# Topic Summary 1

September 28, 2001

## 1 Introduction

This paper is a brief summary of the first topic covered in Computer Science 478-878, Machine Learning. Thus far we have examined two general topics, which included first an overview of machine learning ideas and issues, and second, the idea of concept learning.

Learning, in this scope, is defined as improving performance with experience with respect to some task. The methods that can be used to accomplish this occupy the focus of machine learning; specifically, how can we encourage a program to efficiently solve some problem for which it is inefficient or impossible to describe the solution space statically via a deterministic algorithm.

## 2 Material summary

### 2.1 Lecture 1

A learning problem is defined as a task *T*, a performance measure *P*, and a training experience *E* such that we desire to optimize *P* by way of experiences in *E*. This allows us to define *E*, as well as representation of the task, the behavior to be learned, as well as the algorithm used to approach the problem.

We can describe the training experience according to various options. These include whether the training experience is direct or indirect. In a direct training experience, the program may receive an optimal mapping for given task states, whereas the indirect training experience requires the learner to infer this mapping from the training experience; this parallels rote memorization vs. inductive learning.

Another feature of the training experience that we may manipulate is the presence of a teacher/oracle that can present information in one form or another. The teacher may feed the learner direct training examples, may answer queries (e.g., if the learner is confused by a training example, it may ask for advice from the teacher), or may be entirely absent.

The training experience is geared toward learning a target concept, which takes the form of a function mapping task states to the best response. Typically, the concept that best fits the task is not *operational*, or efficiently computable; due to this, learning programs are typically attempt to generalize to an approximation of the target concept, called a *hypothesis*. The target concept is a member of the concept class of the problem, while approximations to the *concept class* are in the *hypothesis class*

The distinction between hypothesis and concept is one of the issues of machine learning. It may be the case that the concept in question can only be represented as a database of all possible states and appropriate actions, while the hypothesis class is composed of a linear function on weighted features of the task. In this case, we can only expect a learner to find a hypothesis of this type that is hopefully correct more often than not, although it may be impossible to find a member of the hypothesis class that is 100

Once a hypothesis class is determined, the next step in designing a learning program is to select an algorithm for learning an approximation to the target concept. Many of these algorithms are similar to

methods found in operations research, such as linear and integer programming, or local search/iterative improvement algorithms from artificial intelligence.

### 2.2 Lecture 2

The second general topic that has been covered in the class thus far is concept learning, and a look at the general-to-specific ordering of hypotheses. First, we will look at a prototype of the concept learning task.

A concept learning task is defined three features $X$, $H$, and $D$. The instance space $X$ is the set of all possible task states as described by relevant features. The hypothesis class $H$ is the set of approximations to the target concept defined on the features of $X$. The training examples $D$ are examples of the target concept, composed of some instance $x \in X$ and target value of $x$.

Our goal is to find the hypothesis from $H$ that best fits the instances of $D$. We attempt to fit to $D$, rather than $X$, as $X$ is likely to be extremely large. We are able to generalize to $D$ because of the *inductive learning hypothesis*, which says that a hypothesis that approximates a target function well for sufficiently large $|D|$ will approximate the target function well over unobserved instances.

With the inclusion of one more general principle, we are able to examine two simple learning algorithms, *Find-S* and *Candidate Elimination*. The principle that we require is the idea of general-to-specific ordering of hypotheses. To do this we use the more-general-than relation, in which a hypothesis $h_j$ is *more general than* hypothesis $h_k$ if and only if $h_j$ labels positively every instance that $h_k$ labels positively. This relation induces a partial order on $H$, and allows navigation of the hypothesis space in an straightforward fashion.

The algorithms *Find-S* and *Candidate Elimination* take advantage of this ordering in order to find a the most specific appropriate hypothesis, or most expressive interval of hypotheses, respectively. The *Find-S* algorithm begins with an initial hypothesis $h$ that is the most specific hypothesis, and for every positive example $d \in D$ not accepted by $h$, it selects the next most general hypothesis that covers $d$ as the current hypothesis $h$. This yields the most specific hypothesis that is positive for all $d \in D$.

The *Candidate Elimination* algorithm behaves in a similar fashion, but instead uses two initial hypotheses, $h_g$ and $h_s$, the most general and most specific hypotheses respectively. $h_g$ and $h_s$ are the initial members of the general boundary $G$ and the specific boundary $S$ respectively; these boundaries are iteratively tightened to label correctly each $d \in D$.

Each of these algorithms, in fact, any learning algorithm must use some inductive bias in learning an approximation to a target concept. This is due to the fact that the absence of bias in a learner allows it only to learn those instances that are in $D$; it cannot label any instance that it has not seen. Bias allows the learning algorithm to extend a learned hypothesis to unobserved examples.

## 3 Important Features

In the area that has been discussed in these two lectures, perhaps the most interesting features are the selection of a hypothesis class representation, and the idea of inductive bias. We will examine the representation of a hypothesis class first.

### 3.1 Hypothesis classes

It is important to pick a hypothesis class that can yield a reasonably good approximation to the target concept. For instance, if the target concept can only be accurately represented by a high degree polynomial, it is unlikely that we will be able to accurately describe it with a hypothesis class of linear functions. Due to this, the algorithms described above cannot yield an acceptable solution. The problem arises in determining what class of functions will appropriately approximate the concept class, which can be a difficult problem,

and is highly at the programmers discretion. An interesting approach would be to use a genetic approach, beginning with several hypothesis classes, and selecting those that appear most promising during training.

### 3.2 Inductive Bias

Inductive bias is another interesting feature of our discussion thus far. As mentioned above, bias is important in that it allows us to successfully find good approximations to the target concept in many cases. However, the bias (which is a feature of the algorithm chosen for navigating the hypothesis space) may also lead to difficulty. For example, *Find-S* has the bias that the target concept is somewhere in the hypothesis space. If this is not the case, it will choose the most general hypothesis, and simply label all instances positively – an obvious failure in the majority of cases. Unfortunately, determining the bias of an algorithm is probably at least as difficult as that of adequately choosing a hypothesis class, as it may be very difficult to determine what assumptions a given algorithm makes.

## 4 Questions

There are a couple of issues that seem to bear further scrutiny, one of which is a question of definition, while the other has more reaching implications for learning algorithms.

The first question is the justification for the inductive learning hypothesis. To restate, this is the assumption that, given a hypothesis that correctly labels a sufficiently large distribution of training examples, it is likely to correctly label unobserved instances. It is obvious that it is a good guideline, and without it, it is not possible to justify the use of algorithms that learn from a restricted training set. However, the issue is not with its usefulness, but instead with the ascription of the term "hypothesis" to what appears to be a heuristic for the programmer.

The second question is regarding the choice of a learning algorithm. It seems that the majority of algorithms hinted at thus far rely on some sort of gradient descent or local search method. For instance, *Find-S* and *Candidate-Elimination* use a gradient descent method of minimizing conflicts with the training data. This begs the question, "Are there learning methods that rely on other principles?"

## 5 Research

It seems that an interesting direction for future research is the topic of noise tolerance. For example, imagine *Find-S* algorithm receives at least one training example that does not agree with the rest of the training data. Conceivably, this could eliminate the target concept from consideration, and the algorithm could no longer discover an appropriate hypothesis. In the case of *Find-S*, it might be possible to make the algorithm more robust if it had some sort of threshold method for detecting when a training example does not agree with some number of past examples, e.g. one instance occurs four times in the training example, and three of those times it is labeled positively, but the fourth it is labeled negatively. We would like the algorithm to retain this instance as a potential hypothesis, but *Find-S* in it's current condition would eliminate it.

This approach would require that a large number of training examples have already been considered; if noise is encountered early, the algorithm may not be prepared to compensate. One possible solution to this would be to train over the training set multiple times, while examining the training set in different orders. A new hypothesis could be generated each time, and the most specific of these hypotheses could then be chosen as the best approximation. It seems that this approach could increase the robustness of a simple algorithm like *Find-S*, although it would be likely to require an increase in the size of the training data proportional to the frequency of noise.

## 6 Conclusion

This paper has summarized many of the issues we have examined in the first two lectures of the Machine Learning course. Particularly, we have looked at some of the basic principles of machine learning, and some of the key concepts that allow us to formulate approaches to learning problems.