able to predict what network situations are more likely to introduce artifacts.

This paper is structured into three main parts: a description of the algorithm, an examination of the scope questions through example, and a discussion of the theoretical results. We start by presenting the goals of the algorithm and contrast those with other network analysis methods, leading to an examination of the principles of network computation on which the algorithm are based. The algorithm is then described in its basic form for single hidden-layer, single output threshold multi-layer perceptrons.

The example sections begin with examples that serve to introduce the representation while addressing the kind of artifacts present in multi-dimensional networks. We then give examples that contrast representations of networks that do generalize well to those that do not.

The theoretical discussion first addresses the complexity of the basic algorithm and the relationship to potential network complexity. This complexity discussion is continued by describing how the algorithm can be extended to multi-layer multi-output networks and its computational implications, and also by examining complexity from the perspective of representation validation. We then discuss the differences between threshold and sigmoidal networks, where we examine what effect this change of activation function has on their representational ability. We conclude the discussion by tackling the issue of whether learning algorithms can successfully exploit all the potential complexity available in these networks.

2 Network Analysis Methods

To address the questions asked above we seek a way to extract from a network's parameters an alternative representation of its function, a direct representation, one without interdependence between parameters. The representation should be exact, matching the network's function fully, but concise, not introducing redundancy. Only if all three properties are met can the algorithm truly reflect the full underlying computation of these networks and their complexity. Thus, in examining the different approaches to neural network analysis proposed in the literature we need to focus on whether they exhibit these properties of being exact, concise and direct. There are different ways to categorize approaches to network analysis, of these we choose to examine them by the form of their results or representation.

The *rule extraction* approaches (Andrews et al., 1995; Bologna, 1998; Craven and Shavlik, 1995; Setino, 1997; Thrun, 1993) extract a set of symbolic "rules" to describe network behavior. These algorithms can be divided into two broad categories by their treatment of the network, *decompositional* and *pedagogical*. The decompositional approaches try to find satisfiability expressions for each of the network units with respect to its inputs. Depending on the algorithm, this is achieved by first applying discretization, large scale pruning or placing structural limitations on the networks and then performing an exhaustive search on the inputs of all the units. To their credit, by performing an exhaustive search on all the network's constituent units, the decompositional approaches offer a complete alternative representation to the network's function. Nonetheless, except for the cases where the network structure is explicitly limited to facilitate rule-extraction, the rules extracted are not exact, as in most cases the network's function can only be approximated by rules. This representation is also not independent as the rules represent individual network units and as such maintain their distributed representation dependence.

The pedagogical and hybrid approaches to rule extraction do not decompose a network unit by unit, rather they construct rules by sampling the network's response to different parts of the input space, using varying degrees of network introspection to refine their regimes. Sampling makes these algorithms computationally feasible, and depending on the algorithm at times independent, but the algorithms fail to examine the full space of possibilities of network behaviors, so their rules are typically greater approximation than the decompositional approaches.

Weight-state clustering (Gorman and Sejnowski, 1988), *contribution analysis* (Sanger, 1989; Shultz et al., 1995), and *sensitivity analysis* (Intrator and Intrator, 1993, 1997), which do use the network's parameters directly in their analysis, unlike rule extraction do not generate an alternative representation, rather they try to ascertain the regularity in the effect that different inputs have on a network's hidden and output units. That being the case, these methods do not explain global properties of the network, but are limited to specific inputs and like the pedagogical approaches, do not meet our criteria of being exact.

Hyperplane analysis (Pratt and Christensen, 1994; Sharkey and Sharkey, 1993) is a technique by which the underlying hyperplanes of neural network's units are visualized. As such, it is global and uses the network's parameters in its analysis. However, it does not remove interdependence, and does not scale since it is based on understanding the network interdependencies by direct visualization.

Network inversion (Linden, 1997; Maire, 1999) is a technique where locations

in the input are sought which generate a specific output. Maire's work is promising in that it approaches the problem by back-propagating polyhedra through the network and thus generates an exact direct representation. However its main shortcoming is its lack of conciseness, each stage of inversion can generate an exponential number of sub-polyhedra.

Ideally we would like to construct an algorithm that extracts exact, direct and concise representations from any arbitrary neural network. This is not feasible, however. The main impediment is that the functional form of the alternative representation is dependent on the type of activation function used. That means that for different activation functions we would need to use different types of representations, breaking our notion of a unified algorithm. Nevertheless, with different monotonic activation functions the basic methods of network generalization remain the same. Therefore, the approach taken in this paper is to construct an algorithm for a specific activation function (threshold) to address the questions as they pertain to this activation function, and then discuss what effects varying the activation function would have. The concept behind the algorithm is to find the limitations or constraints on how the network output can change under differing input conditions. Stated another way: How does the network architecture govern its decision regions?

3 Decision Regions

The output of a network, interpreted as a classifier, partitions the space of its inputs into separate decision regions. For each possible network output value, there exists a corresponding regions in the input space such that all points in those regions give the same network output. Hence the name *decision region*, since a region reflects the network's output or decision.

The decision regions encompass the full function that a network computes. They describe the complete mapping between the input and the output of the network. Unlike the original network, which does this mapping through the interdependent computation of many units, the decision regions map the input and output directly.

Decision regions can have different shapes and other geometric properties. These properties are directly related to the network architecture used. The Decision Intersection Boundary Algorithm (DIBA) is designed to extract decision regions from multi-layer perceptron networks, a feed-forward network with threshold activation functions. The decision regions for this type of network are polyhedra, the n-dimensional extension of polygons. The algorithm is based on a few principal observations about how multi-layer perceptrons compute.

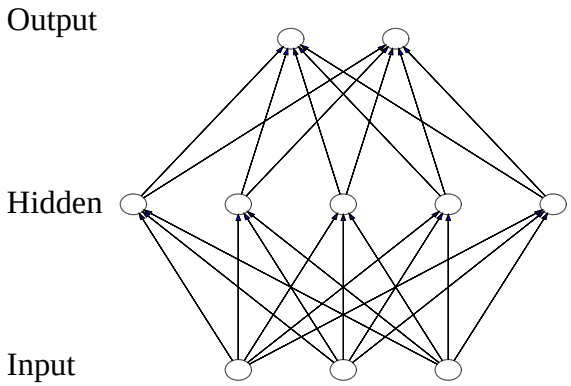

Figure 1. A 3-layer perceptron neural network.

Consider a three layer perceptron neural network, with each layer fully connected to the successive layer as in figure 1, where the output of each of the units is 1 if $\sum_i weight_i input_i > 0$ and 0 otherwise. The first observation is the independence of the output units. There is no information flow or connectivity between any of the output units. This is due to the strictly feed-forward nature of the network. Effectively, each output unit is computing its own value irrespective of the other output units. As such, for now, we can treat each output unit separately in our analysis of the network, generating separate decision regions for each output unit. Since this is a perceptron network with threshold activation functions, an output unit can have either a value of 0 or 1. Therefore we can choose to build decision regions corresponding to either an output value of 0 or 1.

An output unit value is dependent on the activation values at the hidden layer, which are dependent on the values at the input layer. Since the hidden layer units' output can only take on a value of 0 or 1 as well, their effect on the output unit's weighted sum is discrete. In fact, the output unit's weighted sum is just a partial sum of its weights. The full value of each weight is either included in the sum or completely left out.

Any location in input space where an output unit switches values is a decision boundary. Such a switch corresponds to a transition in the output unit's partial sum, either going above threshold when it was previously below or vice-versa. Since the partial sum is a function of which hidden units are active, this threshold transition must be coordinated with a state change in one or more hidden units. This means that a decision boundary must correspond to a region in the input space where at least one hidden unit undergoes a state change.

Each hidden unit divides the input space into two regions, a region where its output is 0 and a region where its output is 1, in effect generating its own decision regions. In this context, each location in input space has associated with it a *hidden state*, a binary number corresponding to the output values of all the hidden units when given that input location. The hidden state corre-

sponds directly to the output unit's partial sum and hence the output unit's value. This leads to the important observation that the only locations in input space where this hidden state can change are across the hidden units' own decision regions, or across the intersection of multiple decision regions. This implies that the basic building blocks of output unit decision regions are the decision boundaries between the intersections of hidden unit decision regions. As such the output unit decision regions are composed of parts of the hidden units' division of the input space.

In the multi-layer perceptron the hidden units divide the input space using hyperplanes. Therefore, the output unit decision regions are composed of high-dimensional faces generated by the intersection of hyperplanes, making them polyhedra. If we want to explicitly describe these faces, we need to specify their individual boundaries. As the intersection of hyperplanes, these boundaries are just lower dimensional faces. Assuming the space is bounded, we can repeat this process recursively, describing each face using its lower dimensional faces, until we reach zero dimensional faces or points. Consequentially, the output unit decision regions can be described by the vertices which delineate them.

4 The Decision Intersection Boundary Algorithm

The basic Decision Intersection Boundary Algorithm is designed to extract the polytopic decision regions of a single output of a three layer perceptron network. Later sections explain the simple modifications necessary to extend it to multiple hidden layers and multiple outputs and discuss the difference between threshold and sigmoid activation functions. The algorithm's inputs are the weights of the hidden layer and output unit, and boundary conditions on the input space. Using boundary conditions guarantees that the decision regions are compact, and can be described by vertices. Its output consists of the vertex pairs, or line segment edges, which describe the decision regions in input space.

As described in the previous section, decision region boundaries are at the intersections of the hidden unit hyperplanes. Thus the algorithm consists of two parts, a part which generates all the possible vertices and connecting line segments by finding the hyperplane intersections, and a part which evaluates whether these basic elements form the boundaries of decision regions. In figures 2 and 3 we see an illustration of these two parts of the algorithm.

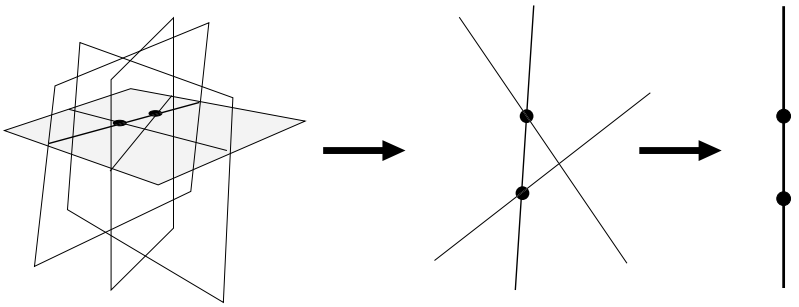

Figure 2. The generative portion of the DIBA algorithm recursively projects the hidden layer hyperplanes down till they are one-dimensional, in order to generate all the intersections of the hyperplanes. Here we see an illustration of two stages of projection from three dimensions to two and from two to one.

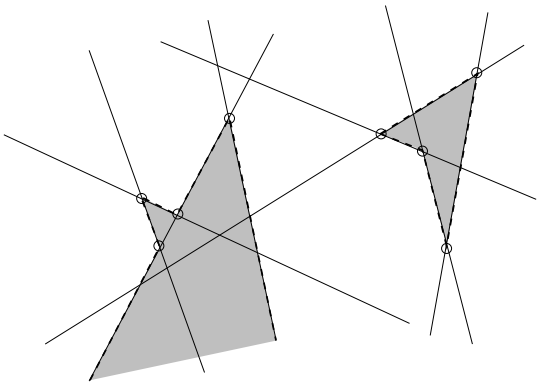

Figure 3. The boundary testing portion of the DIBA algorithm evaluates individual vertices and line segments, that were generated by the recursion, in order to test whether they form corners or edges of decision regions. Here we see the edges and corners of two decision regions the algorithm would recognize in an arrangement of hidden unit hyperplanes.

4.1 Generative Recursion

The generative part of the algorithm is recursive. For each hidden unit hyperplane of dimension n, the algorithm finds the intersections with all the other hyperplanes. These intersections, that are hyperplanes of dimension *d-1*. This procedure is performed recursively until one dimensional hyperplanes, or lines are reached– lines are the basic unit of boundary evaluation in this algorithm.

function extract(*hidden, out, borders*): Return Representation
Input Hidden Units' weights, Output Unit's weights, Borders
Output Representation containing vertices and line segments

if hyperplane_dimension(*hidden*) > 1
    for *base_hyperplane* in *hidden*

```
        for hyperplane in hidden
            if hyperplane ≠ base_hyperplane
                new_hyperplane :=
                    find intersection of hyperplane with base_hyperplane
                if hyperplane is parallel to base_hyperplane then
                    store state of hyperplane with respect to base_hyperplane
                else
                    add new_hyperplane to new_hidden
                endif
            endif
        endfor
        call extract(new_hidden, out)
    endfor
else execute the traversing-the-line routine (appendix A)
```

4.2 Boundary Test

Along the line, the locations of output unit value changes are at the intersections with the remaining one-dimensional hidden unit hyperplanes. At each such location the algorithm needs to test whether the intersection, a vertex, forms a corner boundary– and if the intervening line segment forms a line boundary.

In order to understand the boundary test we need to examine the concept of a boundary in a d-dimensional input space. We start our examination by looking at one hidden unit. If a single hyperplane acts as an output unit border, it implies that at least for a portion of the hyperplane, in a neighborhood on one side of the hyperplane there is one output unit value and in the corresponding neighborhood on the other side of the hyperplane there is a different output unit value. For example, (see figure 4) if we take our input space to be two dimensional and use a line as our hyperplane, the line acts as a boundary if the output unit value on one side of the line is 1 and the output unit value is 0 on the other side of the line. One can think of the hyperplane as forming the boundary. Inversely, one can think of the boundary taking on the form of a hyperplane in that particular region in space. That is, the location in input space where we have an output value transition can be described by a hyperplane. This is the functional view of the boundary. To functionally describe a boundary at that part of the space we need a hyperplane. Taking this view, a boundary is composed of various geometric entities which demarcate output value transitions in input space– and the corner test becomes: do we need a lower dimensional geometric entity to describe the boundary at the intersection of multiple hyperplanes?

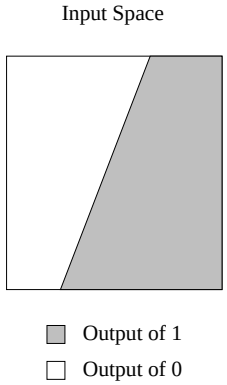

Figure 4. The decision region formed by the division of a two dimensional space by a hyperplane line. The line could be said to be partitioning the space. Or, in a complementary, functional fashion it could be said that the form of the boundary in the space is in the shape of a line.

Take two non-parallel hyperplanes which act as boundaries. Around each of these hyperplanes the boundary takes the form of a *(d-1)*-dimensional manifold. If at their intersection both hyperplanes still act as boundaries, then the functional geometric form of the location with this output value transition across both hyperplanes is the *(d-2)*-dimensional hyperplane formed by the intersection of the two hyperplanes. One can say that the corner formed at the intersection is the description of the location of a complex boundary across multiple hyperplanes.

To illustrate this, let us examine under what circumstances an intersection of lines (2-dimensional hyperplanes) forms a complex boundary. Two lines intersect at a vertex, this vertex could be a corner, a zero-dimensional boundary, or not. In figure 5 we see different boundary configurations at this vertex. Two of the configurations form corners, while the others do not. The two that do form corners have one element in common, both of the hyperplanes that make up the corner are still boundaries in their own right. That is, in the vicinity of the corner both hyperplanes have a part of them that engenders one output unit value on one side of the hyperplane, and another on the other side. As stated before, this corner marks a location in input space where a complex transition takes place, a transition across multiple hyperplanes. This corner test can be naturally generalized to higher dimensions. **In general, what we seek in an intersection that forms a boundary, is that all hyperplanes making up the intersection have at least one face that is a boundary in its vicinity.**

How do we practically check for this intersection boundary condition? In a d-dimensional input space, an intersection of $n \leq d$ hyperplanes partitions the space into 2^n regions (see figure 6). If we consider our hidden units as these hyperplanes, then the space is partitioned into the 2^n possible hidden states. Each possible hidden state with respect to these hidden units is represented

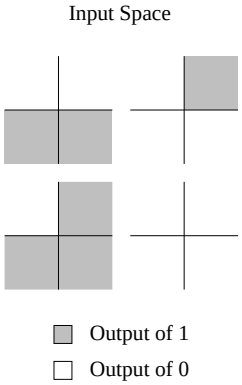

Figure 5. Some possible decision regions formed at the intersection of two lines. Corners are only formed if both lines are boundaries.

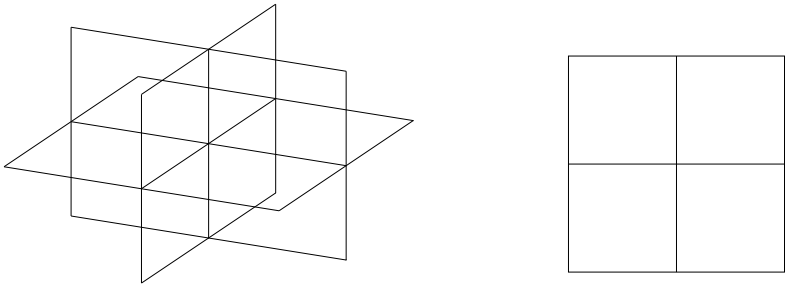

Figure 6. The partitioning of a 2 and 3 dimensional space around the intersection of hyperplanes.

in our input space around the intersection. A hidden unit hyperplane has a boundary face within the input space partitioning of the intersection, if within these 2^n hidden states there exist two hidden states that differ only by the bit corresponding to the hidden unit hyperplane being tested, such that one hidden state induces one value at the output unit and the other state induces another value. This basically says that at least in one location of the partitioning, if we cross the hyperplane we will get two different output unit values. Thus algorithmically the corner test is to go through all possible hidden states at the intersection and check that each hidden unit acts as a boundary.

In the 3-layer single output network case, where the output unit just computes a partial sum of its weights, the test simplifies to finding whether the hyperplane corresponding to the smallest absolute valued weight has a boundary in the intersection. Of course, this is a necessary condition, but it is also sufficient, since if the hyperplane of the smallest absolute valued weight has a boundary then all other hyperplanes making up the intersection must also have boundaries. This is shown mathematically in appendix B.

The boundary test for line segments and corners is the same, since both lines

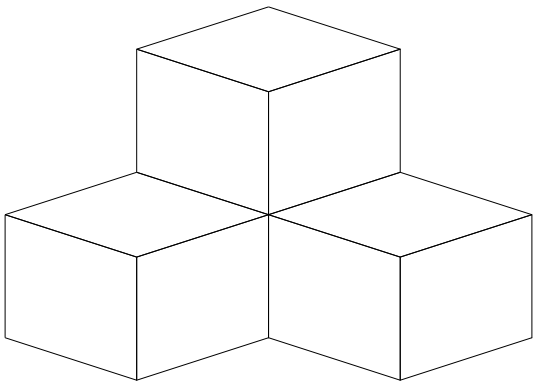

Figure 7. Unlike two dimensional spaces, a corner in a three dimensional space can be in the middle of multiple line segments.

and vertices are just intersections of hyperplanes. Lines are the intersection of *d-1* hyperplanes and vertices are the intersection of d hyperplanes.

4.3 Traversing the Line

When do we perform the boundary test? In two-dimensional input spaces, a corner always either starts or ends a line segment; a corner can not be in the middle of two line segments. In contrast, in higher dimensional spaces, corners can be in the middle of line segments. Figure 7 demonstrates how a three-dimensional corner can be in the middle of multiple line segments. However, even in high dimensional spaces a line segment can only start and terminate at the corners which delineate it. Therefore in traversing a line, we need to check corners at all hidden unit intersections, and check for line segments only between corners which start and end them.

For the intersection boundary test there are two kinds of hidden unit hyperplanes: hyperplanes that make up the intersection and hyperplanes that do not, but whose state *does* affect the output unit value by projecting over the intersection. Before performing the boundary test on a line segment or vertex, we need to assess the contribution of these additional hyperplanes to the output unit partial sum in the intersection vicinity. It is possible to simply sample the hidden state of these hidden units in the vicinity of the intersection. But on the line there is a more economical solution. On the line we are traversing, the projection of these additional hyperplanes is a point. These hyperplanes each divide the line into a half where their hidden state value is 1 and a half where their hidden state value is 0. This gives a directionality to each hyperplane. That is, if we traverse the line from one end to the other, some of the hyperplanes will be in our *direction*, meaning that as we pass them their hidden value state will become 1, and the others will be in the other direction, their hidden value state changing to 0 as we pass them (see figure 8.) We can

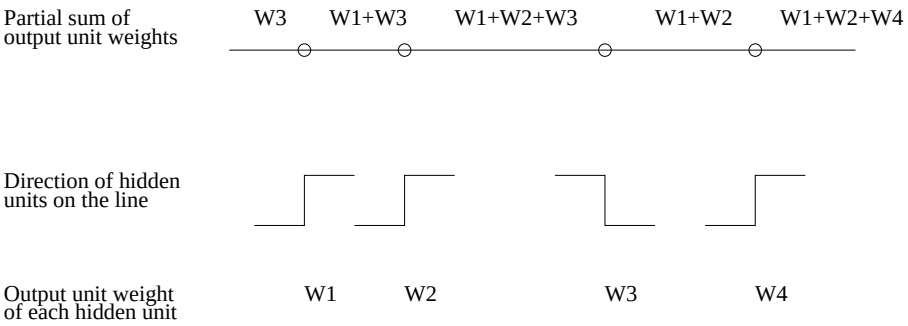

Figure 8. The bi-directionality of hyperplane intersections on a line suggests a two pass algorithm to calculate the partial sums of the output unit weights.

use this property to incrementally quantify the contribution of these units to the hidden state/partial sum.

The bi-directionality suggests a two pass algorithm. Initially, we arbitrarily label the two ends of our line, *left* and *right*. The forward pass starts with the leftmost hyperplane and scans right, hyperplane by hyperplane. We are interested in finding the partial sum contribution for each line segment between hyperplanes. So, as each hyperplane is encountered it is checked for right directionality, and its weight is tallied to a running sum of the weights. This sum is assigned to the current line segment, accounting for the contribution of all the right directed hyperplanes on this line segment. The backwards pass is identical, except it starts from the rightmost hyperplane and scans left, adjusting its running sum with left directed hyperplanes, and adding its sum to the values already generated in the forward pass. Thus, completion of both passes calculates the partial sum contribution of all left and right directed hyperplanes for all the line segments on the line. The actual boundary tests can be conducted during the backwards pass, since the full contribution to the partial sum has been tabulated at that stage. Appendix A contains a pseudo code description of the "Traversing the Line" portion of the algorithm.

5 Examples

The following are examples of applying the Decision Intersection Boundary Algorithm on trained three layer perceptron networks. The training consisted of a mixture of momentum back-propagation and weight mutation hill-climbing [1]. However the units were always treated as threshold for the purposes of the DIBA algorithm. The examples are used to demonstrate the kind of decision region descriptions that can be extracted using the DIBA algorithm and how

[1] We do not go into the specifics of these generic training methods, as the focus of the paper is on the trained networks, not how they were trained. For further reading, see (Bishop, 1995) on these and other training methods.

they can be interpreted. In parallel, by examining the representations of multiple networks we address the questions of scope: What is the form of network artifacts? And what can we learn about how networks generalize from their representation?

5.1 Sphere Networks

Networks construct decision regions to enclose data on which they are trained. Artifacts or noise appears as decision region structure that is unrelated to the training data. By looking at similar networks with different dimensionality, the following three examples demonstrate the relationship between a network's input dimensionality and propensity to noise.

5.1.1 Two-Dimensions: The Circle

Using back-propagation an 80 hidden unit sigmoidal neural network was trained to classify 300 points inside and outside of a circle of radius 1 around the origin. Treating the activation functions as threshold, the DIBA algorithm was then applied to the weights of the network. In figure 9 we see the decision region extracted at the origin, and the points corresponding to the training sample. The decision region allows us to directly view what the network does– exactly where it succeeds and where it fails. The representation is concise, since all the vertices used are necessary to completely describe the decision region. With this exact representation we can make out the nuances of the network's function, for example, note the small protrusion on the bottom right part of the decision region that covers two extremal points.

The DIBA algorithm also allows us to examine other aspects of this network. By zooming out from the area in the immediate vicinity of the origin we can see the network's performance, or generalization ability, in areas of the input space that it was not explicitly trained for. In figure 10 we recognize large artifactual decision regions at a distance of at least 50 from the decision region at the origin. Thus, in this area of the input space where there was no actual training data, a few decision regions formed as artifacts of the network weights that were learned.

5.1.2 Three-Dimensions: The Sphere

A 100 hidden unit network was trained to differentiate between points inside and outside of a sphere centered at the origin. In figure 11 we see the rather successful decision region encapsulating the network's internal representation of the sphere from different angles. Figure 12 illustrates the same phenomena

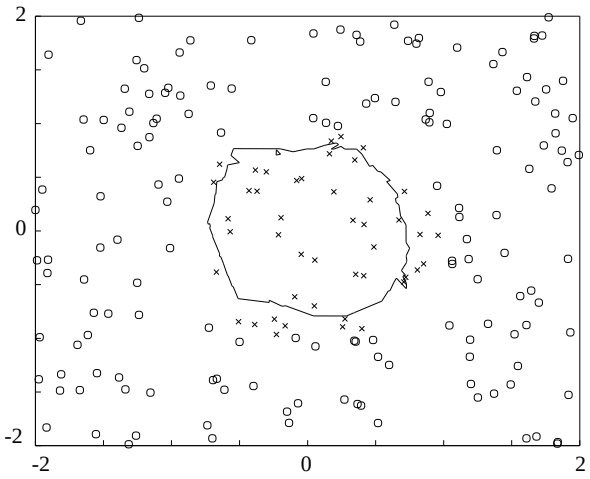

Figure 9. A decision boundary of a network which classifies points inside and outside of a circle.

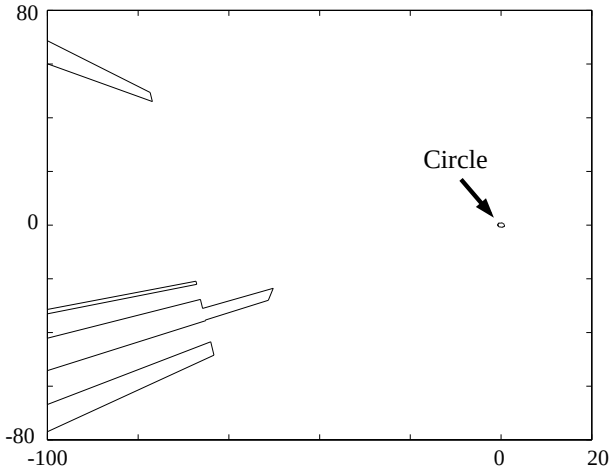

Figure 10. A zoom out of the network illustrated in figure 9 illustrates the existence of artifactual decision regions.

of additional artifact decision regions we saw in the circle for the sphere– the miniature sphere appears amid a backdrop of a large cliff-face like decision region. Note that the artifacts are more complex in the three-dimensional case. As the higher dimensionality of the input space implies the existence of exponentially more hyperplane intersections (vertices) than in the two-dimensional case and with that potentially more artifactual decision boundary corners.

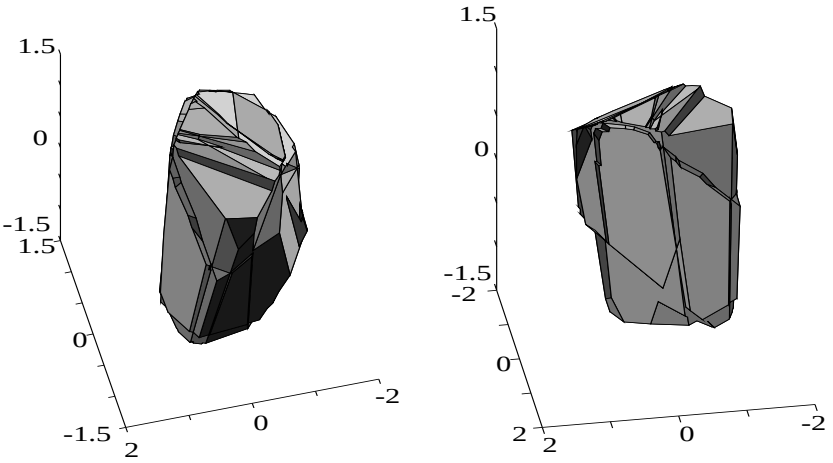

Figure 11. A decision boundary of a network which classifies points inside and outside of a sphere.

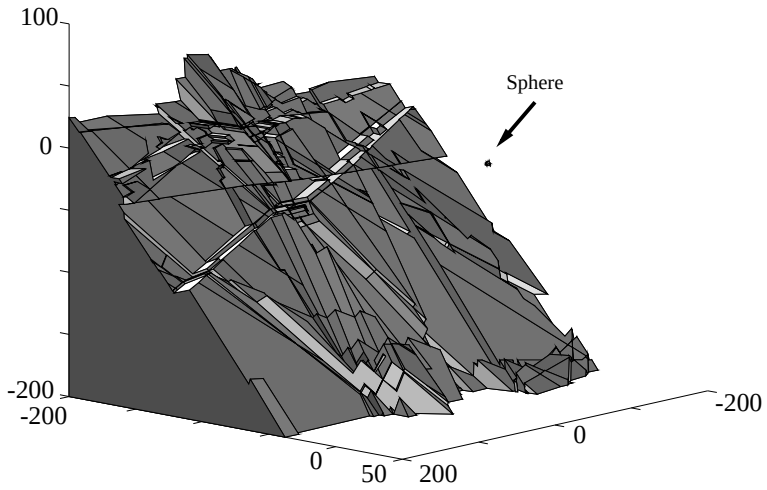

Figure 12. The artifacts around the sphere decision boundary.

5.1.3 Four-Dimensions: The Hyper-Sphere

Another 100 hidden unit network was trained to recognize points inside and outside of a 4-dimensional hyper-sphere centered around the origin. Due to human limitations it is difficult for most people to visualize objects in more than three dimensions (even three can be a challenge at times.) One way to gather information from our high dimensional polytopic decision regions is to describe them in terms of *rules*. That is, we bound each polytope inside of a hyper-rectangle, and examine the rectangles' coordinates. At this first approximation we can elucidate how many decision regions there are, their location in input space and a coarse approximation of their volume. The rectangles can later be incrementally refined to enclose parts of polytopes, thereby giving higher resolution rules, refining our perception of their structure and volume. In this case the hyper-rectangle which covers the polytopes representing the

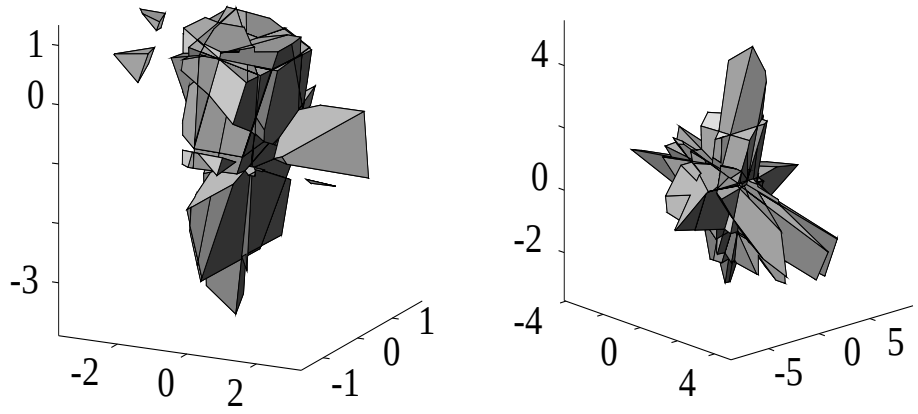

Figure 13. Projections of the four dimensional hyper-sphere polytope across $x_4 = 0$ and $x_4 = 1$.

hyper-sphere has the following minimum and maximum coordinates:

Min:	(-4.45	-5.79	-3.90	-7.93)
Max:	(6.47	6.01	7.39	6.07)

Another way to examine the high dimensional space is to examine projections on to lower dimensional spaces. In figure 13 we see projections of the four dimensional polytope with the fourth component set to zero and one respectively. Needless to say, the four dimensional hyper-sphere looks less and less like a sphere. As before, increasing the number of input dimensions increases the potential for more artifactual boundaries. But this is also coupled with the curse of dimensionality (Bishop, 1995), which says that as we increase the input dimension while keeping the number of training samples constant, our problem (the decision region we are trying to learn) becomes exponentially less specified. Thus, artifacts appear not only at a distance from the training samples, but in higher-dimensional spaces, might appear on our actual decision regions in the form of unwanted complexity.

5.2 Ball Throwing Network

A networks generalization ability is directly related to its decision region structure as defined by the underlying training data. Good generalization can occur if by structuring the data into decision regions, the network uncovers clear underlying properties of how the data is fundamentally organized in the input space.

As a positive example consider a 15 hidden unit network that was trained to predict whether a thrown ball will hit a target. As input, it received the

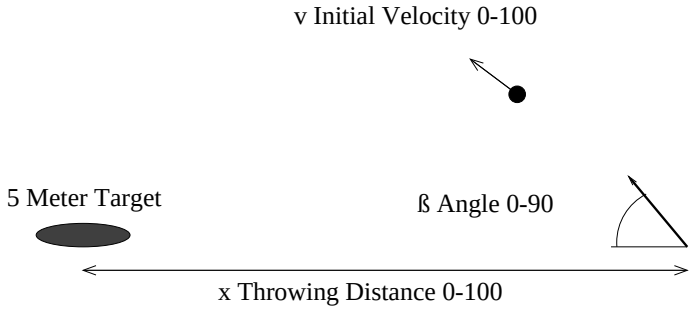

Figure 14. The network is supposed to predict whether the ball will hit the target given throwing angle β, initial velocity v and target distance x.

throwing angle, initial velocity and the target distance (see figure 14.) After training it achieved an 87% success rate on the training data.

This is a simple non-linear system which when solved analytically gives us the following relationship:

$$\left| x - \frac{v^2 sin2\beta}{2g} \right| < 2.5$$

Figure 15 contrasts the decision region generated from the 15 hidden unit neural network with the analytical solution. It is apparent that the network managed to encapsulate the gist of the model, its decision region approximates the analytically derived decision region fairly well. The decision region possesses properties which suggest good network generalization, independently of knowing the actual solution. The decision region clearly defines a specific submanifold of the space. It consists of 4 main sub-decision regions, that are each highly convex but distinct from each other, suggesting that those locations in input space were clearly individually defined in the training set. In addition, the region of input space not in the decision region (the "misses the target" area) is also clearly defined and separated, consisting of a few large convex regions.

5.3 Predicting the S&P 500 Movement

Contrast the previous network with a neural network containing 40 hidden units that was trained to predict the average direction of movement (up or down) for the following month's Standard and Poor's 500 Index. It was trained on monthly macro economic data from 1953 to 1994 from Standard & Poor's DRI BASIC Economics Database. The network's inputs were the percentage change in the S&P 500 from the past month, the differential between the 10 year and 3 month Treasury Bill interest rates, and the corporate bond differential for AAA and BAA rates, indicators used in non-linear economic

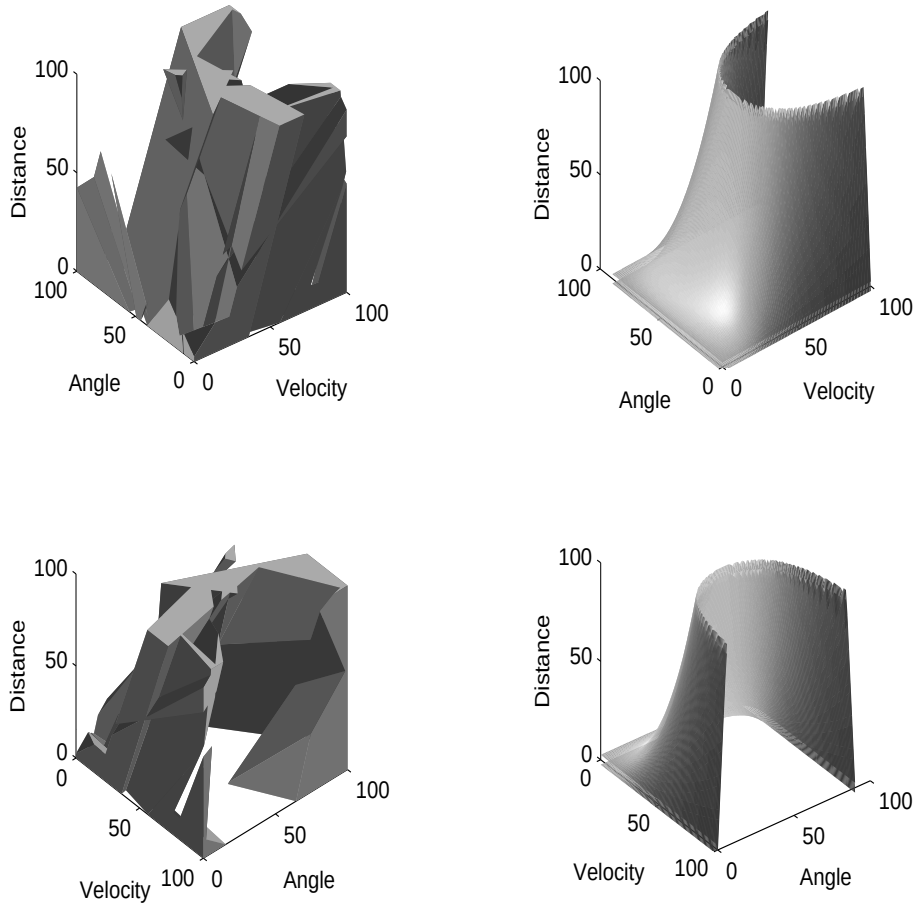

Figure 15. The decision region of the ball throwing neural network (left) contrasted with the decision region of the actual analytical solution (right). Two different perspectives are shown.

forecasting (Campbell et al., 1997). After training, the network achieved a better than 80% success rate on the in-sample data. Figure 16 shows the network's decision regions. from it we can surmise that the network would probably not generalize very well to out of sample data. We can see that rather than learning some underlying regularity, the network tended to highly partition the space to try and enclose all the sporadic data points.

We can use the rule method outlined before to quantify some of these partitioning effects. In the region of input space where the training data resides the network has 28 different decision regions. Of these all but five have a bounding rectangle with a volume of less than one. One decision region has a bounding rectangle which encompasses the whole input space. We can refine the rule for this large decision region by slicing the decision region using another hyperplane and examining the bounding rectangles for the resultant sub-decision regions. If we simultaneously slice this polytope using three hyperplanes, each bisecting the input space across a different dimension, then if the polytope were completely convex, we would expect, at most, to get eight sub-polytopes.

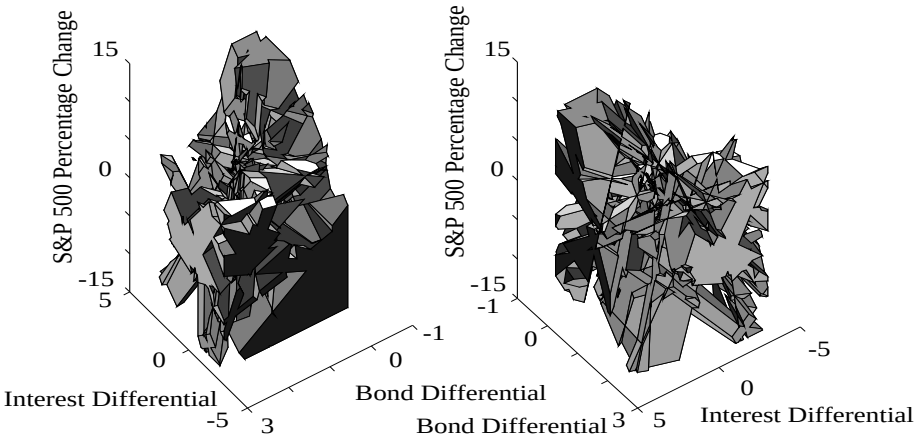

Figure 16. The decision regions of the S&P 500 prediction network.

However for this decision region, this refinement procedure generated 23 separate sub polytopes, implying that the polytope has concavity and probably has some form of irregular branching structure needed to partition the space. A simple analogy would be to contrast an orange with a comb. Both are solid objects. No matter how we slice the orange we will always be left with two pieces. However if we slice the comb across its teeth, it will decompose into many pieces.

5.4 Vowel Recognition

As another example of comparing generalization, this time looking at the effects of dimensionality, two neural networks were trained on the vowel data available at the CMU neural network benchmark repository. This data consists of 10 log area parameters of the reflection coefficients of 11 different steady state vowel sounds. Our interest in this example was to gauge the effect of using different dimensioned input spaces. The reflection coefficients are particularly suited for this test (Makhoul, 1975) because of their mathematical properties: they are orthogonal, the coefficients do not change when larger sets are generated, and using their log area parameters confers a greater spectral sensitivity. Both networks were trained to recognize the vowel sound in the word *had* within the background of the other 10 vowel sounds.

The first network received as input the first two coefficients. After training, it achieved a better than 86% success rate on the training data. Its decision regions and the training data are shown in figure 17. In the input region of $[-3.2, -2.3] \times [0.7, 1.2]$ we see a relatively high degree of partitioning, in that many decision regions are used to secure a perfect classification, implying possibly problematic generalization in that region.

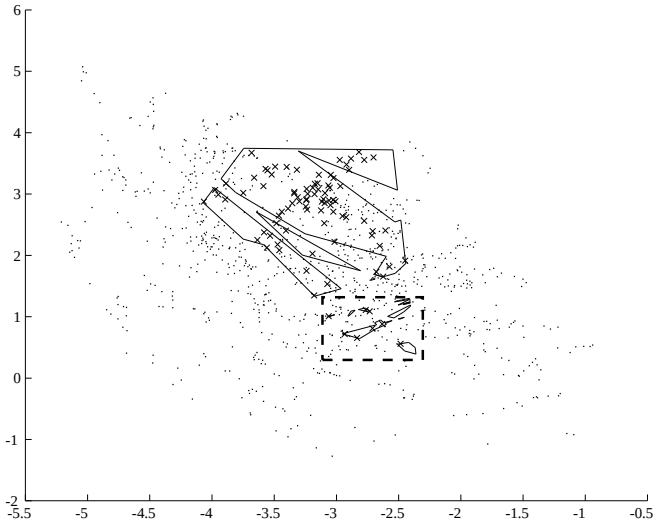

Figure 17. The decision regions of the two input vowel recognition network with the training data in the background. The X's are positive test cases.

The second network received the first four coefficients as inputs, and enjoyed a success rate of over 91% on the training data. It also achieved perfect classification within the input region described. However, it appears to have done so with less partitioning of the space. We can see this using the rule refinement procedure described in the previous example. If we extract the decision regions in this part of input space we get only one rectangle which spans it completely. Conducting the same kind of concavity test we previously used, that is slicing the space using four hyperplanes, each bisecting an input dimension of the region, we get only 10 sub polytopes suggesting a small degree of concavity (less than the 2^4 for a perfectly convex shape). In addition these sub-decision regions are mostly delimited in the third and fourth dimensions with the first two dimensions left to span the whole area of the sub-space. Therefore it appears that the network makes use of these added dimensions to form a more regular decision region in that difficult region of the input space. This must be qualified, since by the curse of dimensionality the increase in the dimensionality made the problem less specified, and as such it became easier to enclose the data points within one, more convex decision region. But the fact that the decision region makes exclusive use of the added dimensions to discriminate, significantly strengthens the claim of potentially better generalization.

6 Discussion

6.1 Algorithm and Network Complexity

The Decision Intersection Boundary Algorithm's complexity stems from its transversal of the hyperplane arrangement in the first layer of hidden units. As such, that part of its complexity is equivalent to similar algorithms, such as arrangement construction (Edelsbrunner, 1987), which are $O(n^d)$, where n is the number of hyperplanes and d is the input dimension. Another aspect of complexity stems from the corner test, which is $O(2^d)$. Even the partial sum case is of similar complexity, since the question of asking whether the lowest magnitude weight acts as a border is equivalent to the *knapsack* problem (Schrijver, 1987) which is NP-complete.

Do we really need to examine every vertex though? Perhaps the network can only use a small number of decision regions, or it is limited with respect to the complexity of the decision regions. In this section we prove that this is not the case by hand constructing a network with $\Omega\left(\left(\frac{n}{d}\right)^d\right)$ different decision regions, where each decision region has 2^d vertices, concluding that networks are capable of having exponential complexity.

We begin with a one dimensional construction of a three layer, single output perceptron network. Start with k hidden units with the same directionality, and assign to them alternating $+1$ and -1 weights. In figure 18a we see such a construction with $k = 4$ hyperplanes. If we set the output unit threshold to 0.5, we see that the hyperplanes partition the input space into three decision regions, a decision region for each line segment with a weight sum of zero.

Let us extend this construction to a two dimensional input space. First we extend the one-dimensional hyperplanes, points, to two-dimensional hyperplanes (lines). We do this by making them parallel in the added dimension. Next we add an additional k hyperplanes that are orthogonal to the original hyperplanes, and again assign to them alternating -1 and $+1$ weights. Figure 18b illustrates this. We see that the additional hyperplanes continue each zero term in a checkerboard pattern in the added dimension, but above zero terms remain above zero in the added dimension. Thus, by adding a dimension, each lower dimensional decision region multiplies to become either $(k+1)/2$ or $k/2+1$ different decision regions, depending on whether k is odd or even.

This construction can be extended to any number of dimensions, so by induction we can see that this construction has $\Omega\left(\left(\frac{n}{d}\right)^d\right)$ decision regions. Since

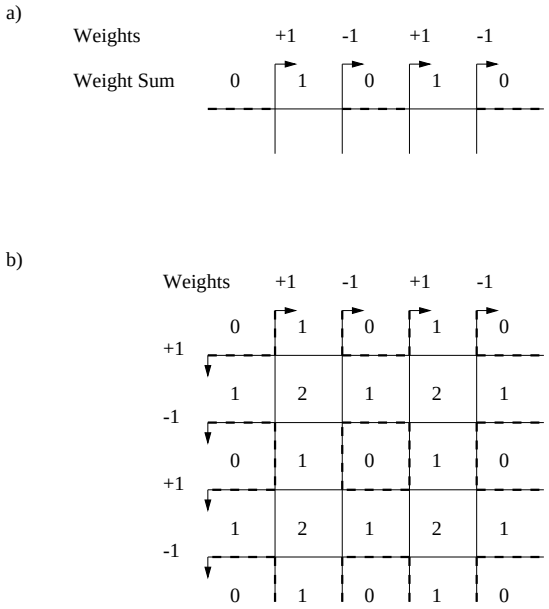

Figure 18. A hand constructed neural network which demonstrates the potential number of decision regions a network can have.

each decision region is a hypercube in d dimensions, it has 2^d vertices. Another point to note is that each hyperplane contributes a face in $\Omega\left(\left(\frac{n}{d}\right)^{d-1}\right)$ decision regions. This example uses parallel lines, the complexity is obviously higher with non-parallel lines in which all hyperplanes intersect with each other.

The result of this construction extends Golea's proof (Golea, 1996) of the NP-hardness of discrete network analysis to the continuous input case, by showing that it is necessary for the algorithm to be exponential as the complexity of the network may also be exponential in its decision regions structure. This complexity result addresses our first theoretical question as it applies not only to this algorithm, but to any algorithm which on some level tries to describe a neural network by enumerating its functions, i.e. listing its decision regions (e.g., rule extraction). Any algorithm that deals with network function on the level of decision regions will need space exponential in the input dimension to fully describe an arbitrary network.

6.2 Extensions to the Algorithm

The formulation of the DIBA algorithm given previously was for a three layer, single output network. Now we address on of the theoretical questions posed earlier: How does the complexity of the network and algorithm change when we allow more hidden layers and multiple outputs?

6.2.1 Additional Hidden Layers

Fundamentally, the addition of more layers to a perceptron neural network does not change the underlying possible locations and shapes of the decision regions. Consider adding an additional layer to our previous three-layer network. We already know that the units in the first hidden layer divided the input space into half-spaces, the units in the second hidden layer (previously the output units) form decision regions composed of the intersections of the divisions in the first layer, so what does the next layer do? Like the units in the second layer, the output values in the third hidden layer can only change across a boundary in the previous layer (another partial sum). And since the values in the second layer can only change across the boundaries of the first hidden layer's hyperplanes, then the boundaries of the third hidden layer are still only composed of the intersections of the first layer's hyperplanes. This argument holds for any additional layers. So in essence the first layer fundamentally defines what possible decision regions the network can express.

In terms of the algorithm itself, the modification is straight forward. The potential locations of the polytope vertices are still the same, we just need to adjust the corner test. Rather than doing the partial sum test for the smallest absolute valued weight, we perform the general corner test. We check for a transition across all the intersection hyperplanes at a vertex, but with respect to an output unit at a higher level. Specifically, we form all the possible hidden states at the vertex, feed them to the rest of the network, and see if they qualify as a corner boundary with respect to the output unit. That is, each hyperplane acts as a boundary at least once in the vicinity of the vertex. Thus, the algorithm complexity does not change with the introduction of multiple hidden layers.

6.2.2 Multiple Output Units

Multiple output units can be handled in much the same way as additional hidden layers, using the generalized corner test. Instead of checking for an output transition with respect to one output unit (either on or off), we can check for an output transition with respect to a combination of output units. For example we might be interested in extracting the decision regions described by having exactly six output units on. Then in order to check if a vertex forms part of such a decision region we check the value of all the output units with respect to the possible hidden states at the vertex. If there is a transition across all hyperplanes forming the intersection, that is, in the vicinity of the intersection across each hyperplane there is a location where the number of activated output units is six on one side and different from six on the other side, then it is a corner of the decision region. This can be applied to any output interpretation regiment that is applied to the output units. Therefore,

the algorithm complexity stays the same for different output interpretations.

6.3 Rule Extraction and Network Validation

Polytopic decision regions offer us a direct representation of the underlying neural network. However, high dimensional polytopes are not immediately fathomable to most people either. The interdependence in the polytopic case is limited to understanding the linear relationships governed by the vertices which delineate the faces.

As proposed earlier, one way to generate more comprehensible descriptions is to generate independent rules in the form of minimum bounding hyper-rectangles (MBR). That is, for each polytope we find the minimum hyper-rectangle which completely encloses it. This is a trivial operation, and it gives us a zeroth order approximation for the location and size of the network's decision regions.

We can improve this rule approximation to an arbitrary degree by dividing the polytopic decision regions into sub-polytopes. In the previous section we saw one approach to this, which was to systematically bisect the polytopes across its dimensions. This incrementally refines each rule into sub-rules each time it is applied.

It is feasible to imagine a more intelligent method to divide the polytopes. For example, we could only divide those polytopes for which the MBR is a bad approximation. It would seem that our geometric analogy would also give us the tools to exactly calculate the efficacy of the approximation, since all we would need to do is compare the volume of the hyper-rectangle with the volume of polytope it was enclosing to get an exact error measure. However, volume computation of n-dimensional polytopes is $\#P$-hard in the exact case (worse than NP-hard) and of high complexity for approximate cases (Khachiyan, 1993). This is a general property of the comparison of different models with perceptron neural networks, not just symbolic approximations–to get an exact error measure we have to compute the volume of the polytopic decision regions. This addresses another of the theoretical questions: The cost of validating the accuracy of alternative representations of network function is computationally hard.

6.4 Sigmoidal Activation Functions

Since the sigmoid is a smooth transition, it is not possible to model the underlying decision regions of a sigmoidal neural network using only vertices

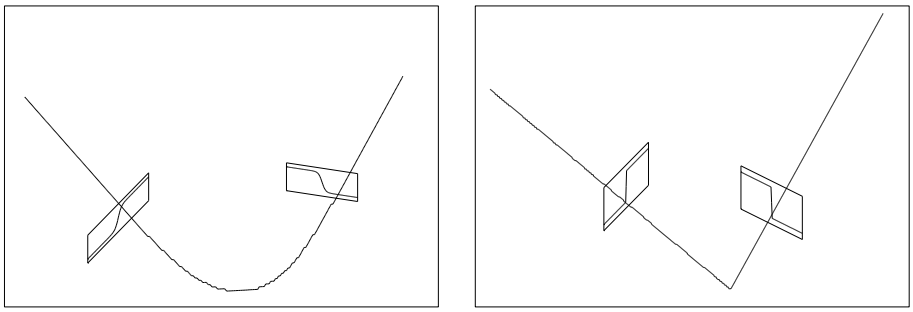

Figure 19. The activation function used in single-hidden-layer network affects the borders formed at hyperplane intersections. While for threshold activation functions (right) the border is a sharp corner whose position is at the exact threshold boundary location, for sigmoidal activation functions the corner becomes rounded and the border is shifted within the width of the sigmoidal transition.

and lines. Thus the basic representational type used by the DIBA algorithm, polytopes, can not capture a sigmoidal network's function exactly. However, by examining the effects that sigmoidal outputs have on the formation of decision boundaries we can get an idea of where the DIBA extracted polytopes reflect and differ from the actual smooth decision regions of a sigmoidal network.

Consider a single hidden-layer network with one output given by the following equation: $output = A(\sum_i o_i A(\overline{w_i x}) - c)$. If the activation function is threshold then as it is non-continuous its decision boundary is described by $\sum_i o_i T(\overline{w_i x}) \approx c$, the area in the input space where the sum passes above or below c. If the activation function is sigmoidal then we can define the boundary as $\sigma(\sum_i o_i \sigma(\overline{w_i x}) - c) = 0.5$, which can be rewritten as $\sum_i o_i \sigma(\overline{w_i x}) = c$.

If the magnitude of the weights, $\|\overline{w_i}\|$, is high then $\sigma(\overline{w_i x})$ approximates the threshold response, $T(\overline{w_i x})$, arbitrarily closely, implying a close fit between the threshold network's extracted decision regions and the sigmoidal network's. At the other end of the spectrum if the magnitude of the weights is significantly small (with respect to the relevant input space) then by Taylor expansion it can be shown that the sigmoidal function acts almost linearly. In such a case the DIBA algorithm will not give a good estimate of the decision boundaries, as now the boundary equation approximates a linear equation. Thus, the relevancy of an extracted representation is dependent on the interplay of these two polarities in sigmoid behavior as they address individual boundaries.

Consider first the case of an isolated intersection of hyperplanes. Figure 19 shows how different activation functions affect the boundary formed at the intersection of two hyperplanes. There are two main visible differences between threshold and sigmoidal: first, the corner changes to a curve that transitions between the directions of the hyperplanes; second, while the orientation of the hyperplane boundaries stay the same, their location shifts.

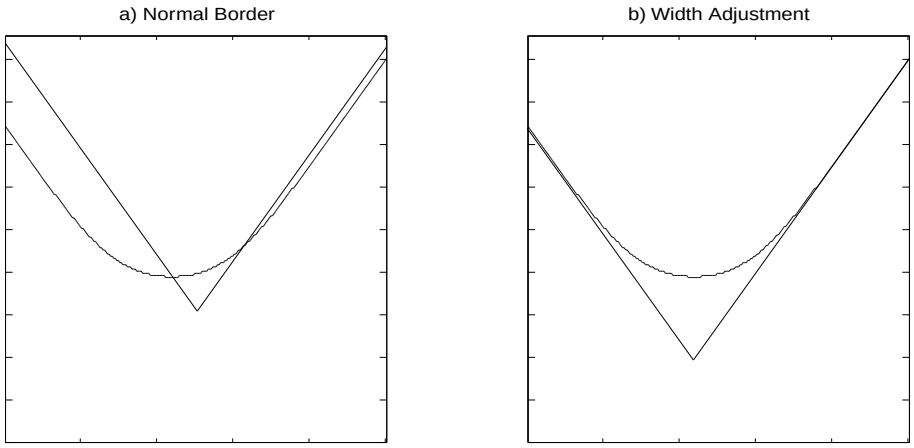

Figure 20. Approximating a sigmoidal corner with thresholds can be done in two ways: a) The regular DIBA approach considers the border as appearing at the center of the sigmoid. b) Compensating for the width effects allows the threshold boundary to be at a tangent to the sigmoidal boundary.

Unlike the threshold activation function unit hyperplanes, where an output transition is completely localized, taking place only where $\overline{w_i x} = 0$, the transitions in sigmoidal activation function units are gradual and can take place anywhere within the *width* (non-asymptotic region) of the sigmoidal hyperplane. How does the actual location shift? For hidden unit j, a hyperplane boundary exists when the boundary equation simplifies to $\sigma(\overline{w_j x}) + \sum_{i \in M \setminus j} o_i = c$, where M contains some of the units. That is, when we move sufficiently away from the widths of the other units, and allow their output to asymptotically converge to either 0 or 1, the boundary equation takes on the form of the hidden unit hyperplane. Thus the equation for the border becomes $\overline{w_j x} = \sigma^{-1}\left(c - \sum_{i \in M \setminus j} o_i\right)$. Contrasted with the location of the hyperplane in the threshold case, $\overline{w_i x} = 0$, implies that the shift in the hyperplane location is directly governed by the difference between the output unit constant, c, and the partial sum of output weights in that boundary location. A difference of 0.5 will not move the location, while other values will adjust it monotonically. From a practical perspective, we can use this measure to gauge how far off a threshold corner is from the boundary location in the sigmoid case. Given a corner, for each hyperplane at its intersection we can calculate its location offsets (there may be more than one if the dimensionality is greater than two.) What this tells us is how far the corner would have to be moved along the offset axis of each hidden unit hyperplane, for its threshold boundaries to be tangent to the actual sigmoidal boundary. Thus these values are an indication of the effect that the sigmoidal output has on shifting the basic location of the boundary. In figure 20 we see an example of how this compensation can be applied to a 2-dimensional corner, moving it to more closely match the sigmoidal effects.

Whereas for isolated intersections we can calculate the hyperplane shift di-

rectly, in the general case we also need to consider hidden units that are not part of the intersection. In the threshold case these units are either on or off, depending on their directionality with respect to the intersection. For the sigmoid case, while these projecting units are not at the center of their linear region (not part of the intersection) their behavior may not be sufficiently asymptotic to be described as on or off. This difference can effect by how much the output threshold is passed, relating to the previous discussion of hyperplane shifting, or in the more extreme case if the boundary exists or not. Thus, any potential confidence measure for the validity of a threshold boundary in a sigmoid context would need to consider how close to saturation these additional units were and how sensitive the actual boundary unit would be to these variations in the partial sum of output weights. When these effects combine to place the boundary in a region distant from the linear portion of its sigmoid, that is, it is a boundary by only a small margin, then the existence and location of the threshold boundary would be questionable.

To better account for these two regimes of the sigmoid, asymptotic and linear, a potential future enhancement of the algorithm would be to model the sigmoid by approximating them with piecewise linear units. Figure 21 contrasts a piecewise linear function with the sigmoid it is approximating, and in figure 22 we see the form of a typical intersection between two border piecewise linear hyperplanes. When multiple linear regions overlap they form a composite linear region. As the figure illustrates, the boundary in this region has the effect of filing down the edge or corner by defining a sub-face between the faces of the intersecting hyperplanes, approximating the sigmoidal units completely smooth boundary. The advantage here is that we are still using the same geometric representation of decision regions, high-dimensional polytopes. But we are coming closer to modeling the sigmoidal effects.

These potential enhancements to the algorithm are not without significant cost, as the intersection of multiple piecewise linear hyperplanes generates a potentially exponential number of different linear regions in its vicinity. Any of these regions could potentially house a border face. The test for whether a border passes through such a region is trivial (check the smallest and largest corners.) However, finding the exact face of the border is a hard problem (Khachiyan, 1993).

The previous discussion was referring to a single hidden layer network. The addition of more hidden layers may substantially change the possible underlying decision regions, depending on the extent of the sigmoidal effects. In the completely general case it becomes hard to make the connection to threshold units, as by the universal approximation properties of sigmoidal networks the additional layers could be sensitized to minute differences in the outputs of the lower layers, and thus hypothetically generate decision regions that are uncorrelated with the shape of the first layer hyperplanes. The threshold analogy

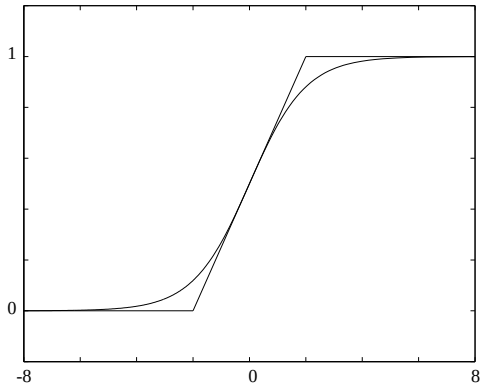

Figure 21. A sigmoidal activation function and its piecewise linear counterpart.

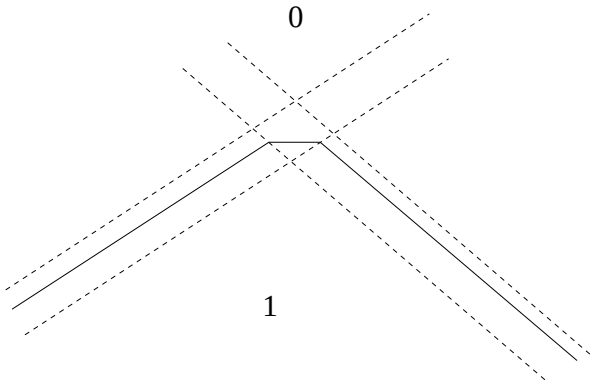

Figure 22. The intersection of two piecewise linear border hyperplanes forms an intermediate border face.

holds stronger when the additional layers have relatively large weights and they engender large margin responses at the borders.

6.5 Generalization and Learning (Proximity and Face Sharing)

Generalization is the ability of the network to correctly classify points in the input space for which it was not explicitly trained. In a semi-parametric model like a neural network, generalization is the ability to describe the correct output for groups of points without explicitly accounting for each point individually. Thus, the model must employ some underlying mechanisms to classify a large number of points from a smaller set of parameters.

In our feed-forward neural network model we can characterize the possible forms of generalization into two mechanisms. The first is by *proximity*: nearby points in the same decision region are classified the same way. The second is by *face sharing*: the same hyperplane is used as a face in either multiple decision regions or multiple times in a decision region. An analogy for this type of generalization would be Manhattan streets, where each street forms

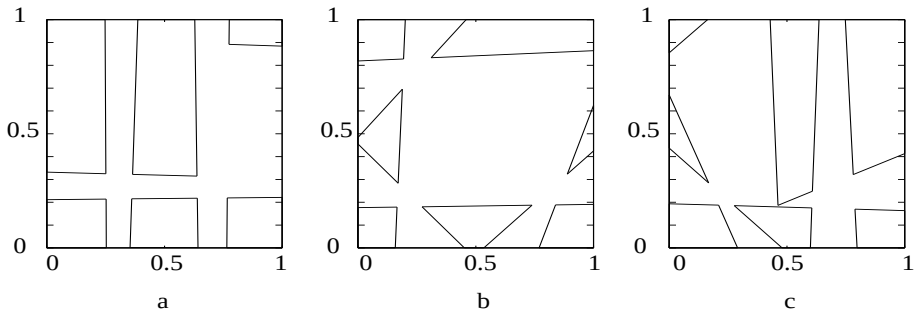

Figure 23. The decision regions of back-propagation networks trained on the decision regions of figure 18b.

the boundary of many different blocks.

Changing the network architecture does not radically affect the kind of generalization possible. Instead of hyperplanes another shape could be used or the non-linear activation functions could be modified in a locally continuous fashion. Fundamentally we would still have just two kinds of generalization, proximity and face sharing, in these types of feed-forward network architectures.

Given these two mechanisms, how well do learning algorithms exploit them? Proximity mandates the ability to enclose regions in space with similar outputs. It is intuitive that learning algorithms which pull borders (Hyperplanes in this case) towards similar output points and push borders away from different output points, should be geared to do some form of proximity generalization by forming decision regions around similar points. However, face sharing generalization is more combinatorial in nature, and might not be as amenable to continuous deformation of parameters as found in many algorithms.

To illustrate this point a group of neural networks with 8 hidden units were trained on the decision regions illustrated in figure 18b, a problem requiring face sharing to solve. One thousand data points were taken as a training sample. Two hundred different networks were used. Each was initialized with random weights in the range of -2 to 2, and trained for 300 online epochs of back-propagation with momentum. Of the 200 networks, none managed to learn the desired 9 decision regions. The network with the best performance generated only six decision regions (figure 23a). In examining its output unit's weights, we saw that only one weight changed sign, the weight initially closest to zero, indicating a predisposition to this configuration in the initial conditions with respect to the learning algorithm. Or stated more directly, the learning algorithm did not cause sufficient combinatorial modifications to facilitate face sharing. Figures 23b and c show the decision regions for some of the other, more successful networks.

This issue brings up some potential research questions. How do direct network

construction algorithms contrast or coexist with training algorithms with respect to the decision regions they generate? Can we employ geometric regularization in learning algorithms to generate decision regions with specific properties?

7 Conclusion

We started the paper by asking what factors affect the complexity of neural network representation extraction in general. We further asked, what can be learned about a network's success from its representation, especially in terms of generalization and artifacts? The tool we developed for the task is the Decision Intersection Boundary Algorithm. An algorithm that can be used to extract exact, concise and direct representations of the decision regions of threshold multi-layered perceptron networks.

The scope questions were addressed by analyzing multiple example networks to see where they introduce noise, when they generalize, and what forms their computation can take on. Using the examples of sphere networks at different dimensionality we explored the relationship between the input dimension, the location of the training data and the appearance of artifacts, clearly demonstrating that artifacts are not only not uncommon but become quickly prevalent in higher dimensional input spaces. We then explored generalization for three different networks by examining the properties of their decision regions, looking at: convexity, concavity, the quantity of decision regions, their location, and their orientation. In general, how the decision regions partition the input space was seen as an indicator of generalization. Over partitioning in the form of many or highly concave decision regions in relatively lower dimensional input spaces with dense training data, or overly simplified and convex regions in high-dimensional sparse cases, was seen as indicating suspicious generalization. In addition, for higher-dimensional spaces where the decision regions can not be directly visualized we explained how using hyper-rectangles and slicing the decision regions can be analyzed for these different properties at any desired resolution.

We started addressing the theory questions by analyzing the complexity of the algorithm, proving that even though the algorithm's complexity is exponential, it is unavoidable, since networks are capable of generating an exponential number of decision regions, where each decision region's complexity is also exponential. By explaining how the algorithm can be extended to additional hidden layers, and multiple output units, we showed that these modifications do not fundamentally impact the kind of processing the network is capable of, or their complexity, as the arrangement of hyperplanes in the first layer of hidden units fundamentally defines what possible decision regions the network

can express.

Next, we discussed the computational cost of finding the exact error between a network and any approximate representation, showing the cost to be exponential due to the complexity of volume calculations in high-dimensional spaces. Then, we explored the ramifications of examining networks with sigmoidal activation functions, showing where they are similar and different from their threshold counterpart. This lead to a discussion of what factors in the network weight structure impact the relationship between the sigmoidal and threshold boundaries.

We concluded by asking how well can learning algorithms use the two forms of generalization, proximity and face sharing. Is it possible for these networks to exploit all their potential complexity? Even though back-prop did not show promising results for face sharing, this remains an open question and a direction for further exploration of future learning and network construction algorithms.

Beyond our questions, the DIBA algorithm can also be used as an applied tool. As an exact, direct and concise representation extraction algorithm, the DIBA algorithm can easily be used to analyze reasonably sized threshold networks, [2] where knowing the exact representation is essential to validate them for real-world deployment. Or, using the concepts and methods presented in this paper, it can be used to shed light on how any such network generalizes. DIBA can also be used to study learning, in the simple case to visualize decision regions changing during learning, or as we have done in other experiments, to explore how well learning algorithms form specific decision regions. For examples and source code implementing the basic algorithm, please look at http://www.demo.cs.brandeis.edu/pr/DIBA.

References

Andrews, R., Cable, R., Diederich, J., Geva, S., Golea, M., Hayward, R., Ho-Stuart, C., and Tickle, A. (1995). An evaluation and comparison of techniques for extracting and refining rules from artifical neural networks. *Knowledge-Based Systems Journal*, 8(6).

Bishop, C. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.

Bologna, G. (1998). Symbolic rule extraction from the dimlp neural network. In Wermter, S. and Sun, R., editors, *Hybrid Neural Systems*, pages 240–254. Springer-Verlag.

[2] A five dimensional, 100 hidden unit network takes less than 10 seconds to analyze on a Pentium II.

Campbell, J., Andrew, W., and Mackinlay, A. (1997). *The Econometrics of Financial Markets.* Princeton University Press.

Craven, M. and Shavlik, J. (1995). Extracting comprehensible concept representations from trained neural networks. In *IJCAI 95 Workshop on Comprehensibilty in Machine Learning.*

Edelsbrunner, H. (1987). *Algorithms in Combinatorial Geometry.* Springer-Verlag.

Golea, M. (1996). On the complexity of rule-extraction from neural networks and network-querying. In *Proceedings of the Rule Extraction from Trained Artificial Neu ral Networks Workshop, AISB'96*, pages 51–59.

Gorman, R. and Sejnowski, T. (1988). Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75–89.

Intrator, N. and Intrator, O. (1993). Interpreting neural-network models. In *Proceedings of the 10th Israeli Conference on AICV*, pages 257–264. Elsevier.

Intrator, O. and Intrator, N. (1997). Robust interpretation of neural-network models. In *Proceedings of the VI International Workshop on Artificial Intelligence and Statistics.*

Khachiyan, L. (1993). Complexity of polytope volume computation. In *New Trends in Discrete and Computational Geometry*, chapter 4. Springer-Verlag.

Linden, A. (1997). *Iterative Inversion of Neural Networks and its Applications*, chapter B5.2. Oxford University Press.

Maire, F. (1999). Rule-extraction by backpropagation of polyhedra. *Neural Networks*, pages 717–725.

Makhoul, J. (1975). *Linear Prediction in Automatic Speech Recognition*, chapter 2. Academic Press Inc., New York, New York.

Pratt, L. and Christensen, A. (1994). Relaxing the hyperplane assumption in the analysis and modification of back-propagation neural networks. In *Cybernetics and Systems 94*, pages 1711–1718. World Scientific.

Sanger, D. (1989). Contribution analysis: A technique for assigning responsibilities to hidden units in connectionist networks. *Connection Science*, 1:115–138.

Schrijver, A. (1987). *Theory of Linear and Integer Programming.* Wiley-Interscience.

Setino, R. (1997). Extracting rules from neural networks by pruning and hidden-unit splitting. *Neural Computation*, 9(1):205–225.

Sharkey, N. and Sharkey, A. (1993). Adaptive generalization. *Artificial Intelligence Review*, 7:313–328.

Shultz, T., Oshina-Takane, Y., and Takane, Y. (1995). Analysis of unstandardized contributions in cross connected networks. In *Advances in Neural Information Processing Systems 7.* MIT Press.

Thrun, S. (1993). Extracting provably correct rules from artificial neural networks. Technical report, University of Bonn.

Appendix A: Pseudo Code of Traversing the Line

In this appendix a pseudo code description of the Traversing the Line portion of the Decision Intersection Boundary Algorithm (DIBA) is outlined. It is the termination step of the recursion described in the Generative Recursion portion of the algorithm.

It calculates the contribution of the different hyperplanes to each potential line segment, and then finds the corners and actual line segments.

else { *hidden* hyperplane dimension ≤ 1 }
    sort the *hidden* by their position on the line

Forward pass.

```
running_sum := 0
clear array segments
current_segment := 1
for hyperplane = leftmost hyperplane to rightmost hyperplane
    if hyperplane not a border then
        if hyperplane is forward directed then
            Add weight out(hyperplane) to running_sum
        endif
        segment(current_segment) := running_sum
        current_segment++
    endif
endfor
```

Backward pass.

```
running_sum := 0
in_line := false
current segment--
for hyperplane = rightmost hyperplane to leftmost hyperplane
    if beginning of border delimited region and check_line(current_segment) then
        in_line := true
        Add the node to the representation
    elseif hyperplane not a border then
        current_segment--
        if hyperplane is backward directed then
            Add weight out(hyperplane) to running_sum
        endif
        segment(current_segment) := segment(current_segment) + running_sum
        if in the border delimited region then
```

```
                    new_in_line := check_line(current_segment)
                    if in_line and new_in_line then
                        if check_corner(current_segment, out(hyperplane)) then
                            Add the vertex to the representation
                            Connect current vertex to last corner
                        endif
                    elseif in_line and not new_in_line then
                        Add the vertex to the representation
                        Connect current vertex to last corner
                    elseif not in_line and new_in_line then
                        Add the vertex to the representation
                    endif
                endif (in border region)
            endif (not border)
        endfor
        if in_line then
            Add the vertex to the representation
            Connect current vertex to last corner
        endif
endif
```

Appendix B: Proof of partial sum corner test

For a 3-layer network with one output unit (partial sum case) it is sufficient to test whether the hidden unit with smallest absolute valued weight has a boundary in the intersection. If it does, then all the other hyperplanes which make up the intersection also have boundaries.

Let $W = \{w_1 \ldots w_n\}$ be the set of partial weights corresponding to the hyperplanes making up the intersection. Define $S \subseteq W$, as a hidden state. Let T be the threshold, such that if $\sum_{w \in S} w \geq T$ then the output value is 1, otherwise the output value is 0.

Assume that there exists a w_m, such that for all $w \in W$, $|w_m| \leq |w|$, and there exists a hidden state, S, which does not contain w_m, such that if $w_m > 0$ then $\sum_{w \in S} w < T$ and $w_m + \sum_{w \in S} w \geq T$, or if $w_m < 0$ then $\sum_{w \in S} w \geq T$ and $w_m + \sum_{w \in S} w < T$.

If $\alpha \in W$, and α is not w_m, we need to demonstrate that there exists a hidden state such that α acts as a boundary.

- If $w_m > 0$, $\alpha > 0$ and $\alpha \notin S$, then $\sum_{w \in S} w < T$ and $\sum_{w \in S} w + \alpha \geq T$.

- If $w_m > 0$, $\alpha > 0$ and $w \in S$, then $\sum_{w \in S} w + w_m \geq T$ and $\sum_{w \in S} w + w_m - \alpha < T$.
- If $w_m > 0$, $\alpha < 0$ and $\alpha \notin S$, then $\sum_{w \in S} w + w_m \geq T$ and $\sum_{w \in S} w + w_m + \alpha < T$.
- If $w_m > 0$, $\alpha < 0$ and $\alpha \in S$, then $\sum_{w \in S} w < T$ and $\sum_{w \in S} w - \alpha \geq T$.
- If $w_m < 0$, $\alpha > 0$ and $\alpha \notin S$, then $\sum_{w \in S} w + w_m < T$ and $\sum_{w \in S} w + w_m + \alpha \geq T$.
- If $w_m < 0$, $\alpha > 0$ and $\alpha \in S$, then $\sum_{w \in S} w \geq T$ and $\sum_{w \in S} w - \alpha < T$.
- If $w_m < 0$, $\alpha < 0$ and $\alpha \notin S$, then $\sum_{w \in S} w \geq T$ and $\sum_{w \in S} w + \alpha < T$.
- If $w_m < 0$, $\alpha < 0$ and $\alpha \in S$, then $\sum_{w \in S} w + w_m < T$ and $\sum_{w \in S} w + w_m - \alpha \geq T$.

Therefore all hyperplanes have a boundary in the vicinity of the intersection.