Exact Representations from Feed-Forward Networks

Ofer Melnik and Jordan Pollack, Volen Center for Complex Systems
Brandeis University, Waltham, MA, USA
melnik@cs.brandeis.edu *pollack@cs.brandeis.edu*

We present an algorithm to extract representations from multiple hidden layer, multiple output feed-forward perceptron threshold networks. The representation is based on polytopic decision regions in the input space– and is exact not an approximation like most other network analysis methods. Multiple examples show some of the knowledge that can be extracted from networks by using this algorithm, including the geometrical form of artifacts and bad generalization. We compare threshold and sigmoidal networks with respect to the expressiveness of their decision regions, and also prove lower bounds for any algorithm which extracts decision regions from arbitrary neural networks.

Introduction

Despite being prolific models, feed-forward neural networks have been ubiquitously criticized for having a low degree of comprehensibility. This is mostly due to the *distributed* representation of information within a neural network. What we seek is 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. There have been many different approaches to neural network analysis. The *rule extraction* [1, 2, 10] approaches do usually generate an independent representation, but it is an approximation and in many instances not based on a direct analysis of the network parameters, rather the network is used as an oracle. *Weight-state clustering* [4], *contribution analysis* [8], and *sensitivity analysis* [5], do use the network's parameters in their analysis. However, they do not generate an alternative representation, instead 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. *Network inversion* [7] is a technique where locations in the input are sought which generate a specific output. Maire's recent work is promising in that it approaches the problem by back-propagating polyhedra through the network. Its main shortcoming is its lack of conciseness, because each stage of inversion can generate an exponential number sub-polyhedra.

The motivation behind the approach to network analysis taken in this paper is to examine neural computation from first principles– to arrive at the underlying constraints of how the network output can change under differing input conditions. Using these constraints leads to a direct, exact and concise representation of network function, as well as to an algorithm to extract the representation from feed-forward threshold activation function networks.

The output of a network partitions the space of its inputs into separate decision regions. For each possible network output value, there exists a corresponding region in the input space such that all points in that region give the same network output.

The Decision Intersection Boundary Algorithm (DIBA) presented in this paper, is designed to extract decision regions from a multi-layer perceptron network, 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 principle observations about how multi-layer perceptrons compute: 1) The outputs of a network are independent of each other. 2) Since the output values of hidden units are 0 or 1, the output units just compute a partial sums of their weights. 3) Output unit values change only when hidden unit values change. 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 polytopes which can be described by the vertices which delineate them.

The rest of this paper is organized as follows: First, we describe the two parts which constitute the basic DIBA algorithm. Then, the algorithm is used to analyze three example networks, showing different properties of their decision regions. We follow with a discussion of algorithm extensions, the effects of sigmoidal activation functions, and algorithm and network complexity, showing that for exact representaion extraction the DIBA algorithm performs on par with any other potential algorithm.

The Decision Intersection Boundary Algorithm

The Decision Intersection Boundary Algorithm is designed to extract the polytopic decision regions of a single output of a three layer perceptron network. Its 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. The algorithm consists of two parts, a part which generates all the possible vertices and connecting line segments, and a part which evaluates whether these basic elements form the boundaries of decision regions.

Recursion

The generative part of the algorithm is recursive. For each hidden unit hyperplane of dimension d, the algorithm projects on to it all the other hyperplanes. These projections are hyperplanes of dimension $d-1$. This procedure is performed recursively until one dimensional hyperplanes (lines) are reached. Lines are the basic unit of boundary evaluation in this algorithm.

The Boundary Test and Additional Units

Along the line, the locations of output unit value changes are at the intersections with the remaining lines. 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 an d-dimensional input space. 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. Abstractly, this boundary represents the location in input space where we have an output value transition that can be described by a hyperplane. Taking this view in general, a boundary is composed of various geometric entities which demarcate output value transitions in input space. For an intersection of hyperplanes (a lower dimensional geometric entity) to be a boundary, what we seek is that all hyperplanes making up the intersection have at least one face that is a boundary in the vicinity of the intersection. Note that the boundary test for line segments and corners is the same, since both lines and vertices are just intersections of hyperplanes. Lines are the intersection of n-1 hyperplanes and vertices are the intersection of n hyperplanes.

How do we in practice check for this intersection boundary condition? In an d-dimensional input space, an intersection of $n \leq d$ hyperplanes partitions the space into 2^n regions. 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 for these hidden units is represented 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, which 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.

For the intersection boundary test there are two kinds of hidden unit hyperplanes: 1) hyperplanes that make up the intersection and 2) hyperplanes that do not, but do affect the output unit partial sum value. 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. On the line there is an economical solution to incrementally quantify the contribution of these additional units to the partial sum, by performing two passes on the line.

As described, the DIBA algorithm is composed of a main recursion over the dimensions of the hyperplanes and a boundary test to evaluate which intesections form actual parts of decision regions. In the next section we demonstrate how the algorithm can be put to practice on some example neural networks.

Example 1, The Circle

The DIBA algorithm was used to analyze an **80** hidden unit sigmoidal neural network that was trained using back-propagation to classify points inside and outside of a circle of radius 1 around the origin, by treating the activation functions as thresholds. In figure 1 we see the decision region that DIBA extracted around the origin, and the points corresponding to the training sample. The decision region is a direct view of what the network does– exactly where it succeeds and where it fails. The representation is concise, because all the vertices used are necessary to completely describe the decision region. With this exact representation we 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.

With the DIBA algorithm we can 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 2 we see large artifactual decision regions at a distance of at least 50 from the decision region around the origin formed in an area where there was no training data.

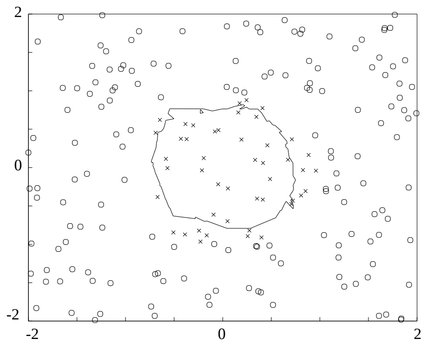

Figure 1: A decision boundary of a network which classifies points inside (x) and outside of a circle (o).

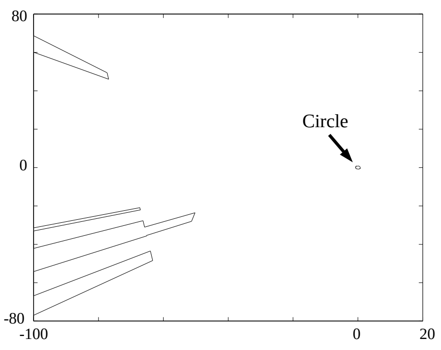

Figure 2: A zoom out of the network shows the existence of large artifactual decision regions.

Example 2, The Sphere

A **100** hidden unit network was trained to differentiate between points inside and outside of a sphere centered at the origin. The left image in figure 3 shows the decision region extracted with DIBA encapsulating the network's internal representation of the sphere. The right image in the figure shows the same artifacting phenomena that we saw in the circle– the miniature sphere appears amid a backdrop of an ominous giant cliff face. Note how the increased dimensionality between the 2-D circle and 3-D sphere introduces much more complex artifacts.

Example 3, Predicting the S&P 500 Movement

A neural network with 40 hidden units was trained using heuristic driven hill-climbing 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 macroeconomic data from 1953 to 1994. 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. After training, the network achieved a better than 80% success rate on the data. Figure 4 shows the network's decision regions that were extracted using DIBA. Immediately we can surmise that the network would probably not generalize very well to out of sample data. We can see that rather than extracting some underlying regularity, the network tended to highly partition the space in an attempt to enclose all the inconsistent data points.

How can we quantitatively describe this intuition about the network? 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 rectangle's coordinates. The rectangles can later be refined to enclose parts of polytopes, thereby giving higher resolution rules. We can use this rule method to quantify 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

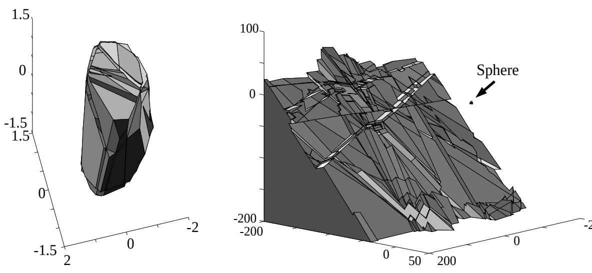

Figure 3: The decision region of a network which classifies points inside a sphere, and the artifacts around the sphere decision boundary.

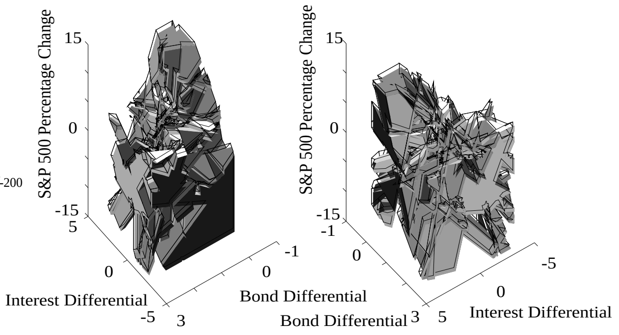

Figure 4: Two views of the decision regions of the S&P 500 prediction network.

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. 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. 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.

This hyper-rectangle slicing rule method can be used to analyze higher-dimensional spaces where we can not directly vizualize the decision regions. By allowing us to enumerate, localize and decompose decision regions, this method can exactly guage the geometry and topology of how the space is partitioned and covered by the decision regions, there by giving a real assessment of what a network has learned in higher dimensional spaces. Unfortunately, due to space limitations in this article, examples of this can not shown.

Discussion

Additional Hidden Layers and Multiple Output Units

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. 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). 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 hidden layer fundamentally defines what possible decision regions the network can express.

In terms of the algorithm, the modification for additional layers is straight forward. The potential locations of the polytope vertices are still the same, we just need to adjust the boundary 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.

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.

Algorithm and Network Complexity

The DIBA 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 [3], 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 simpler partial sum case (singel hidden layer, one output) is equivalent to the *knapsack* problem [9] which is NP-complete.

Do we really need to examine every vertex though? Perhaps networks can only use a small number of decision regions, or it are limited with respect to the complexity of the decision regions. Unfortunately, that is not the case, because we can hand construct a network with $\Omega((n/d)^d)$ different decision regions, where each decision region has 2^d vertices.

We begin with a one dimensional construction of a three layer perceptron network. Start with k hidden units with the same directionality, and weight them with alternating $+1$ and -1 weights. In figure 7a 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 zero weight sum.

We can 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 weight them with alternating -1 and $+1$ weights. Figure 7b 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 $\Omega((n/d)^d)$ decision regions. Since 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 $\Omega((n/d)^{d-1})$ decision regions. This example uses parallel lines, the complexity is obviously higher with non-parallel lines in which all hyperplanes intersect with each other. This complexity result applies not only to this algorithm, but to any algorithm which on some level tries to describe a neural network by enumerating its functions (e.g., rule extraction). That is, any such algorithm will need exponential space to fully describe an arbitrary network.

Sigmoidal Activation Functions

It is not possible to model the underlying decision regions of a sigmoidal neural network using only vertices and lines, since the sigmoid is a smooth transition. However we can get a good idea of the implications of this form of nonlinearity by modeling the units with piecewise linear units (see figure 5.) Unlike the threshold activation function unit hyperplanes, where an output transition is completely localized, the transitions in piecewise or sigmoidal activation function units are gradual and take place over the *width* of the hyperplane. In figure 6 we see the form of a typical intersection between two border piecewise linear hyperplanes. Notice how each hyperplane resides within the width of a linear region. The actual location of the boundary falls somewhere within that region– wherever the output unit exactly passes its decision threshold. When multiple linear regions overlap they form a composite linear region. As the figure illustrates, the boundary in this region has the effect of smoothing out the edge or corner by defining a subface between the faces of the intersecting hyperplanes. A sigmoidal unit would generate a smoother transition.

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 also a hard problem [6].

The addition of more hidden layers still does not substantially change the possible underlying decision regions. The output of any higher level unit is dependent on the values of the lower level units. This means that a transition in the value of a higher level unit must accompany a transition in a lower level unit. Therefore, the decision regions can only be within the width of the first layer's hyperplanes, where all basic transitions take place.

Conclusion

We have demonstrated how the Decision Intersection Boundary Algorithm (DIBA) can be used to extract exact, concise representations of multi-layered perceptron networks. Multiple example networks were analyzed using the network, illustrating the kind of information which can be elucidated from DIBA's decision

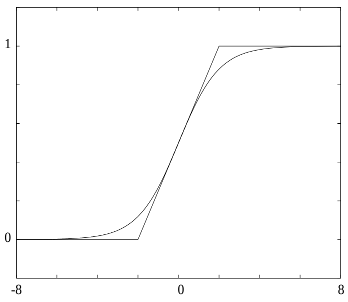

Figure 5: A sigmoidal activation function and its piecewise linear counterpart.

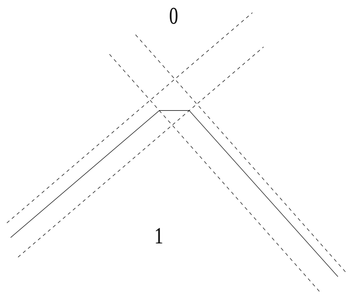

Figure 6: The intersection of two piecewise linear border hyperplanes forms an intermediate border face.

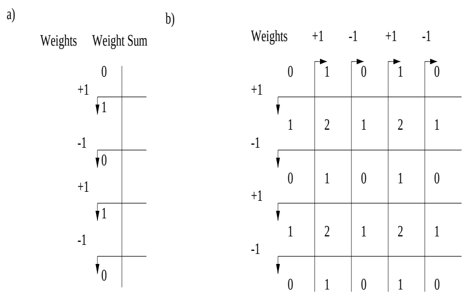

Figure 7: A hand constructed neural network which demonstrates the potential number of decision regions a network can have.

region representations (e.g., relationships between shape and training data; artifacts and high-complexity in areas where training data exists and does not exist; the relationship between convexity, concavity and how the space is partitioned, etc). Even 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 at any desired resolution. In the discussion we explained how the algorithm can be extended to additional hidden layers, and multiple output units, as well as the ramifications with respect to the type of decision regions generated by switching the activation function to a sigmoid. We concluded by analyzing the complexity of the algorithm, proving that even though the algorithm's complexity is exponential, there is an exponential lower bound on **any** algorithm that enumerates all the decision regions of arbitrary neural networks.

The DIBA algorithm can easily be used for reasonably sized networks, (a five dimensional, 100 hidden unit network takes less than 10 seconds to analyze on a pentium II) where knowing the exact representation is essential to validation, as well as shedding light on how any 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 decsion regions. For examples, please look at http://www.demo.cs.brandeis.edu/pr/DIBA. For networks with a large number of inputs, the potential exponential network complexity may overwhelm its utility. However, the obvious disparity between the amount of data used to train networks and their potential complexity suggests alternative approximate approaches which we demonstrate in another paper submitted to this conference.

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

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

[3] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, 1987.

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

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

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

[7] F. Maire. Rule-extraction by backpropagation of polyhedra. *To Appear in Neural Networks.*

[8] D. Sanger. Contribution analysis: A technique for assigning responsibilties to hidden units in connectionist networks. *Connection Science*, 1:115–138, 1989.

[9] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley-Interscience, 1987.

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