Geometry-Based Learning Algorithms

George H. John
Computer Science Department
Stanford University
Stanford, CA 94305
gjohn@cs.Stanford.EDU

UNPUBLISHED DRAFT – NOT FOR CITATION – COMMENTS WELCOME

March 1993

Abstract

We present CHILS , the Convex Hull Inductive Learning System, a novel supervised learning algorithm based on approximating concepts with sets of convex hulls. We introduce a theoretical methodology for describing the power of a concept representation language and use it to compare convex hulls with other geometrical concept representations. The Domain Transform framework (DT) provides a clear way to compare the power of supervised learning systems, allowing us to characterize a class of domains which is learnable by some systems but cannot be learned by other systems. DT can be used similarly to compare the expected generalization performance of different domains.

1 Introduction

When performing studies of supervised machine learning algorithms, and when designing new algorithms in particular, it is important to keep in mind the distinction between the learning element and the performance element. Regardless of the intricacies of the learning element, mathematically the performance element is just a function that maps elements of the domain to elements of the class label set. What does this function look like when the system is trained on a given dataset? What can the function look like in general? This function (commonly called a discriminant function or a decision surface) defines precisely what the system has learned during the training phase. To characterize the form of the decision surface is to define precisely what kinds of domains can and cannot be learned. Yet, surprisingly little regard is given to characterizing decision surfaces in machine learning literature.

The primary contribution of this paper is a simple methodology for describing the power of a learning algorithm by characterizing the properties of its decision surfaces. We introduce the idea of the invariance of a decision surface under certain kinds of topological transformations of the entire domain. We view learning algorithms as performing topological transformations on the domain, hence we call this framework DT for Domain Transformation. By characterizing the performance of a learning system on some canonical domain and proving that the performance of the system does not vary under certain kinds of topological transformations, we are able to make statements about the theoretical power of the learning algorithm. Similarly, by characterizing a domain in terms of a set of topological transformations on a canonical domain, we are then able to make statements about which algorithms can learn the domain, and how difficult it will be to learn.

In this paper we describe several geometry-based learning algorithms whose decision surfaces yield themselves to simple analysis. The first algorithm discussed, CHILS (Convex Hull Inductive Learning System), is a novel but simple learning algorithm whose development brought about the ideas that led to this paper. We wish to emphasize that we describe only a very simple version of the algorithm here, and are not advertising the described version as a serious learning algorithm. The intent is merely to give an idea of how convex hulls might be learned. The primary contribution of the algorithm is a new representational bias that can be thought of as combining the bias of a nearest-neighbor classifier with a separating hyperplane bias. A nice features is that sets of hulls can approximate arbitrary shapes. We discuss theoretical advantages of this approach in section 4.

2 Geometry-Based Learning

All supervised induction algorithms can be thought of in terms of the geometrical properties of their decision surfaces. By Geometry-Based Learning we mean to identify the set of learning systems whose learning element directly modifies and reshapes the decision surface. (We restrict our discussion to continuous domains with discrete class labels, but since there is a trivial way to map nominal attributes into continuous space, this should not be seen as a shortcoming.)

This section outlines some simple methods for employing basic geometrical constructs to learn decision surfaces. The point of this section is not to present cutting-edge learning algorithms. Rather, in order to talk about geometric concept description languages we felt it important to convince the reader that a plausible induction algorithm exists for each. CHILS is a novel system we have developed. The rest are simplified versions of real classifiers that have been proposed elsewhere. The goal of this section is not to describe every algorithm in detail, but rather to outline an algorithm by which the geometrical construct can be used to learn decision surfaces. In this study we are more concerned with the representational power of the concept-space the algorithm searches than the algorithm for finding the correct concept or decision surface. The power of these representations will be discussed in the next section on DT.

2.1 The CHILS Algorithm

The CHILS learning algorithm is quite simple. The learning element identifies decision surfaces by convex hulls. Recall that the two dimensional convex hull of a set of points is a set of line segments that bound the entire set of points. (Glance ahead at Figure 1 for an example.) Strictly speaking, the n dimensional version of this is called a convex polytope, but we will continue to call them hulls and discuss our algorithm in the 2-D case since the extension of our algorithm to d dimensions is straightforward. The hull defines a set of linear inequalities, all of which hold for all points inside the hull. (Each inequality defines a half-space, and the intersection of all the half- spaces is the set of points inside the hull. By definition, this intersection is bounded.)

The algorithm we present is incremental. The reasons for an incremental algorithm are partly historical (the original domain that inspired this work was a robot learning an action model where incremental learning was required), and partly for convenience (we avoid searching over the space of all possible decision surfaces and instead make greedy choices each time we receive a new training instance). The algorithm is semi-incremental in that it stores all observed training instances.

TRAINING
1. Receive training instance P and class label C.
2. If P is inside some hull H from some class other than C, break H so that it does not include P.
3. Extend all class C hulls H_i to cover P, call the new hulls H_i'.
4. For all hulls H_i', if H_i' now overlaps a hull from some class other than C, throw out H_i'; otherwise replace H_i with H_i'.

CLASSIFYING
1. If P is within some hull H then output H's associated class label.
2. Otherwise, output "?".

Table 1: Training and classification algorithms for CHILS .

Table 1 shows the training and classification algorithms for CHILS , which are quite simple. Much work has been done in the field of computational geometry to find efficient algorithms for finding convex hulls in high-dimensional spaces. Preparata in [7] discusses efficient ways to dynamically repair 2D hulls. Our incremental algorithm for learning hulls is shown in Table 1. The algorithms for extending and breaking hulls are described below.

To incorporate a new point P into an existing hull, we simply remove all faces which disagree with the new point, and then repair the hull to include the new point. Removing faces is easy – each face is a function which is 1 inside the hull and 0 outside, so we just remove faces for which $f(P) = 0$. In d dimensions, this results in d points now being "open" (*i.e.*, having $d - 1$ instead of d connected faces) and we simply close the hull by adding d new faces, where each face passes through P and $d - 1$ open points. If, while extending the hull to cover P we find that we've covered points from a different class, we leave P unincorporated and leave the hull unchanged, and may use P to form a new hull if any later training instances are near it.

Our algorithm for breaking a hull (when the hull misclassifies a training instance P' from a different class) is to find the hyperplane through P' that intersects the hull with minimal surface area[1], divide the points in the hull into 2 sets using this hyperplane, and form 2 new convex hulls that do not include P', using an algorithm similar to the one above. In this case, we end up with 2 open hulls. Considering one hull, it has d open points, which we close by simply connecting them all together. Some points formerly inside the hull may now be "orphaned" (*i.e.*, outside the newly repaired hull). We re-incorporate them using the above algorithm for hull extension.

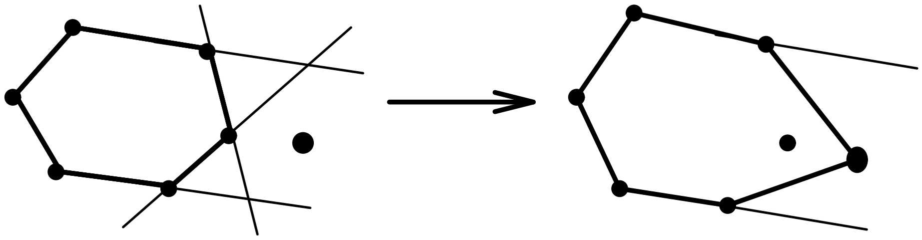

Figure 1: Incorporating a new instance Extending a hull to enclose a new training point outside the hull. Hull faces which misclassify the point are removed, and the hull is then repaired with the new point as a vertex.

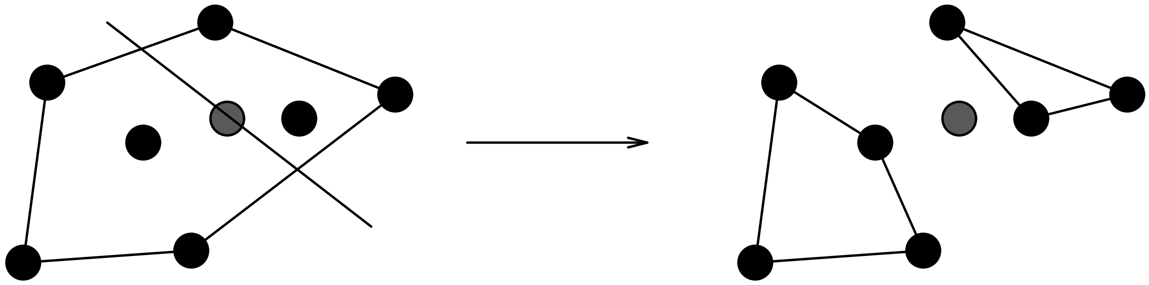

Figure 2: Splitting a Hull Breaking a hull to exclude a new training point from a different class. The algorithm requires storage of all training examples so that the two new hulls can be repaired properly.

This algorithm and the next two algorithms are restricted to learning closed sets as concepts. Since the decision surfaces are the boundaries of the sets, this means that there will be points which the classifier does not associate with any category. (See Figure 3. The area between the hulls is not associated with any category. The system would respond "?" when asked to classify any point not inside some hull.) Again, we stress that this is only a simple algorithm, clearly many different strategies might be used to guess the correct class of a point non in any hull, perhaps based on the proximity and size of close hulls, for example.

Our approach is similar to the CAT-AE algorithm developed by Gross [3], except that our algorithm directly learns hyperplanes not parallel to any axis. CAT is restricted to learning hyperplanes parallel to the axes of the instance space, and must use "attribute evolution" to create new attributes which are linear combinations of the original attributes so that it can effectively learn convex hulls. (Oddly, Gross begins his thesis with a discussion of why convex hulls aren't a desirable representation.)

Figure 3 shows how convex hulls capture the biases of nearest-neighbor methods and hyperplane-methods. When few points have been observed, hulls tend to "assume" that local hyperplanes are the right decision surface. After many points have been observed, hulls approach equivalence with nearest-neighbor methods.

A variation of CHILS , Counterfactual-CHILS (CCHILS) allows hulls from different classes to overlap, provided that some ordering exists on the hulls in the overlapping region. (This can be thought of as an extension of Rivest's Decision Lists [8] to continuous domains.)

[1] Various approximations to this are quite fast. For example, one can simply sample many hyperplanes randomly and pick the one with minimal surface area.

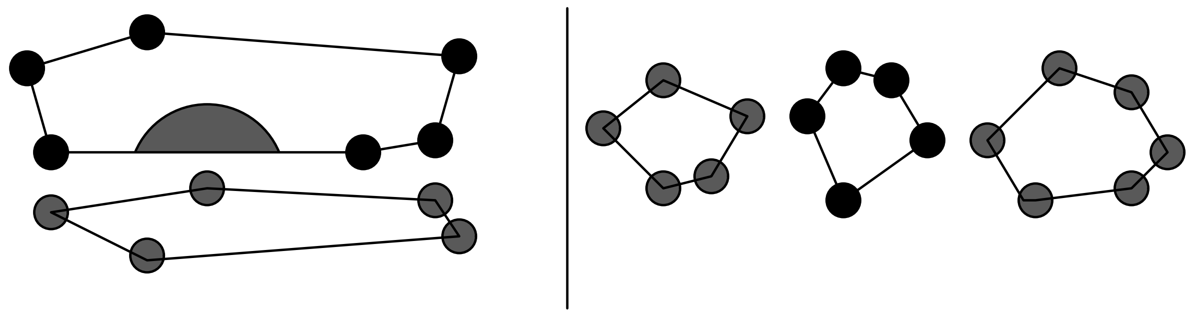

Figure 3: Convex Hull bias. The left half shows what we call a "hyperplane" bias. The gray region would be classified as white by the hypersphere or nearest-neighbor algorithms discussed later, but is classified as black by the convex hull. The right half shows that convex hulls can also have a kind of "nearest-neighbor" bias when the data warrants – in this case all points categorized as black have a black point as their nearest neighbor.

2.2 Learning with Hyper-Rectangles

The RECT learning system is a simplification of CAT [3]. The algorithm is exactly like CHILS , except that concepts are represented as hyperrectangles. (For example, a cube is a special 3D hyperrectangle.) The training algorithm is the same as before. "Breaking" a rectangle (when it includes a point P that it shouldn't) is done by choosing a hyperplane to split the rectangle. This hyperplane must be parallel to an axis of instance space in order to result in 2 rectangles that don't include P. Thus, since the hyperplane passes through P and is parallel to one of the d axes, there are only d choices for splitting. We choose the split that maximizes the lost volume. (This should make intuitive sense – if there were really two clusters of points at opposite ends of the rectangle, this procedure will break them into two separate rectangles.)

Salzberg [10, 9] has also investigated hyper rectangle-based classifiers and found them to perform as well or better than other learning algorithms on three datasets.

2.3 Learning with Hyper-Spheres

The SPHERE learning system borrows ideas from what are today called Restricted Coulomb Energy network classifiers, but were introduced as Hyperspherical Classifiers in the 1960's. See [4] for a brief overview and bibliography. Our SPHERE learning algorithm is again much like CHILS , except that concepts are represented as spheres. When joining points together to form a sphere, pick the smallest enclosing sphere. To break a sphere, pick the partitioning of points that results in the smallest total volume of the 2 new spheres.

2.4 Nearest-Neighbor Learning: Learning Voronoi Partitions

The VORONOI learning system is simply a nearest-neighbor classifier [1]. The decision surface is defined by the Voronoi diagram of the points. Figure 4 shows an example of a set of points and the corresponding Voronoi decision surface.

2.5 Learning with Hyper-Planes

The PLANE learning system comes from Nilsson [6, Sec. 4.3]. The system attempts to find a hyperplane separating the instances in one class from instances in the other. It is a parametric learning method: the concept description is a half-space, and the decision surface is characterized by one hyperplane. The position and rotation of the plane are parameters that the learning mechanism adjusts in an attempt to separate the domain. Domains that can be separated by hyperplanes are called linearly separable. Many different training algorithms have been proposed, see for example Widrow's review of the ADALINE [12].

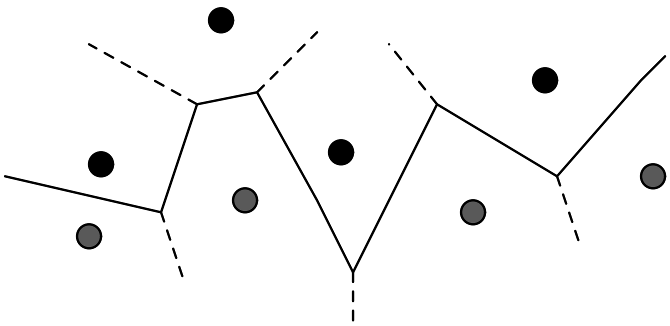

Figure 4: Voronoi tessellation of a set of points in Euclidean 2-space.

2.6 Learning with Neural Nets and Backpropagation

Rummelhart and McClelland [5] introduce an algorithm for training a multi-layer neural network. Essentially, a neural network consists of an input layer, which holds the attributes of the instance currently being presented, an internal layer, which represents a set of hyperplanes and calculates the distance from the hyperplanes to the points, usually scaled by some sigmoid function, and an output layer, which assigns a class to the point based on its distance from the hyperplanes represented by the internal layer. This class assignment function is again a hyperplane, this time over the vector of outputs of the internal nodes. For example, the 3-layer NN used later to learn XOR is:

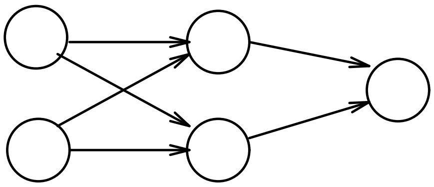

Figure 5: The 3-layer neural net. Nodes on the left hold the input values. The internal nodes multiply their inputs by a weight vector and scale with a sigmoid function. Their output is sent to the output node, which multiplies their inputs by a weight vector and again scales with a sigmoid. The actual value of the output is generally unimportant during classification – only the sign is important.

3 The DT Framework

We define a domain D to be a tuple of an instance space, a set of class labels, and the target concept. So, $D = \langle IS, L, C \rangle$, $C : IS \mapsto L$. We define a transformation function T as $T : IS \mapsto IS'$. So a domain D' is the T-transform of domain D iff $D' = \langle IS, L, C' \rangle$ where $C'(T(x)) = C(x)$. All geometric transformations of domains in Euclidean 2-space are shown in Figure 6. All topological transformations are shown in Figure 7. Note that we intend to depict the transformation of an entire domain, not just an instance in the domain. For example, the classic example of desired invariance under certain instance transformations is character recognition, where we desire to recognize characters regardless of their position, scale, rotation, and slant. Domain transformation is a different idea entirely, and can be thought of as moving all instances in instance space without changing their class label. The notion of domain transformation independence under linear transformations has been thoroughly studied by Duda and Hart [1] and Fukunaga [2].

Anyone who deals with real data should be concerned about invariance to domain transformations. For example, measuring the length of an object in inches instead of centimeters changes the scale of the domain along one axis. This effects an unscaled nearest-neighbor algorithm as well as the naive SPHERE algorithm.

Thus a desirable property of a learning algorithm would be insensitivity to domain transformations. Formally, we define a learning algorithm as a function from a labeled instance set LIS to a concept C. Labeled instance sets are generated from domains, sampling instances from the instance space IS and using the target concept C to assign

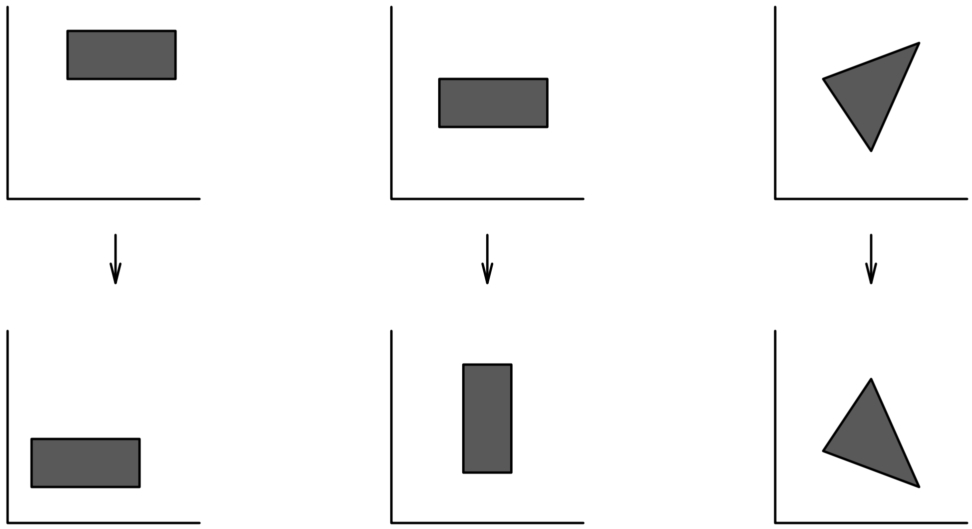

Figure 6: Geometric Transformations. Translation, rotation, inversion. These preserve geometric congruence.

a class label from L. The transformation of a labeled instance set $T(LIS)$ is generated by replacing each labeled instance $\langle inst, lab\rangle$ with $\langle T(inst), lab\rangle$.

We define a learning algorithm A to be invariant under a transformation T if, for all domains D, for any labeled instance set LIS generated from that domain, $A(LIS) = A(T(LIS))$. So if we get the same decision surface when learning D and $T(D)$ then A is invariant under T. We define a learning algorithm to be invariant under a class of transformations if, for all transformations T in the class, A is invariant under T.

For example, of the learning algorithms discussed here, hyperrectangles are the only representation that is not invariant under rotation. Thus, when deciding on a learning algorithm to use on a particular domain, unless it is somehow known that all decision surfaces are parallel to the axes of the instance space, one should expect the performance of the hyperrectangle classifier to be erratic.

Say you sample 100 points from $D1$ and learn a convex hull $H1$. Now transform $D2 = T(D1)$ with T being the function mapping $\langle x, y\rangle$ to $\langle 2x, y-3\rangle$. This is a kind of linear stretching and translation. Now take the same 100 points from $D2$ and learn again. Call the learned hull $H2$. It will of course be the case that for all points x in $IS1$ (not just those in the training sample), $H1(x) = H2(T(x))$. That is, the hulls are the same. Hyperspheres, the Nearest-Neighbor algorithm, and hyperplanes all vary with stretching.

The point is that when going from one domain to another it's as if rather than throwing the old one away and bringing in a new one we can just transform the old one into the new one. If an algorithm is invariant under the transformations then it's more powerful than one that isn't, and we should expect it to exhibit similar classification accuracy on the new domain. If we can pick a set of transformations then we can define relative complexity between domains as the number of transformations we have to do (more importantly, which kinds of transformations). Thus, given a canonical domain and transformation operators we have a framework for comparing the complexity of domains. We can speak of a domain in terms of its DT-distance from the canonical domain.

Figure 8 shows neural net being trained. The internal nodes of a 3-layer NN effectively perform a domain transformation from domain to a canonical domain! They attempt to map whatever domain they're trying to learn into a linearly-separable domain, their canonical domain. This shows how the internal nodes topologically transform instance space. We call this space the transform space. As learning progresses, the internal nodes actually learn a transformation from the instance space to a linearly separable space.

Thus we are able to define a canonical domain for neural nets: the linearly separable domain with some Euclidean metric space as the instance space, the set 0, 1 for the class label space, and the hyperplane which maps the vector x into 1 if $x1 > 0$, and 0 if $x1 < 0$. The top node in the 3-layer XOR NN can be thought of as transforming transform space into the canonical domain.

Convex hulls are invariant under all geometric transformations and all topological transformations except nonlinear stretching. For this reason we promote hulls as the proper representational choice in Euclidean metric spaces.

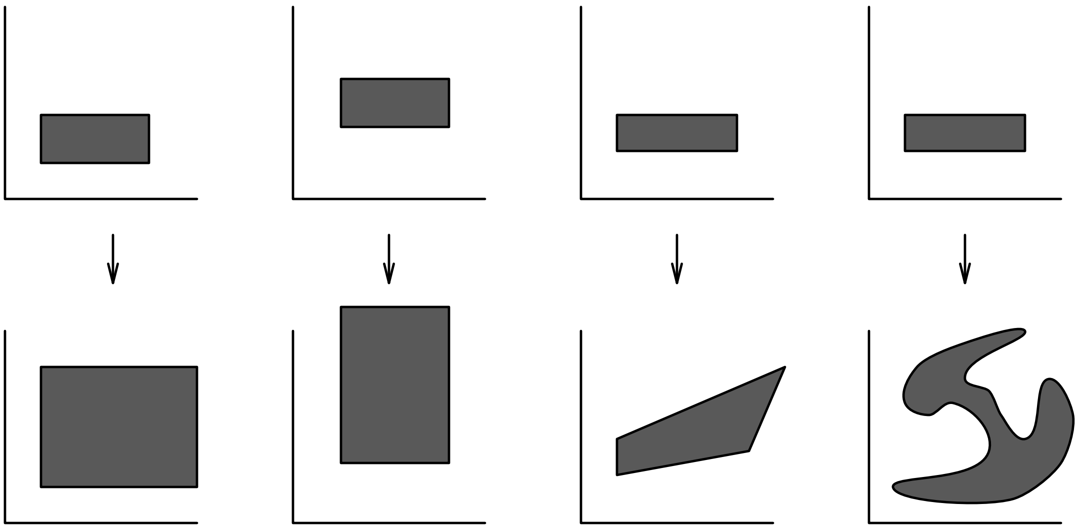

Figure 7: Topological Transformations. Scaling, stretching linearly parallel to the axes, skewed linear stretching, and unconstrained stretching.

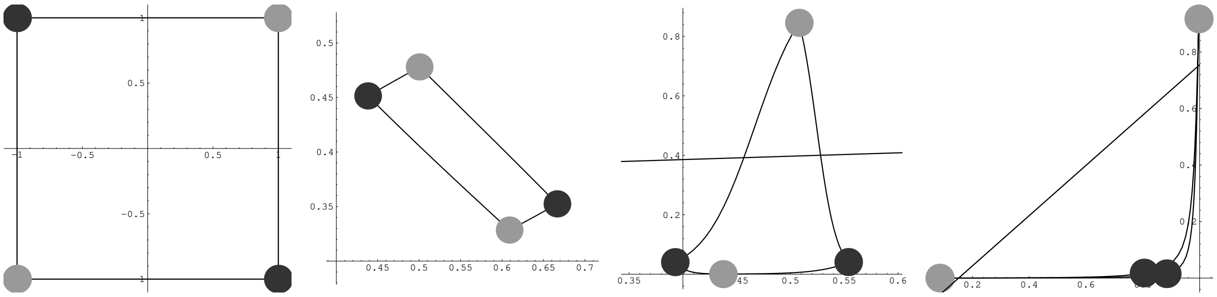

Figure 8: Neural net training. The far left shows the original XOR dataset. The next 3 plots show the neural network's internal representation of the datapoints (the activation of the 2 hidden units plotted as x-y coordinates) at various stages during training. To better show how the network warps the input space, we also draw straight lines between the four points and show how these lines become stretched and curved. The straight line appearing in the last two plots is actually the output unit, which just implements a linear separator in the hidden unit activation space.

Invariance is perhaps a stricter notion than will be useful – this remains to be seen. Another notion is learnability-invariance, which relaxes the notion of classifying all points in the domain in the same way, and instead only requires that the points in the training set still be classified correctly. This is too weak, though, because with it all classifiers that can learn a training set are equal.

4 Conclusions

While we recognize the importance of empirical studies, we feel that there is still a need for a good theoretical understanding of our concept representation languages and our domains. PAC theory [11] allows us to state bounds on the worst-case number of examples needed to identify a concept given a particular representation language, but it gives us no other way to characterize a representation. DT theory provides a way to measure the learning-difficulty of a domain, as well as the power of a learning algorithm. Learning algorithms which are invariant under more transformations will exhibit consistent performance over a wider range of domains. Domains which are DT-further from a canonical domain are expected to be harder to learn. DT-distance from a canonical domain provides an interesting way of characterizing the learning-simplicity or learning-difficulty of the domain – the expected generalization performance.

5 Acknowledgments

Thanks to Nils Nilsson for supporting this work, and to Pat Langley and David Stork for pointers to related work.

References

[1] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

[2] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Electrical Science. Associated Press, London, 1972.

[3] K. Gross. *Concept Acquisition Through Attribute Evolution and Experiment Selection*. PhD thesis, Carnegie Mellon University, 1990.

[4] M. J. Hudak. Rce networks: An experimental investigation. In *IJCNN-91*, 1991.

[5] J. McClelland and D. Rumelhart. *Parallel Distributed Processing*. MIT Press, 1986.

[6] N. J. Nilsson. *Learning Machines*. Morgan Kaufmann, 1990.

[7] F. Preparata and M. Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, 1985.

[8] R. L. Rivest. Learning decision lists. *Machine Learning*, 2:229 – 246, 1987.

[9] S. Salzberg. Nested hyper-rectangles for exemplar-based learning. In K. P. Jantke, editor, *Analogical and Inductive Inference*, pages 184–201, Berlin, Germany, October 1989. Springer-Verlag.

[10] S. Salzberg. A nearest hyperrectangle learning method. *Machine Learning*, 6(3):251–276, May 1991.

[11] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[12] B. Widrow and M. A. Lehr. 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 79(8):1415–1442, September 1990.