Improving Generalization with Active Learning*

January 23, 1992

David Cohn
Brain and Cognitive Sciences
Massachusetts Inst. of Technology
Cambridge, MA 02139

Les Atlas
Electrical Engineering
University of Washington
Seattle, WA 98195

Richard Ladner
Computer Science & Eng.
University of Washington
Seattle, WA 98195

Abstract

Active learning differs from passive "learning from examples" in that the learning algorithm assumes at least some control over what part of the input domain it receives information about. In some situations, active learning is provably more powerful that learning from examples alone, giving better generalization for a fixed number of training examples.

In this paper, we consider the problem of learning a binary concept in the absence of noise (Valiant 1984). We describe a formalism for active concept learning called *selective sampling*, and show how it may be approximately implemented by a neural network. In selective sampling, a learner receives distribution information from the environment and queries an oracle on parts of the domain it considers "useful." We test our implementation, called an *SG-network*, on three domains, and observe significant improvement in generalization.

1 Introduction: Random Sampling vs. Active Learning

Most neural network generalization problems are studied only with respect to random sampling: the training examples are chosen at random, and the network is simply a passive learner. This approach is generally referred to as "learning from examples." Baum and Haussler (1989), examine the problem analytically for neural networks; Cohn and Tesauro (1992) provide an empirical study of neural network generalization when learning from examples. There have also been a number of empirical efforts, such as Le Cun et al. (1990), aimed at improving neural network generalization when learning from examples.

Learning from examples is not, however, a universally applicable paradigm. Many natural learning systems are not simply passive, but instead make use of at least some form of active learning to examine the problem domain. By *active learning*, we mean any form of learning in which the learning program has some control over the inputs it trains on. In natural systems (such as humans), this phenomenon is exhibited at both high levels (e.g. active examination of objects) and low, subconscious levels (e.g. Fernald and Kuhl's (1987) work on infant reactions to "Motherese" speech).

Within the broad definition of active learning, we will restrict our attention to the simple and intuitive form of concept learning via *membership queries*. In a membership query, the learner queries a point in the input domain and an oracle returns the classification of that point. Much work in formal learning theory has been directed to the study of queries (see e.g.: Angluin 1986, Valiant 1984), but only very recently have queries been examined with respect to their role in improving generalization behavior.

In many formal problems, active learning is provably more powerful than passively learning from randomly given examples. A simple example is that of locating a boundary on the unit line interval. In order to achieve an expected position error of less than ϵ, one would need to draw $O(\frac{1}{\epsilon}\ln(\frac{1}{\epsilon}))$ random training examples. If

*As published in *Machine Learning* **15**(2):201-221, 1994. A preliminary version of this paper appears as (Cohn et al., 1990).

one is allowed to sequentially make membership queries, then binary search is possible and, assuming a uniform distribution, a position error of ϵ may be reached with $O(\ln(\frac{1}{\epsilon}))$ queries.

One can imagine any number of algorithms for employing membership queries to do active learning. We have been studying the problem of learning binary concepts in an error-free environment. For such problems, a learner may proceed by examining the information already given and determining a *region of uncertainty*, an area in the domain where it believes misclassification is still possible. The learner then asks for examples exclusively from that region. This paper discusses a formalization of this simple approach, which we call *selective sampling*.

In Section 2, we describe the concept learning problem in detail and give a formal definition of selective sampling, describing the conditions necessary for the approach to be useful. In Section 3 we describe the SG-network, a neural network implementation of this technique inspired by version-space search (Mitchell, 1982). Section 4 contains the results of testing this implementation on several different problem domains, and Section 5 discusses some of the limitations of the selective sampling approach. Sections 6 and 7 contain reference to related work in the field and a concluding discussion of the paper.

2 Concept Learning and Selective Sampling

Given an arbitrary domain X, we define a *concept* c to be some subset of points in the domain. For example, X might be a two-dimensional space, and c might be the set of all points lying inside a fixed rectangle in the plane. We *classify* a point $x \in X$ by its membership in concept c: we write $c(x) = 1$ if $x \in c$, and $c(x) = 0$ otherwise. A popular use of artificial neural networks is as concept classifiers: x is presented as the input to an appropriately trained network, which then activates a designated output node above some threshold if and only if $x \in c$, that is, if x is an instance of concept c. Formally, a *concept class* C is a set of concepts, usually described by some description language. In the above example, our class C may be the set of *all* two-dimensional, axis-parallel rectangles (see Figure 1). In the case of neural networks, the concept class is usually the set of all concepts that the network may be trained to classify.

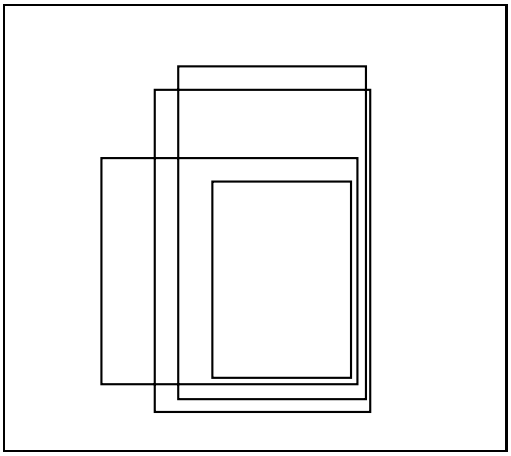

Figure 1: A concept class defined as the set of all axis-parallel rectangles in two dimensions. Several positive and negative examples are depicted, as are several consistent concepts in the class.

2.1 Generalization

For target concept t, a training example is a pair $(x, t(x))$ consisting of a point x (usually drawn from some distribution $\mathcal{P}$), and the point's classification $t(x)$. If $x \in t$, then $t(x) = 1$, and we say that $(x, t(x))$ is a *positive* example. Otherwise, $t(x) = 0$ and $(x, t(x))$ is a *negative* example. A concept c is *consistent* with an example $(x, t(x))$ if $c(x) = t(x)$, that is, if the concept produces the same classification of point x as the target. The *error* of c, with respect to t and distribution $\mathcal{P}$, is the probability that c and t will disagree on a random example drawn from $\mathcal{P}$. We write this as

$$\epsilon(c, t, \mathcal{P}) = Pr[c(x) \neq t(x)], \text{ for } x \text{ drawn randomly according to } \mathcal{P}.$$

The generalization problem is posed as follows: for a given concept class C, an unknown target t, an arbitrary error rate ϵ and confidence δ, how many examples do we have to draw and classify from an arbitrary distribution $\mathcal{P}$ in order to find a concept $c \in C$ consistent with the examples such that $\epsilon(c, t, \mathcal{P}) \leq \epsilon$ with confidence at least $1 - \delta$? This problem was formalized by Valiant (1984) and has been studied for neural networks in (Baum and Haussler, 1989) and (Haussler, 1989).

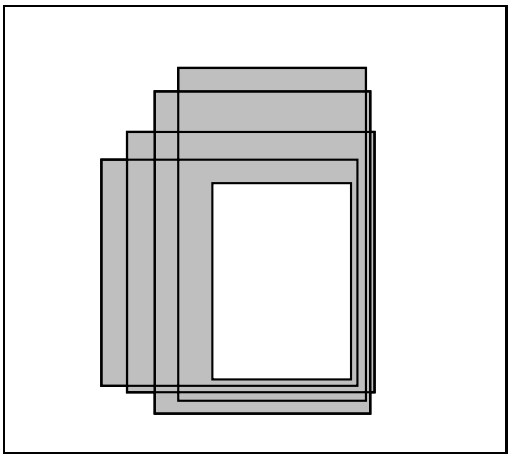

Figure 2: The region of uncertainty, $\mathcal{R}(S^m)$, is the set of all points x in the domain such that there are two concepts that are consistent with all training examples in S^m and yet disagree on the classification of x. Here, $\mathcal{R}(S^m)$ is shaded.

2.2 The region of uncertainty

If we consider a concept class C and a set S^m of m examples, the classification of some regions of the domain may be implicitly determined (Figure 2); all concepts in C that are consistent with all of the instances may agree in these parts. What we are interested in here is the areas that are *not* determined by available information – what we define to be the *region of uncertainty*:

$$\mathcal{R}(S^m) = \{x : \exists c_1, c_2 \in C,\ c_1, c_2 \text{ are consistent with all } s \in S^m, \text{ and } c_1(x) \neq c_2(x)\}.$$

For an arbitrary distribution $\mathcal{P}$, we can define the *size* of this region as $\alpha = Pr[x \in \mathcal{R}(S^m)]$. Ideally, in an incremental learning procedure, as we classify and train on more examples, α will be monotonically non-increasing. A point that falls outside $\mathcal{R}(S^m)$ will leave it unchanged; a point inside will further restrict the region. Thus, α is the probability that a new, random point from $\mathcal{P}$ will reduce our uncertainty.

As such, $\mathcal{R}(S^m)$ serves as an envelope for consistent concepts; any disagreement between those concepts must lie within $\mathcal{R}(S^m)$. Because of this, $\mathcal{R}(S^m)$ also bounds the potential error of any consistent hypothesis we choose. If the error of our current hypothesis is ϵ, then $\epsilon \leq \alpha$. Since we have no basis for changing our current hypothesis without a contradicting point, α is also a bound on the probability of an additional point reducing our error.

2.3 Selective sampling is active learning

Let us consider learning as a sequential process, drawing examples one after another, and determine how much information each successive example gives us. If we draw at random over the whole domain, then the probability that an individual sample will reduce our error is α, as defined above, which decreases to zero as the we draw more and more examples. This means that the efficiency of the learning process also

approaches zero; eventually, most examples we draw will provide us with no information about the concept we are trying to learn.

Now consider what happens if we recalculate $\mathcal{R}(S^m)$, the region of uncertainty after each new example, and draw examples only from within $\mathcal{R}(S^m)$. Then each example will reduce $\mathcal{R}(S^m)$, and will reduce our uncertainty, with no decrease in efficiency as we draw more and more examples. We call this process *selective sampling.*

If the distribution $\mathcal{P}$ is known (e.g. $\mathcal{P}$ is uniform), then we perform selective sampling directly by randomly querying points (according to $\mathcal{P}$) that lie strictly inside $\mathcal{R}(S^m)$. Frequently however, the sample distribution, as well as the target concept is unknown. In this case, we cannot choose points from our domain with impugnity or we risk assuming a distribution that differs greatly from the actual underlying $\mathcal{P}$. In many problems, though, we can still make use of distribution information without having to pay the full cost of drawing and classifying an example. Rather than assuming that the drawing of a classified example is an atomic operation (as in Valiant, 1984 and Blumer et al., 1988), we may divide the operation into two steps: first, that of drawing an unclassified example from the distribution, and second, querying the classification of that point. If the cost of drawing a point from our distribution is small compared to the cost of finding the point's proper classification, we can "filter" points drawn from our distribution, drawing at random, but only selecting, classifying and training on those that fall in $\mathcal{R}(S^m)$. This approach is well suited to problems such as speech recognition, where unlabeled speech data is plentiful, but the classifying (labeling) of speech segments is a laborious process.

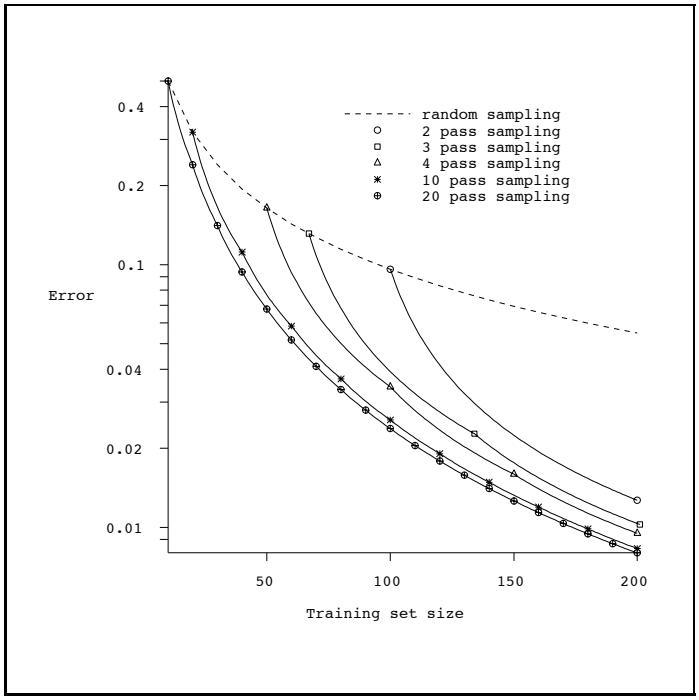

Figure 3: As the batch size in selective sampling approaches one, the process yields diminishing improvements for the added computational costs. This figure plots error vs. training set size for selective sampling using different batch sizes for learning an axis-parallel rectangle in two dimensions.

Since calculating $\mathcal{R}(S^m)$ may be computationally expensive, we may want to perform selective sampling in batches. On the first pass, we draw an initial batch of training examples S_0^m from $\mathcal{P}$ train on it, and determine the the initial $\mathcal{R}(S^m)$. We then define a new distribution $\mathcal{P}'$ to sample from that is zero outside $\mathcal{R}(S_0^m)$, but maintains the relative distribution of $\mathcal{P}$ inside $\mathcal{R}(S_0^m)$. We can then make a second pass, drawing a second batch of training examples S_1^m, adding it to the first, and determining a new, smaller $\mathcal{R}(S_0^m \cup S_1^m)$. The smaller the batch size is, and the more passes are made, the more efficiently the algorithm will draw training examples (see Figure 3). However, since $\mathcal{R}(S^m)$ is recalculated on each pass, this advantage must

be weighed against the added computational cost incurred in this calculation.

2.4 Approximations to selective sampling

Even for simple concept classes, such as the set of all axis-parallel rectangles in two dimensions, it may be difficult or computationally expensive to exactly represent the region of uncertainty. For the class of rectangles, negative examples that lie along the corners of the region can add complexity by causing "nicks" in the outer corners of $\mathcal{R}(S^m)$ (as in Figure 2). With more realistic, complicated classes, representing $\mathcal{R}(S^m)$ exactly can easily become a difficult, if not impossible, task. Using a good approximation of $\mathcal{R}(S^m)$ may, however, be sufficient to allow selective sampling. Practical implementations of selective sampling are possible with a number of approximations to the process, including maintaining a close superset or subset of $\mathcal{R}(S^m)$.

Assume we are able to maintain a superset $\mathcal{R}^+(S^m) \supseteq \mathcal{R}(S^m)$. Since any point in $\mathcal{R}(S^m)$ will also be in the superset, we can selectively sample inside $\mathcal{R}^+(S^m)$ and be assured that we will not exclude any part of the domain that is of interest. The penalty we pay is that of efficiency: we may also train on some points that are *not* of interest. The efficiency of this approach, as compared with pure selective sampling, can be measured as the ratio $Pr[x \in \mathcal{R}(S^m)]/Pr[x \in \mathcal{R}^+(S^m)]$.

If we are only able to maintain a subset $\mathcal{R}^-(S^m) \subseteq \mathcal{R}(S^m)$, our sampling and training algorithm must take additional precautions. On any given iteration, some part of $\mathcal{R}(S^m)$ will be excluded from sampling. Because of this, we will need to ensure that on successive iterations we will choose subsets that cover the entire region of uncertainty (an example of this technique will be discussed in the next section). We will also need to keep the number of examples on each iteration small to prevent oversampling of one part of the domain.

For the remainder of this paper, we will denote an arbitrary algorithm's approximation to the true region of uncertainty as $\mathcal{R}^*(S^m)$.

3 Neural Networks for Selective Sampling

The selective sampling approach holds promise for improved generalization in many trainable classifiers. The remainder of this paper is concerned with demonstrating how an approximation of selective sampling may be implemented using a feedforward neural network trained with error backpropagation.

The backpropagation algorithm (Rumelhart et al., 1986) is a *supervised* neural network learning technique, in that the network is presented with a training set of input/output pairs $(x, t(x))$ and learns to output $t(x)$ when given input x. To train a neural network using standard backpropagation, we take training example $(x, t(x))$ and copy x into the input nodes of the network (as in Figure 4).[1] We then calculate the individual neuron outputs layer by layer, beginning at the first "hidden" layer and proceeding through the output layer. The output of neuron j is computed as

$$o_j(x) = \sigma\left(\sum_i o_i(x) \cdot w_{j,i}\right),$$

where $w_{j,i}$ is the connection weight to neuron j from neuron i, and $\sigma(a) = 1/(1+\exp(-a))$ is a sigmoidal "squashing" function that produces neuron outputs in the range $[0,1]$. We define the error of the output node n as $\delta_n(x) = (o_n(x) - t(x))^2$. This error value is propagated back through the network (see Rumelhart et al., 1986 for details), so that each neuron j has an error term $\delta_j(x)$. The connection weights $w_{j,i}$ are then adjusted by adding

$$\Delta w_{ji}(x) = \eta \delta_j(x) o_j(x), \tag{1}$$

where η is a constant "learning rate."

This adjustment incrementally decreases the error of the network on example $(x, t(x))$. By presenting each training example in turn, a sufficiently large network will generally converge to a set of weights where

[1] We assume that all inputs have been normalized to the range $[0,1]$.

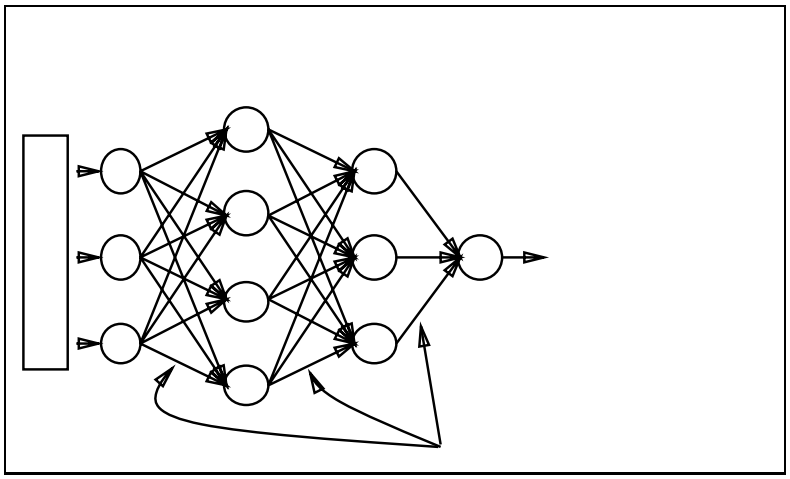

Figure 4: A simple feedforward neural network. Each node computes the weighted sum of its inputs, passes that sum through a sigmoidal "squashing" function, and passes the result on as its output.

the network has acceptably small error on each training example. In the concept learning model, the target values of training examples are 1 or 0, depending on whether or not the input is an instance of the concept being learned. Patterns are trained on until their error is less than some threshold.

At this point, we need to draw attention to the distinction between a neural network's architecture and its configuration.[2] The *architecture* of a neural network refers to those parameters of the network that do not change during training; in our case, this will be the network's topology and transfer functions. The *configuration* of a network refers to the network parameters that *do* change during training: in this case, the weights given to each of the connections between the neurons. Although there are network training algorithms that involve changing a network's topology during training (e.g. Ash, 1989), we consider here only those with fixed topologies that train by weight adjustment. The theory and methods described here should, with some modification, be equally applicable to other trainable classifiers.

For a neural network architecture with a single output node, the concept class C is specified by the set of all configurations that the network can take on. Each of these configurations implements a mapping from an input x to an output in $[0,1]$, and many configurations may implement the same mapping. If we set a threshold (such as 0.5) on the output, then we may say that a particular configuration $\tilde{c}$ represents a concept c such that $x \in c$ if and only if $\tilde{c}(x) > 0.5$ (see Figure 5). Having trained on training set S^m then, we can say that the network configuration $\tilde{c}$ implements a concept c that is consistent with training set S^m. Below, we will use c to denote both the concept c and the network $\tilde{c}$ that implements it.

Below, we consider a naïve algorithm for selective sampling with neural networks and examine its shortcomings. We then describe the SG-net, based on the version-space paradigm (Mitchell, 1982), that overcomes these difficulties.

3.1 A naïve neural network querying algorithm

The observation that a neural network implementation of a concept learner may produce a real-valued output that is thresholded suggests a naïve algorithm for defining a region of uncertainty. When a network is trained with these tolerances, we can divide all points in the domain into one of three classifications: "1" (0.9 or greater), "0" (0.1 or less), and "uncertain" (between 0.1 and 0.9). We may say that this last category corresponds to a region where the network is uncertain, and may thus define it to be $\mathcal{R}^*(S^m)$, our approximation to the region of uncertainty (Figure 6).

The problem with applying this approach is that it measures only the uncertainty of that particular configuration, not the uncertainty among configurations possible given the architecture. While it is in fact

[2]The terminology is that of Judd (1988).

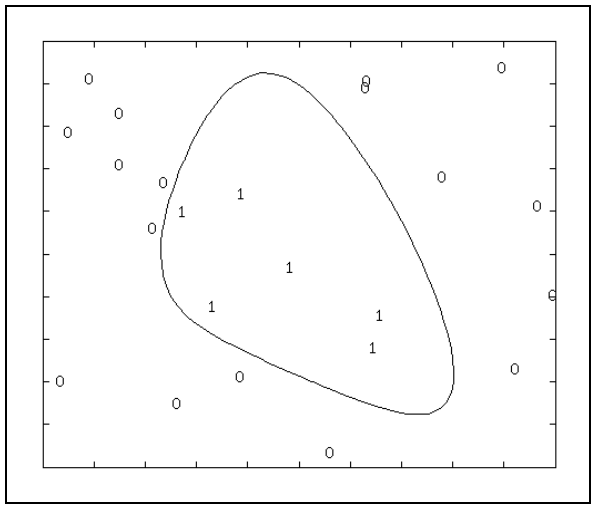

Figure 5: The thresholded output of the trained neural network $\tilde{c}$ serves as a classifier representing a concept c that is (hopefully) similar to the unknown target concept.

a *part* of $\mathcal{R}(S^m)$, the full region is comprised of the differences between all possible consistent network configurations.

This limitation is exacerbated by the inductive bias of some learning algorithms, including backpropagation. The backpropagation algorithm, when attempting to classify a set of points, tends to draw sharp distinctions and become "overly confident" in regions that are still unknown. As a result, the $\mathcal{R}^*(S^m)$ chosen by this method will in general be a very small subset of the true region of uncertainty.

A pathological example of this behavior is exhibited in Figures 7a and 7b. In Figure 7a, the initial random sampling has failed to yield any positive examples in the triangle on the right. Training by backpropagation on the examples yields a region of uncertainty (between the two contours) that concentrates on the left half of the domain, completely to the exclusion of the right. The final result of 10 iterations of querying and learning is shown in Figure 7b. This strategy (and related ones) is prone to failure of this form whenever there are regions of detail in the target concept that are not discovered in the initial random sampling stage.

3.2 Version space

Mitchell (1982) describes a learning procedure based on the partial ordering in generality of the concepts being learned. Some concept c_1 is "more general" than another concept c_2 if and only if $c_2 \subset c_1$. If $c_1 \not\subset c_2$ and $c_2 \not\subset c_1$, then the two concepts are incomparable. For a concept class C and a set of examples S^m, the *version space* is the subset $C_{S^m} = \{c \in C, \text{such that } c \text{ is consistent with all } s \in S^m\}$. To bound the concepts in the version space, we can maintain two subsets, $S, G \subseteq C_{S^m}$. S is the set of all "most specific" consistent concepts, that is, $S = \{c \in C_{S^m}, \nexists c' \in C_{S^m}, c' \subset c\}$. Similarly, $G = \{c \in C_{S^m}, \nexists c' \in C_{S^m}, c' \supset c\}$ is the set of "most general" concepts. For any consistent concept c, it must be the case that $s \subseteq c \subseteq g$ for some $s \in S$ and $g \in G$.

One may do active learning with a version space by examining instances that fall in the "difference" of S and G, that is, the region $S\Delta G = \bigcup\{s\Delta g : s \in S, g \in G\}$ (where Δ is the symmetric difference operator). If an instance in this region proves positive, then some s in S will have to generalize to accommodate the new information; if it proves negative, some g in G will have to be modified to exclude it. In either case, the version space, the space of plausible hypotheses, is reduced with every query.

3.3 Implementing an active version-space search

Since an entire neural network configuration represents a single concept, a complete version space cannot be directly represented by any single neural network. In fact, Haussler (1987) pointed out that the size of the

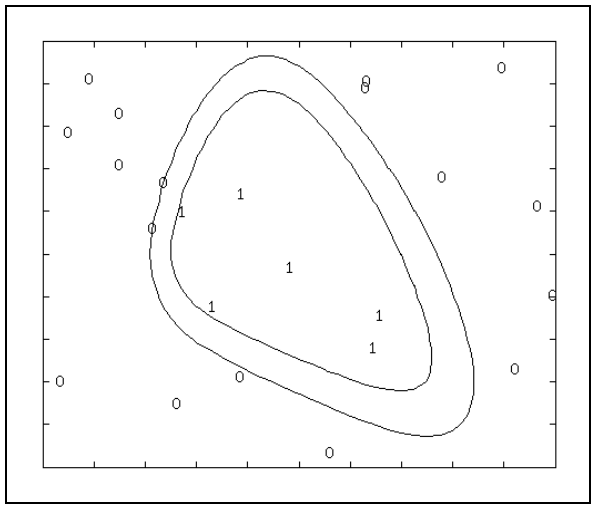

Figure 6: A naïve approach to representing the region of uncertainty: we can use the network's transition area between 0 and 1 to represent the part of the domain where the network is "uncertain."

S and G sets could grow exponentially in the size of the training set. Representing these sets completely would require keeping track of and manipulating an exponential number of network configurations.

We can, however, modify the version-space search to make the problem tractable. This can be done if we impose, according to the distribution $\mathcal{P}$, a strict index of ordering on *all* concepts in the class. We will define a concept c_1 to be "more general" than concept c_2 if and only if for a random point x drawn from $\mathcal{P}$, $Pr[x \in c_1] > Pr[x \in c_2]$. Under this definition, the generality of all concepts in the class is comparable, and it makes sense to speak of an ordering in which we can represent a single "most general" concept g and a single "most specific" concept s. There may still be many concepts with the same generality, but this is no impediment. We need only know that there are no concepts, in the "most general" case, with a *greater* generality than the concept g we have chosen.

By maintaining these two concepts, we have a window into our version space: $\mathcal{R}^*(S^m) = s\Delta g$ will be a subset of $S\Delta G$. Thus, a point $x \in \mathcal{R}^*(S^m)$ is guaranteed to reduce the size of our version space. If positive, it will invalidate s and leave us with another s, either a more general one, or an equally specific one that includes the new point. Similarly, if the new point is classified as negative, it will invalidate g. By proceeding in this fashion, we can approximate a step-by-step traversal of the S and G sets using a fixed representation size.

3.4 The SG-net: a neural network version-space search algorithm

Since we are interested in selecting examples that improve the generalization behavior of some given neural network architecture N, we define the concept class in question to be the the set of concepts learnable by N with its learning algorithm. If we can manage to obtain network configurations that represent the s and g concepts described above, then it is a simple matter to implement the modified version-space search. In the following two subsections, we first describe how one may learn a "most specific" or "most general" concept associated with a network, and then describe how these two networks may be used to selectively sample the $\mathcal{R}^*(S^m)$ defined by regions where they disagree.

3.4.1 Implementing a "most specific/general" network

Below, we describe how one may learn s, a "most specific" concept consistent with some given data. The case for learning g, a "most general" concept, is analogous.

A most specific network for a set of examples S^m (according to distribution $\mathcal{P}$), is one that classifies as positive those example points that are in fact positive and classifies as negative as much as possible of the rest of the domain. This requirement amounts to choosing a c consistent with S^m that minimizes $Pr[x \in c]$.

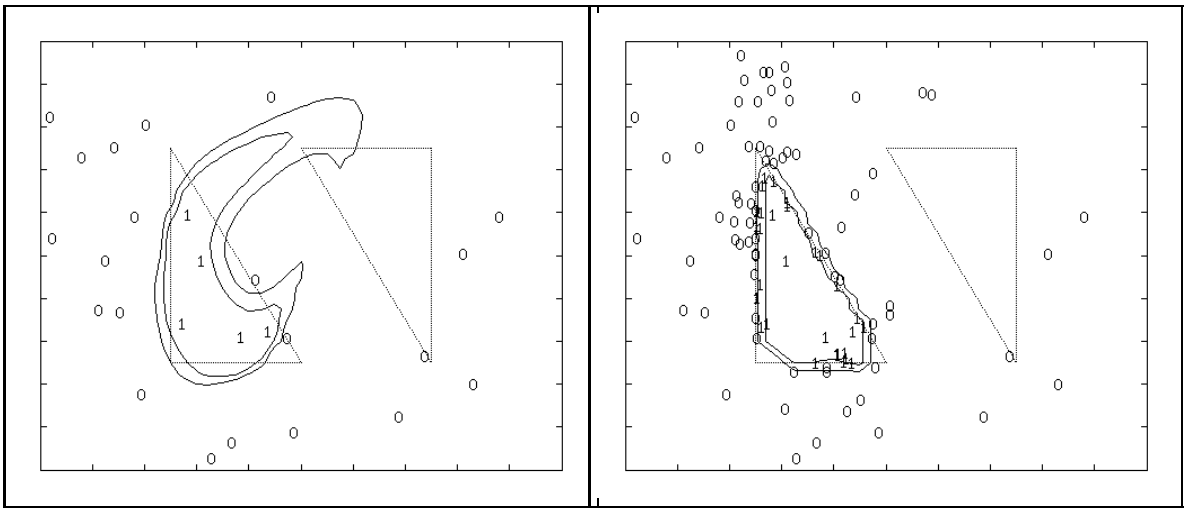

Figure 7: A pathological example of naïve network querying. In (a) on left, an initial random sample has failed to detect the second, disjoint region of the target concept. In (b) on right, after 10 successive iterations then, the naïve querying algorithm has ignored that region and concentrated on the region where is *has* seen examples. The dotted line denotes the true boundary of the unknown target concept.

Such a network may be arrived at by employing an *inductive bias*. An inductive bias is a predisposition of a learning algorithm for some solutions over others. Most learning algorithms inherently have at least some form of an inductive bias, whether it is a preference for simple solutions over complex ones, or a tendency to choose solutions where the absolute values of the parameters remain small.[3]

What we will do is explicitly add a *new* inductive bias to the backpropagation algorithm: by penalizing the network for any part of the domain that it classifies as positive, we add a bias that prefers specific concepts over general ones. The weight of this penalty must be carefully adjusted: if it is large enough to outweigh the training examples, the network will not converge on the training data. It must, however, be large enough to outweigh any other inductive bias in the learning algorithm, and force it to find a most specific configuration consistent with S^m.

This "negative" bias may be implemented by drawing unclassified points from $\mathcal{P}$ (or creating them in the case where $\mathcal{P}$ is known), and arbitrarily labeling them as negative examples. We then add these "background" examples to the training set (Figure 8). This creates a background bias over the domain that is weighted by the input distribution $\mathcal{P}$: the networks that have the least error on these background patterns will be the ones that are the most specific according to $\mathcal{P}$.

In order to allow the network to converge on the actual training examples in spite of the background examples, we must balance the influence of background examples against that of the training data. As the network learns training example x, the error term $\delta_j(x)$ in Equation 1 will approach zero, while the error term of an arbitrary background example y may remain constant. Unless the "push" that the random background example exerts on the network weights ($\Delta w_{ji}(y)$) is decreased to match that of the normal training examples ($\Delta w_{ji}(x)$), the background examples will dominate and network will not converge on a solution.

We achieve balance by using different learning rates for training examples and background examples. We dynamically decrease the background learning rate as a function of the network's error on the training set. Each time we present a training example x, we calculate a new background learning rate $\eta' = \gamma\delta(x)\eta$, where $\delta(x)$ is the error of the network on x, and $0 < \gamma \leq 1$ is a constant. We then train on a single background

[3]The inductive biases inherent in backpropagation have not been well studied, but there appears to be a tendency to fit the data using the smallest number of units possible.

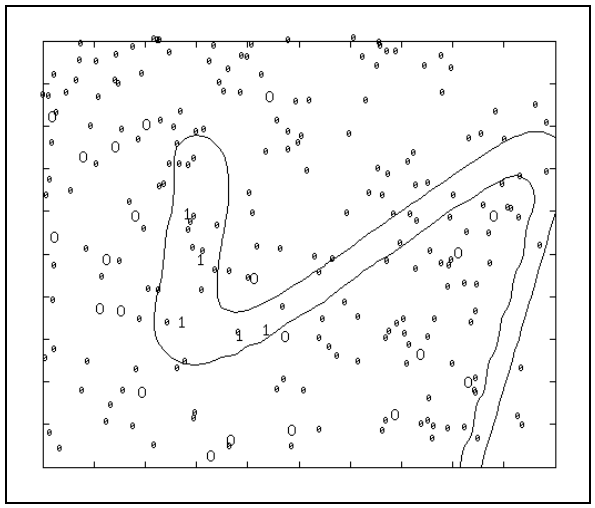

Figure 8: Training on a large number of "background" points in addition to the regular training data forces the network into a "most specific" configuration.

example using this value of η' and repeat. Formally, the algorithm is as follows:

1. Initialize network to random configuration c.
2. If for all actual training examples $(x, t(x)) \in S^m$, $((c(x) - t(x))^2 < threshold$, stop.
3. Otherwise, select next actual training example $(x, t(x))$.
4. Calculate the output error of network c on input x, $\delta(x) = (c(x) - t(x))^2$, and backpropagate through the network, adjusting weights according to

$$\Delta w_{ji}(x) = \eta \delta_j(x) o_j(x).$$

5. Calculate new background learning rate $\eta' = \gamma \delta(x) \eta$.
6. Draw a point y from $\mathcal{P}$ and create background example $(y, 0)$.
7. Calculate the output error $\delta(y) = (c(y_k))^2$, and backpropagate through the network, adjusting weights according to the modified equation

$$\Delta w_{ji}(y) = \eta' \delta_j(y) o_j(y).$$

7. Go to Step 2.

Optimally, γ should be set such that the weight update on the background patterns is always infinitesimally smaller than the weight update on the actual training patterns, allowing the network to "anneal" to a most specific configuration. This however, requires a prohibitive amount of training time. Empirically, we have found that setting $\gamma = 0.75$ provides an adequate bias and still allows convergence in a reasonable number of iterations.

A similar procedure can be used to produce a "most general" network, adding a positive inductive bias by classifying all background points drawn from $\mathcal{P}$ as positive.

3.4.2 Implementing active learning with an SG-net

Once we can represent concepts s and g, it is a simple matter to test a point x for membership in $\mathcal{R}^*(S^m)$ by determining if $s(x) \neq g(x)$. Selective sampling may then be implemented as follows: if a point drawn from the distribution is not in $s\Delta g$ (if the two networks agree on their classification of it), then the point is discarded. If point *is* in $s\Delta g$, then its true classification is queried and it is added to the training set. In practice, we can merge the inputs of the s and g networks, as illustrated in Figure 9, and train both together.

It is important to note that this technique is somewhat robust, in that its failure modes degrade the efficiency of a single sampling iteration rather than causing overall failure of the learning process. If either

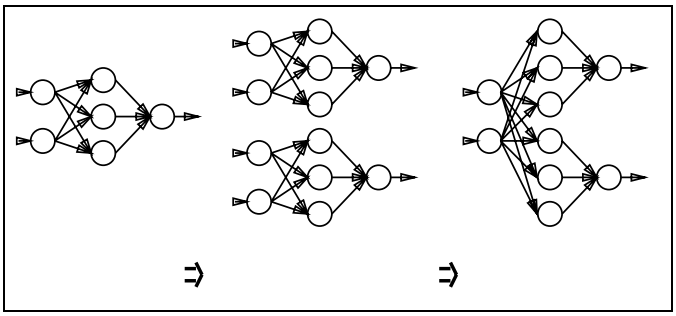

Figure 9: Construction of an SG-network equivalent to the original.

the s or g networks fail to converge on the training data, the points that failed to converge will be contained in $s\Delta g$, and that region will be eligible for additional sampling on the next iteration. In most cases, we have found that these additional examples will suffice to "push" the network out of its local minimum.

If the network does converge on the training set, but settles on solutions that are not near the most specific/general networks consistent with the data, the examples gleaned in the next iteration are still useful. Since they were chosen by virtue of lying in areas where the two networks disagreed, the points will settle discrepancies between the two. This may lead to some oversampling of the region, but will not, in and of itself, cause the technique to fail.

The effects of these two failure modes can be minimized by keeping the number of examples taken on each iteration small. This increases the efficiency of the learning process in terms of the number of examples classified, but as we have observed, there is a tradeoff in the computational resources required. Each time new data is added to the training set, the network may have to completely readjust itself to incorporate the new information. We have found that in practice, with large training set sizes, it is often most efficient to simply retrain the entire network from scratch when new examples are added. Recent work by Pratt (1993) offers hope that this retraining may be made more efficient by use of "information transfer" strategies between iterations.

4 Experimental Results

Experiments using selective sampling were run on three types of problems: solving a simple boundary-recognition problem in two dimensions, learning a 25-input real-valued threshold function, and recognizing the secure region of a small power system.

4.1 The triangle learner

A two-input network with two hidden layers of 8 and 3 units and a single output was trained on a uniform distribution of examples that were positive inside a pair of triangles and negative elsewhere. This task was chosen because of its intuitive visual appeal and because it requires learning a non-connected concept – a task that demands more from the training algorithm (and sample selection scheme) than a simple convex shape.

The baseline case consisted of 12 networks trained on randomly drawn examples with training set sizes from 10 to 150 points in increments of 10 examples. Eight test cases were run on the same architecture with data selected by four runs of an SG-network using 15 selective sampling iterations of 10 examples each (Figures 10a and 10b). Additionally, 12 runs of the naïve querying algorithm described in Section 3.1 were run for comparison.

The networks trained on the selectively sampled data showed marked, consistent improvement over both the randomly sampled networks and the ones trained with naïve querying (Figure 11). The naïve querying algorithm displayed much more erratic performance than the other two algorithms, possibly due to the pathological nature of its failure modes.

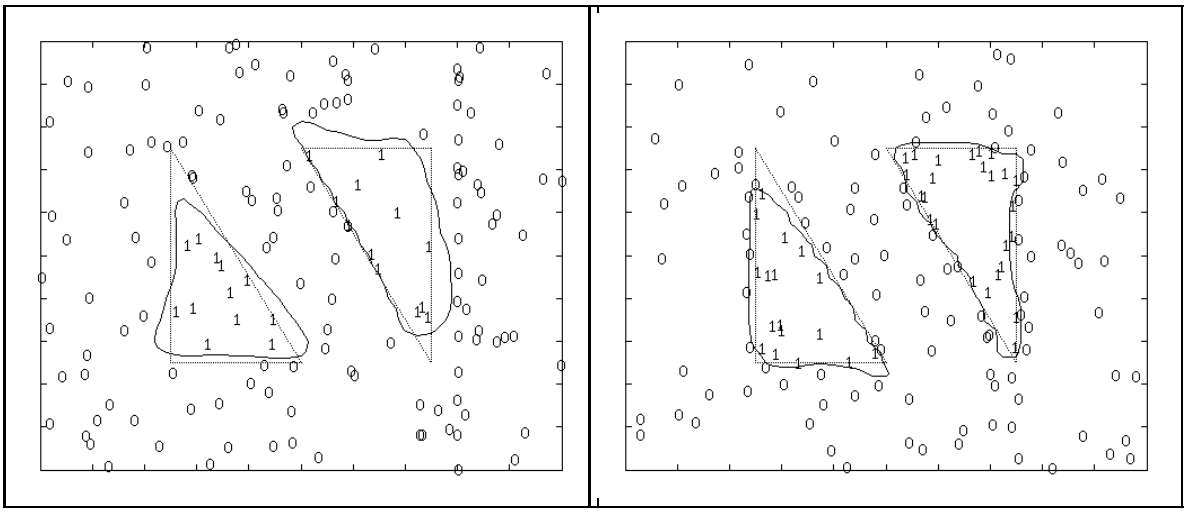

Figure 10: The triangle learner problem. When learned by 150 random examples in (a) on left, and when learned by 150 examples drawn in 15 passes of selective sampling in (b) on right. The dotted line denotes the true boundary of the unknown target concept.

4.2 Real-valued threshold function

We used the 25-bit real-valued threshold problem as a quantitative measure of network performance on a simple but higher-dimensional problem. Six runs of selective sampling, using iterations of 10 examples per iteration, were trained on the problem and compared to 12 identical networks trained with randomly sampled data. The results (Figure 12) indicate a much steeper learning curve for selective sampling.

Plotting the generalization error against the number of training examples m, networks trained on the randomly sampled data exhibited a roughly polynomial curve, as would be expected following Blumer et al. (1988). Using simple linear regression on $\frac{1}{\epsilon}$, the error data fit $\epsilon = (a \cdot m + b)^{-1}$ (for $a = 0.0514$ and $b = -0.076$) with a coefficient of determination (r^2) of 0.987. Networks trained on the selectively sampled data, by comparison, fit $\epsilon = (a \cdot m + b)^{-1}$ with $r^2 = 0.937$, indicating that its fit to the polynomial was not as good.

Visually, the selectively sampled networks exhibited a steeper drop in the generalization error, as would be expected from an active learning method. Using linear regression on the natural logarithm of the errors, the selectively sampled networks exhibited a decrease in generalization error matching $\epsilon = e^{a \cdot m + b}$ for $a = -0.0218$ and $b = 0.071$ with $r^2 = 0.995$ (up until the error drops below 1.5%), indicating a good fit to the exponential curve. By comparison, the randomly sampled networks' fit to $\epsilon = e^{a \cdot m + b}$ achieved only $r^2 = 0.981$.

In this domain, the SG-network appears to provide and almost exponential improvement in generalization with increasing training set size, much as one would expect from a good active learning algorithm. This suggests that the SG-network represents a good approximation to the region of uncertainty (in this domain) and thus implements a good approximation of selective sampling.

Additional experiments, run using 2, 3, 4, and 20 iterations indicate than the error decreases as the sampling process is broken up into smaller, more frequent iterations. This observation is consistent with an increased efficiency of sampling as new information was incorporated earlier into the sampling process.

4.3 Power system security analysis

If various load parameters of an electrical power system are within a certain range, the system is secure. Otherwise it risks thermal overload and brown-out. Previous research (Aggoune et al., 1989) determined that this problem was amenable to neural network learning, but that random sampling of the problem domain was inefficient in terms of examples needed. The range of parameters over which the system will be run is known, so distribution information is readily available. For each set of parameters (each point in the domain), one can analytically determine whether the system is secure, but this must be done by solving a

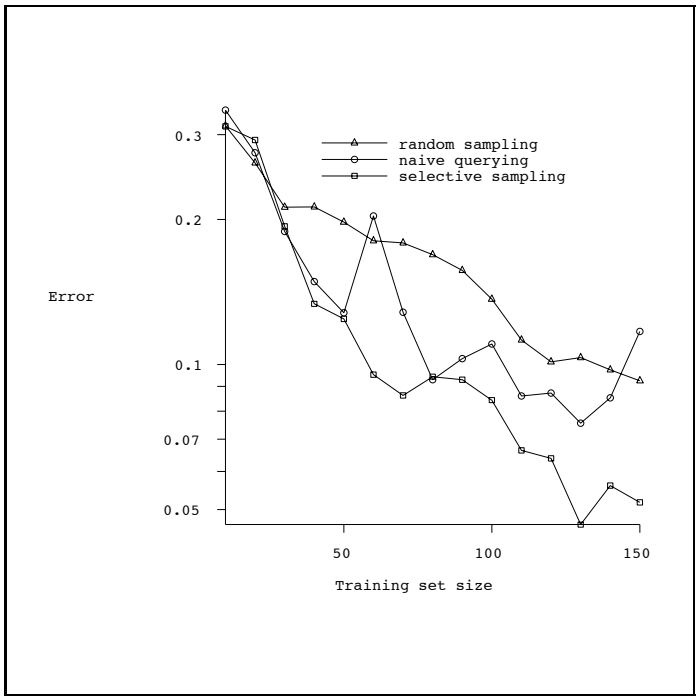

Figure 11: Generalization error vs. training set size for random sampling, naïve querying, and selective sampling. The irregularity of the naïve querying algorithm's error may be due to its intermittent failure to find both triangles in the intial random sample.

time-consuming system of equations. Thus, since the *classification* of a point is much more expensive than the determination of an input distribution, the problem is amenable to solution by selective sampling.

The baseline case of random sampling in four dimensions studied by Hwang et al. (1990), was used for comparison. In our experiments, we ran six sets of networks on the initial, random training sets (with 500 data points) and added a single iteration of selective sampling. Networks were trained on a small second iteration of 300 points (for a total of 800) as well as a large second iteration of 2000 (for a total of 2500 points). These results were compared to the baseline cases of 800 and 2500 points of randomly sampled data.

We estimated the network errors by testing on 14,979 randomly drawn test points. The improvement that the single extra iteration of selective sampling yielded for the small set was over 10.7% of the total error (5.17% instead of 5.47%), while on the large set it resulted in an improvement of 12.6% of total (4.21% instead of 4.82%). This difference is significant with greater than 90% confidence.

5 Limitations of the Selective Sampling Approach

There are a number of limitations to the selective sampling approach; some are practical, as mentioned in the previous section discussing implementations of the technique, while others are more theoretical.

5.1 Practical limitations

As discussed earlier in this paper, an exact implementation of selective sampling is practical only for relatively simple concept classes. As the class becomes more complex, it becomes difficult to compute and maintain an accurate approximation of $\mathcal{R}(S^m)$.

In the case of maintaining a superset, increased concept complexity seems to lead to cases where $\mathcal{R}^+(S^m)$ effectively contains the entire domain, reducing the efficiency of selective sampling to that of random sampling. The example in Section 2.4 illustrates this nicely: while a bounding box suffices as an approximation

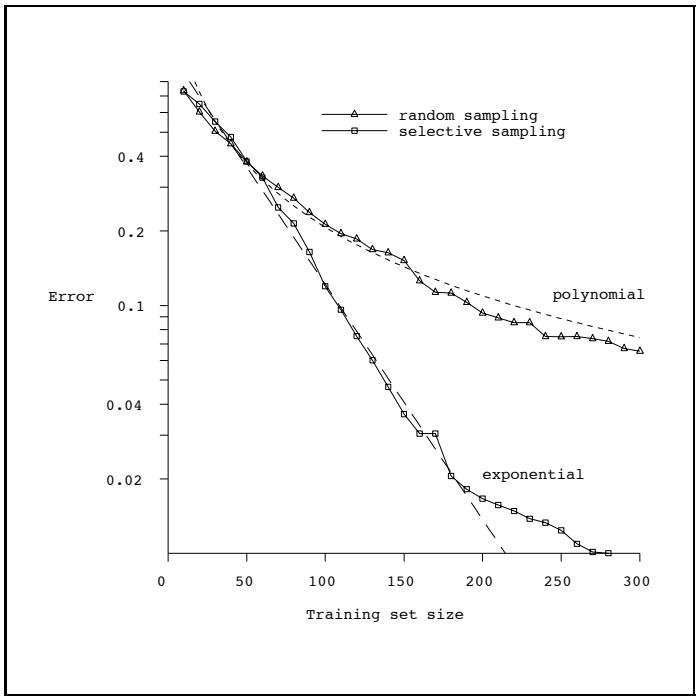

Figure 12: Generalization error vs. training set size for random sampling, and selective sampling. Standard deviation of error averages 0.00265 for the random case and 0.00116 for the selectively sampled case.

for rectangles in two dimensions, the "nicks" in such a box bounding a 20-dimensional figure could conceivably require the approximation to contain most of the domain space.

In the case of maintaining a subset, increased concept complexity leads to an extreme where $\mathcal{R}^-(S^m)$ contains only a very small subset of $\mathcal{R}(S^m)$. In these cases, oversampling of regions becomes a critical problem, and due to the inductive bias of the training algorithm, even a training set size of only one may omit large regions of the domain.

5.2 Theoretical limitations

Selective sampling draws its power from the ability to differentiate a region of uncertainty from the bulk of the domain. In cases where the representational complexity of the concept is large (as in a neural network with many hidden units), however, $\mathcal{R}(S^m)$ can extend over the whole domain until the concept is already well-learned. That is, even though the maximum error may be small, due to the number of places that this error may arise, the total uncertainty may remain large. Thus, depending on the desired final error rate, selective sampling may not come into effect until it is no longer needed. Similarly, if the input dimension is very large, the bulk of the domain may be uncertain, even for simple concepts. One method for avoiding this problem is the use of Bayesian probabilities to measure the degree of utility in querying various parts of the region of uncertainty. This approach has recently been studied by David MacKay (1991), and is discussed briefly in the following section.

6 Related Work

The work described in this paper is an extension of results published by Cohn et al. (1990). Prior to that work, and since then, there have been many related results in active learning. There is a large body of work studying the effects of queries from the strict learning theory viewpoint, primarily with respect to learning formal concepts such as Boolean expressions and finite state automata.

Angluin (1986) showed that while minimal finite state automata were not polynomially learnable (in the Valiant sense) from examples alone, they could be learned using a polynomial number of queries to an oracle that provides counter-examples. Valiant (1984) considers various classes that are learnable using a variety of forms of active learning.

Work by Eisenberg and Rivest (1990) puts bounds on the degree to which membership queries examples can help generalization when the underlying distribution is unknown. Additionally, given certain smoothness constraints on the distribution, they describe how queries may be used to learn the class of initial segments on the unit line.

Actual implementations of querying systems for learning have only recently been explored. Work done by Hwang et al. (1990) implements querying for neural networks by means of inverting the activation of a trained network to determine where it is uncertain. This approach shows promise for concept learning in cases with relatively compact, connected concepts, and has already produced impressive results on the power system static security problem. It is, however, susceptible to the pathology discussed in Section 3.1. An algorithm due to Baum and Lang (1991), uses queries to reduce the computational costs of training a single hidden-layer neural network. Their algorithm makes queries that allow the network to efficiently determine the connection weights from the input layer to the hidden layer.

Seung et al. (1992) independently proposed a similar scheme for selecting queries, basing it on a lack of consensus in a "committee" of learners. Freund et al. (1993) showed that as the size of the committee increases beyond the two learners used in selective sampling, the accuracy of one's "utility" estimate increases sharply.

Work by David MacKay (1992) pursues a related approach to data selection using Bayesian analysis. By assigning prior probabilities to each concept (or each network configuration) one can determine the utility of querying various parts of $\mathcal{R}(S^m)$. The fact that a point lies within $\mathcal{R}(S^m)$ means that there are consistent configurations that disagree on the classification of that point. If that point is on the edge of $\mathcal{R}(S^m)$ though, it may be that only a very few configurations disagree, and querying the point will only decrease the size of $\mathcal{R}(S^m)$ by an infinitesimally small amount. Using Bayesian analysis, one may, in effect, determine the "number" of configurations that disagree on a given point, and thus determine what parts of $\mathcal{R}(S^m)$ are most uncertain.

7 Conclusion

In this paper we have presented a theory of selective sampling, described a neural network implementation of the theory, and examined the performance of the resulting system in several domains.

Selective sampling is a very rudimentary form of active learning, but it has the benefit of a formal grounding in learning theory. In the neural network implementation tested, it demonstrates significant improvement over passive random sampling techniques on a number of simple problems.

The paradigm is suited for concept learning problems where the relevant input distribution is known, or where the cost of obtaining an unlabeled example from the input distribution is small compared with the cost of labeling that example. While the limitations of selective sampling become apparent on more complex problem domains, the approach opens the door to the study of more sophisticated techniques for querying and learning by the natural and intuitive means of active learning.

Acknowledgements

This work was supported by National Science Foundation grant number CCR-9108314, the Washington Technology Center, and the IBM Corporation. The majority of this work was done while David Cohn was with the Dept. of Computer Science and Engineering, University of Washington. The remainder was done while David Cohn was at IBM T. J. Watson Research Center, Yorktown Heights, NY 10598. We would like to thank Jai Choi and Siri Weerasooriya for their work in running simulation data for the power system problem. We would also like to thank two anonymous referees for their suggestions on an earlier version of this paper.

References

M. Aggoune, L. Atlas, D. Cohn, M. Damborg, M. El-Sharkawi and R. Marks II. (1989) Artificial neural networks for power system static security assessment. In *Proceedings, International Symposium on Circuits and Systems.*

D. Angluin. (1986) Learning regular sets from queries and counter-examples. Tech Report YALEU/DCS/TR-64, Yale University.

T. Ash. (1989) Dynamic node creation in backpropagation networks. *ICS Report 8901*, Institute for Cognitive Science, University of California, San Diego.

E. Baum and D. Haussler. (1989) What size net gives valid generalization? In D. Touretzky, ed., *Advances in Neural Information Processing Systems 1*, Morgan Kaufmann.

E. Baum and K. Lang. (1991) Constructing hidden units using examples and queries. In R. Lippmann et al., eds., *Advances in Neural Information Processing Systems 3*, Morgan Kaufmann.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. (1989) Learnability and the Vapnik-Chervonenkis dimension. *JACM* **36**(4):929–965.

A. Blum and R. Rivest. (1989) Training a 3-node neural network is NP-complete. In D. Touretzky, ed., *Advances in Neural Information Processing Systems 1*, Proceedings of the Neural Information Processing Systems Conference, Morgan Kaufmann.

D. Cohn, L. Atlas and R. Ladner. (1990) Training connectionist networks with queries and selective sampling. In D. Touretzky, ed., *Advances in Neural Information Processing Systems 2*, Morgan Kaufmann.

D. Cohn and G. Tesauro. (1992) How tight are the Vapnik-Chervonenkis bounds? *Neural Computation* **4**(2):249–269.

B. Eisenberg and R. Rivest. (1990) On the sample complexity of pac-learning using random and chosen examples. In M. Fulk and J. Case, eds., *ACM 3rd Annual Workshop on Computational Learning Theory*, Morgan Kaufmann.

A. Fernald and P. Kuhl. (1987) Acoustic Determinants of Infant Preference for Motherese Speech. *Infant Behavior and Development 10*:279–293.

Y. Freund, H. S. Seung, E. Shamir and N. Tishby. (1993) Information, prediction, and query by committee. In S. Hanson et al., eds., *Advances in Neural Information Processing Systems 5*, Morgan Kaufmann.

D. Haussler. (1987) Learning conjunctive concepts in structural domains. In *Proceedings, AAAI '87*, pp. 466–470.

D. Haussler. (1989) Generalizing the pac model for neural nets and other learning applications. *UCSC Tech Report UCSC-CRL-89-30.*

J. N. Hwang, J. Choi, S. Oh and R. Marks II. (1990) Query learning based on boundary search and gradient computation of trained multilayer perceptrons. *IJCNN 90, San Diego, June 17-21.*

S. Judd. (1988) On the complexity of loading shallow neural networks. *Journal of Complexity*, 4:177–192.

Y. Le Cunn, J. Denker, and S. Solla. (1990) Optimal brain damage. In D. Touretzky, ed., *Advances in Neural Information Processing Systems 2*, Morgan Kaufmann.

D. MacKay. (1992b)Information-based objective functions for active data selection, *Neural Computation* **4**(4): 590–604.

T. Mitchell. (1982) Generalization as search. *Artificial Intelligence* **18**:203–226.

L. Y. Pratt. (1993) Discriminability-based transfer between neural networks In C.L. Giles et al., eds., *Advances in Neural Information Processing Systems 5*, Morgan Kaufmann.

D. Rumelhart, G. Hinton and R. Williams. (1986) Learning internal representations by error propagation. In D. Rumelhart and J. McClelland, eds., *Parallel Distributed Processing*, MIT Press.

H. S. Seung, M. Opper, and H. Sompolinsky. (1992) Query by committee. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pp. 287–294, ACM, New York.

L. Valiant. (1984) A theory of the learnable. *Communications of the ACM*, 27:1134–1142.