Two New Frameworks for Learning

B. K. Natarajan

CMU-RI-TR-87-25

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

November 1987

Table of Contents

Abstract

This paper presents two new formal frameworks for learning. The first framework requires the learner to approximate an unknown function, given examples for the funtion as well as some background information on it. It is shown that this framework is no more powerful than a framework that allows the learner to see examples but not background information. The second framework explores learning in the sense of improving computational efficiency as opposed to acquiring an unknown concept or function. Specifically , the framework concerns the acquisition of heuristics from examples over problem domains of special structure. A theorem is proved identifying some conditions sufficient to allow the efficient acquisition of heuristics over the aforementioned class of domains.

1. Introduction

This paper concerns learning algorithms – algorithms that construct good approximations to unknown functions from examples for those functions. The recent interest in formal methods in machine learning started with the introduction of a formal framework for concept learning in [Valiant 1984]. Since then, the framework has been extended and analyzed by numerous authors [Blumer et al. 1986, Natarajan 1987a, Kearns et al. 1987]. Unfortunately, the framework appears rather limited in scope and does not seem to capture the essence of many of the learning paradigms and architectures in use by the experimentalists. Since one of the important goals of theoretical research in machine learning is to develop a general framework for the problem, it is necessary to formulate and analyze alternative frameworks that capture the behaviour of learning models popular among workers in Artificial Intelligence. With the above in mind, this paper presents two new frameworks for learning (a) a learning framework that captures the essential ingredients of what is called a "learning architecture" in the AI literature, (b) a learning framework for the acquisition of heuristic rules as a means of improving computational efficiency. The former is a framework that provides the learning algorithm with randomly chosen examples of the function to be learned *and* some background information or "theory" about the function to be learned. It is a widely held intuition among workers in Artificial Intelligence that such a framework is strictly more powerful than the one of [Valiant 1984]. The latter is a framework specifically aimed at algorithms that construct heuristics in problem-solving domains such as symbolic integration. In analysing these two frameworks, we prove two theorems, one on each framework.

We begin by extending the results of [Blumer et al. 1986, Natarajan 1987] on the learnability of boolean-valued functions to the learnabality of general functions. To do so, we give a new and simple definition of the dimension of a family of functions and use it to prove a theorem identifying the most general class of function families that are learnable from polynomially many examples. Our results hold only for discrete domains. For continuous domains, we show how the results of [Blumer et al 1986] for boolean-valued functions may be modified to include general-valued functions. We also establish that our notion of dimension is equivalent to the more complicated Vapnik-Chervonenkis dimension of [Blumer et al.1986, Vapnik and Chervonenkis 1971]. The theorem of this section will be heavily used in the following sections and is the first of our results.

We then propose a new framework for learning, one that attempts to capture the essential ingredients of the "general learning architectures" of the experimentalists [Laird et al 1986, Mitchell et al 1986]. This is a major contribution of the paper. Specifically, the framework requires the learning algorithm to learn a function from examples for the function. The examples are picked at random by the teacher. In addition, the teacher provides the learner with some "theory" relevant to the concept to be learned, with the understanding that the concept to be learned is consistent with the "theory" presented. For instance, when teaching a concept in geometry, the teacher may present the learner with some basic theorems in geometry in addition to examples for the concept to be learned, in the hope that this would accelerate the learning process. Our main result here is a theorem stating that the class of function families learnable in this framework (i.e. from few examples and short theories) is exactly the class of families learnable in the framework of [Valiant 84] (i.e. from few examples and *no* theories). As it happens, the proof of this theorem is remarkably simple, owing to the intuitive strength of the new notion of dimension introduced in this paper. Yet, the theorem has some unintuitive consequences. Firstly, it directly implies that learning from background information and examples is no more powerful than learning from examples alone. This contradicts the beliefs prevalent in the Artificial Intelligence community. Secondly, and more subtly, the theorem leads to the realization that although background information cannot reduce the information complexity of learning, it could reduce the computational complexity of processing the information obtained from examples. This opens up a rich new area of theoretically interesting problems, one of which is stated in this paper but left open.

Finally, we develop a learning framework that explores learning as a means of improving computational efficiency rather than learning new concepts. Consider the problem of learning symbolic integration. Theoretically speaking, given a table of integrals the student should become an expert instantly. However, the student appears to need some sample problems and solutions before he develops any facility with integrals. Our framework attempts to capture the flavour of the above. Define a problem domain D on an alphabet Σ to be the pair (G,O) where G is the *goal function* (boolean valued function on Σ^*), and O is the set of *operators* (length preserving functions on Σ^*). These notions will be made precise later. An algorithm for D would take an input string x and transform it using the operators in O so that the transformed string satisfies G, if such is possible. A *meta-domain* M is simply a set of domains, and a meta-algorithm for M is an algorithm that takes as input the specification of a domain $D \in M$, calls for a small number of randomly selected examples for D, and produces as output an efficient algorithm for the domain D. To illustrate the power of the framework, we exhibit a set of domains each of which possesses a simple polynomial time algorithm. We show that although the task of computing an efficient algorithm for a domain from its specification is NP-complete for this set of domains, the task is quite tractable within our framework. We then prove a theorem identifying some conditions sufficient to allow the existence of a meta-algorithm within the proposed framework. To our knowledge, this is the first formalization where examples provide no new information to the learner, and serve only to improve the computational complexity of processing the information already possessed by the learner.

2. Preliminaries

We now describe our version of the learning framework proposed by [Valiant 1984]. We will call this Framework 1, to distinguish it from those that follow. Without loss of generality, let Σ be the binary alphabet and Σ^* the set of all binary strings. We consider functions from Σ^* to Σ^*. An example of a function f is a pair $(x, f(x))$. A *learning algorithm* is an algorithm that attempts to infer a function from examples for it. The learning algorithm has at its disposal a routine EXAMPLE, that at each call produces an example for the function to be learned. The probability that a particular example (x,y) will be produced by a call of EXAMPLE is $P(x)$, as given by the probability distribution P. Also, the probability that the learned function will be queried on a particular string x is $P(x)$. The distribution P can be arbitrary and unknown.

We define a *family* of functions F to be any set of length preserving functions from Σ^* to Σ^*. The

n^{th}-subfamily F_n of a family F, is the family of functions induced by F on Σ^n. Specifically, if $F = f_1, f_2, ... f_i, ...$, then $F_n = g_1, g_2, ... g_i, ...$ where g_i is defined as follows.

$g_i(x) = f_i(x)$ if $|x| = n$

undefined otherwise

A *basis* for F_n is a subset B_n of F such that for each $g \in F_n$, there is exactly one function $f \in B_n$ such that f and g agree on Σ^n.

Following [Valiant 1984], we say that a family of functions is learnable if there exists a uniformly convergent learning algorithm for it. Specifically, a family of functions F is *learnable* if there exists a learning algorithm that

(a)takes as input integers n and h.

(b)makes polynomially many calls of EXAMPLE, both in the adjustable error parameter h and in the problem size n. EXAMPLE produces examples of some function in F_n.

(c)For all functions f in F_n and all probability distributions P on Σ^n, with probability $(1-1/h)$ the algorithm outputs a function g in F such that

$$\sum_{x \in S} P(x) \leq 1/h$$

where

$S = \{x \mid |x| = n \text{ and } f(x) \neq g(x)\}$

We assume that the learning algorithm's output is the index of the learned function in some acceptable indexing of the functions in family F. Furthermore, if the learning algorithm runs in time polynomial in n and h, we say that the family is *polynomial-time learnable.*

The *dimension* of a sub-family F_n, denoted by $dim(F_n)$, is given by

$dim(F_n) = log(|F_n|)/(2n)$.

A family F is of dimension $D(n)$ if for all n, $dim(F_n) \leq D(n)$. If $D(n)$ is polynomial in n, we say that F is of polynomial dimension.

For any set of examples S, define the set $\Pi_F(S)$ as the set of all subsets of S obtained by intersecting S with the functions in F. i.e

$\Pi_F(S) = \{R \mid R \subseteq S$, and $\exists f \in F$ such that

f agrees with S on R

and disagrees with S on $S-R\}$.

If $\Pi_F(S) = 2^S$, we say that F *shatters* S.

Lemma 1: If F_n is of dimension d, then there exists a set of d examples that is shattered by F_n.

Proof: Omitted for brevity. Please see [Natarajan 1987b]. •

Theorem 1: A family of functions is learnable if and only if it is of polynomial dimension.

Proof: Omitted for brevity. Uses Lemma 1. Please see [Natarajan 1987b]. •

As our results above are based on information theoretic methods, it is difficult to extend them directly to continuous spaces where each example can be of infinite length. On the other hand, the results in [Blumer et al 1986] for learning boolean-valued functions are obtained using some classical results in probability theory and are valid over continuous domains. In the following, we show how to extend their results to general functions.

As in [Blumer et al. 1986], we define the *Vapnik-Chervonenkis dimension* $d_{vc}(F)$ of a family F as follows. $d_{vc}(F)$ is the smallest integer d such that no set of cardinality $d+1$ is shattered by F.

Since we no longer need the notion of a sub-family, we modify our definition of learnability accordingly. In particular, a family of functions F is learnable if there exists an algorithm that
(a)takes as input an integer h,
(b)makes polynomially many calls of EXAMPLE, polynomial in the adjustable error parameter h.
(c)as in the earlier definition of learnability.

With these definitions in hand, we can state the following theorem.

Theorem 2: For any finite alphabet Σ, a family of functions from Σ^* to Σ^* is learnable if and only if it is finite Vapnik-Chervonenkis dimension.

Proof: The proof of this theorem is similar to the proof of the corresponding theorem for boolean valued functions [Blumer et al. 1986]. •

To establish the relationship between the two measures of dimension, we have the following.

Theorem 3: For any family F
$dim(F_n) \leq d_{vc}(F_n) \leq (2n)dim(F_n)$.

Proof: Omitted for brevity. Please see [Natarajan 1987b]. •

Lastly, we give a result that attempts to introduce computational complexity into Theorem 1. Define an *ordering* of a family of functions to be an algorithm that
(a)takes as input an integer n and a set $S = \{e_1, e_2, ... e_i ..\}$ of examples such that each e_i is a pair of strings of length n.
(b)produces as output a function $f \in F$ that is consistent with S, if such exists. i.e, $(x,y) \in S$ implies $y=f(x)$.

Furthermore, if the ordering runs in time polynomial in the length of its input, we say it is a polynomial-time ordering and F is *polynomial-time orderable*.

Theorem 4: A family of functions is polynomial-time learnable if it is of polynomial dimension and is polynomial-time orderable.

Proof: Follows from that of Theorem 1. •

3. Learning Architectures

Workers in Artificial Intelligence have long sought to build general-purpose learning programs that may be used over many domains. Specifically, such programs or "architectures" take as input a description of the family of functions to be learned and after some precomputation, behave like learning algorithms for that family. We will refer to such algorithms as "learning architectures".

Consider a learning architecture M that works over a set of families G_1, G_2...,G_i.... i.e, M takes as input the description of some G_i and then behaves as a learning algorithm for G_i. Now, if $G = G_1 \cup G_2 \cup ... G_i ...$ is itself a family of low dimension, then, it follows from Theorem 1 that we can build a learning algorithm for G and not bother with the complications of M. The interesting question is whether it is possible for G to be of intractably-high dimension and yet be decomposable into G_1,....G_i... such that each G_i is of low dimension and each G_i has a short description that can be fed into the learning architecture. In order to answer this question, we consider the framework of the following section.

3.1 Learning from Examples and Background Information

We now present a learning framework that allows the learning algorithm to see examples for the function to be learned as well as some background information. We will call this Framework 2.

Let F be a family of functions. A *theory* for F is simply any total function from F to Σ^*.

A learning algorithm for F is an algorithm that attempts to infer functions in F from examples and background information. The learning algorithm has at its disposal a routine TEACHER, that is best described as the pair <EXAMPLE, T>, where EXAMPLE is the source of random examples described in Framework 1 and T is a theory for F. When attempting to teach the learning algorithm any function $f_i \in F_n$: On the first call of TEACHER, TEACHER returns $t_i \in T(f)$ where f is any function in F that agrees with f_i on Σ^n. On subsequent calls, TEACHER returns a randomly chosen example for f_i by invoking EXAMPLE recursively.

We say that a family of functions F is *learnable* in Framework 2 if there exists a learning algorithm A and a theory T for F such that

(a)A takes as input integers n, h.

(b)A makes polynomially many calls of TEACHER = <EXAMPLE,T>, polynomial in n and h. TEACHER should return a theory of length polynomial in n.

(c)For all functions f in F_n, and all probability distributions P over the examples for f, the algorithm deduces with probability $(1-1/h)$ a function g in F such that

$$\sum_{x \in S} P(x) \leq 1/h$$

where

$S = \{x \mid |x| = n \text{ and } f(x) \neq g(x)\}$

Furthermore if the learning algorithm runs in time polynomial in n and h, we say that F is *polynomial-time learnable.*

Abusing notation, we extend the theory function T to subsets of F as follows.

For $B \subseteq F, T(B) = \{T(f) | f \in B\}$.

Also, we define the inverse of a theory T to be the function T^{-} from Σ^* to subsets F as given below.

For $t \in \Sigma^*, T^{-}(t) = \{f | f \in F, T(f) = t\}$

We are now ready to state our main result.

Theorem 5: A family of functions F is learnable Framework 2 if and only if

(a)F is of polynomial dimension.

(b)F is learnable in Framework 1.

Proof: (Part(a)) (if) By Theorem 1, if F is of polynomial dimension, then F is learnable in Framework 1. Hence it is learnable in Framework 2 as Framework 1 is but a special case of Framework 2.

(only if) Let A be a learning algorithm for F in Framework 2 using a TEACHER =<EXAMPLE, T> for some theory T for F. For any n, let T_n be the set of theories offered by TEACHER over all the functions in F_n. Surely $T_n = T(B_n)$ for some basis B_n for F_n. Whatever the interpretation of the theories used by A, the set of functions A considers consistent with a theory $t_i \in T_n$ contains the set $T^{-}(t_i) \cap B_n$. Hence, if for all n, A requires polynomially many examples after seeing t_i, then by Theorem 1, $T^{-}(t_i) \cap B_n$ must be of dimension bounded by a polynomial in n. Also, the length of the theories must be bounded by a polynomial in n as A is a learning algorithm for F in Framework 2. From these two bounds and the following claim, we conclude that F is of polynomial dimension.

Claim 1: Let F_n be the n-th subfamily of a family F, B_n a basis for F_n, and T any theory for F. Let A be a learning algorithm for F with TEACHER = <EXAMPLE, T>. Then, there exists a theory $t_i \in T(B_n)$ such that

$2n dim(T^{-}(t_i) \cap B_n) + length(t_i) \geq n \cdot dim(F_n)$.

Proof:

Let $T_n = T(B_n)$.

Since $\forall f \in F, f \in T^{-}(T(f))$, we have

$B_n = \bigcup_{t \in T_n} T^{-}(t)$.

Now $|F_n| = |B_n| = |\bigcup_{T_n} T^{-}(t)|$.

Let $t = max\ \{length(t_i) \mid t_i \in T_n\}$

and $d = max\ \{dim(T^{-}(t_i)) | t_i \in T_n\}$.

Hence,

$|F_n| \leq 2^t 2^{2nd}$ and hence

$2^{2n dim(F_n)} \leq 2^{t + 2nd}$.

Which in turn implies that

$\exists\ t_i \in T_n$ such that

$length(t_i) + 2n\ dim(T^{-}(t_i)) \geq 2n\ dim(F_n)/2$

$= n dim(F_n)$

which is as required.

(Part (b)) Follows from (part (a)) and Theorem 1. This completes the proof. •.

This answers our question at the beginning of this section: If G is a family of high dimension, then G is not decomposable into component families of low dimension with short descriptions. It is the understanding of this author that it is widely believed in the Artificial Intelligence community that learning architectures can be efficiently applied to domains of intractably-high dimension [Mitchell 1987]. As we see from the above, this is not true. Does this mean that learning architectures are not very useful? No, for three reasons. The first reason is primarily of theoretical interest. Specifically, if $NP \neq RP$, there are families of functions that are polynomial time learnable in Framework 2, but not polynomial time learnable in Framework 1.

Theorem 6: If a family F is polynomial dimension, then F is polynomial-time learnable in Framework 2.

Proof: For each f_i in F_n, simply choose t_i to be the index of f_i. Since $dim(F_n)$ is polynomial in n, there exists a basis B_n for F_n such that the indices of B_n are of length polynomial in n. •

If $NP \neq RP$, then we know that there exist function families that are of polynomial dimension but are not polynomial-time learnable [Kearns et al. 1987]. Hence we have the following:

Corollary: If $NP \neq RP$, then
$\{F | F$ is p-time learnable in Framework 1$\} \not\subseteq \{F | F$ is p-time learnable in Framework 2$\}$.

The second reason is of practical interest. Let A_1 and A_2 be two polynomial time learning algorithms for a family F in Frameworks 1 and 2 respectively. Now, A_1 could run in time as little as $nh{\cdot}dim(F_n)$ on inputs (n,h) [Natarajan 1987b, Theorem 1]. A_2 could run in time $n{\cdot}dim(F_n)$ on the same input, simply by choosing the theories to be the indices of the functions as in the proof of Theorem 6. Thus, A_2 could be faster than A_1 by a factor of h, something that could be of significant practical importance.

Thirdly, in situations where the cost of obtaining an example is, bit for bit, significantly more than the cost of a comparable amount of background information, it is advantageous to use all the "theory" available. Again, this is of practical significance.

3.2 An Open Problem

First some notation: we use $>_a$, $=_a$, $<_a$ to denote asymptotically greater than, equal to and less than respectively, as relations on functions.

The last corollary prompts that we ask the following question. Is there a learning hierarchy over the complexity measure of theory length? Specifically, does there exist an infinite collection of functions $\{g_1(n), g_2(n)...g_i(n)...\}$ where $g_i >_a g_{i-1}$, such that for each g_i, there exists a family of functions that is p-time learnable with g_i long theories but not with g_{i-1} long theories? In an attempt to answer this question, we consider the following model of computation. Let $f{:}\Sigma^* \rightarrow \Sigma^*$ be a function. An algorithm A is said to compute f with $g(n)$ long theory if (a)A receives x as input and produces $f(x)$ as output.

(b)A also receives $e(f(x))$, where $e:\Sigma^* \rightarrow \Sigma^*$ is a function such that $|e(y)| \leq g(|y|)$. We call e, the theory or *advice function* and $e(f(x))$ the *advice*. We also say that A receives advice of length g.

The intent here is to provide the algorithm A with some short advice on the output string y, short compared to the length of y. (While this model may appear similar to the model of [Karp and Lipton 1980], it is quite different altogether.) We now ask whether there exists a hierarchy of functions $g_1 <_a g_2 <_a \ldots$ such that for each g_i there exists some function f that is p-time computable with g_i advice, but not with g_{i-1} advice.

Define FSAT to be the following problem.
Input: A boolean formula Φ of n variables.

Output: Any satisfying assignment for Φ.

Clearly FSAT is NP-complete. Using FSAT, we can exhibit a weak hierarchy as follows.

Claim 2: If NP$\not\trianglelefteq \bigcup_{k\geq 1} DTIME(2^{g(n^k)})$ for some $g(n) >_a logn$, then for any $r(n)$ such that $logn <_a r(n) \leq_a g(n)$, there exists a function that is computable in polynomial time with $g^{-}r(n)$ advice, but not with $r(n)$ advice. It is assumed that g is a one-one function and g^{-} is the inverse of g.

Proof:(sketch) By assumption, FSAT $\notin$ $DTIME(2^{g(n)})$ and hence is not computable with $g(n)$ advice. But surely, FSAT is computable with n advice. This proves the claim for $r(n) = g(n)$. By a simple padding argument, this can be generalized to any $r(n)$, $logn <_a r(n) \leq_a g(n)$, completing the proof. •

We can also exhibit an equally weak learning hierarchy as follows.

Claim 3: If NP$\not\trianglelefteq$ $\bigcup_{k\geq 1} RTIME(2^{g(n^k)})$ for some $g(n) >_a logn$, then for any $r(n)$ such that $logn <_a r(n) \leq_a g(n)$, there exists a family of functions that is polynomial time learnable with $g^{-}r(n)$ theory, but not with $r(n)$ theory. (Here, RTIME stands for random-time, and again g is assumed one-one.)

Proof:(sketch) Similar to the proof of the previous claim. Hinges on the result of [Kearns et al. 1987] showing the problem of ordering boolean threshold functions to be NP-complete.•

Unfortunately, the above hierarchies are rather weak, and are based on strong assumptions. While we do not have stronger results, we feel compelled to point out that this problem might be of interest from the cryptography viewpoint as well. Specifically, suppose that E were a cryptographically secure encryption function with an m-bit key. Given polynomially many examples of the form $(x,E(x))$, and $m-O(m)$ bits as advice on the key, is it possible to efficiently compute the key of an encryption function that agrees with the examples?

We close this section with a conjecture.

Conjecture: If P$\neq$NP, FSAT is not polynomial time computable with $g(n) = n-O(n)$ advice.

4. Learning as Improvement in Computational Efficiency

In this section, we develop a framework to explore learning in the sense of improving computational efficiency. This is of considerable practical importance [Mitchell 1983].

Define a *problem domain* D to be the pair (G, O), where

(a)The *goal function* $G{:}\Sigma^* \rightarrow (0,1)$ is a total function from Σ^* to $(0,1)$ computable in polynomial time.

(b)O is a finite set of *operators* $\{o_1, o_2,...\}$ where each $o_i{:}\Sigma^* \rightarrow \Sigma^*$ is a length preserving function computable in polynomial time. The operators need not be total functions.

For the problem of symbolic integration discussed in the introduction, G would simply be the rule that the expression was free of integral signs and the operator set O would be a table of standard integrals.

The *specification* of a domain $D = (G,O)$ is a set of programs for G and O that run in polynomial time. Notation: for any string x, we denote the length of x by $|x|$. We say $x \in \Sigma^*$ is *solvable* if there exists a sequence σ of operators in O (written $\sigma \in O^*$) of length $|x|$ or less such that $G(\sigma(x)) = 1$. $\sigma(x)$ is a *solution* of x and σ is a *solution sequence* of x. An algorithm for D is a deterministic program that takes as input $x \in \Sigma^*$ and computes a solution sequence for x, if such exists.

A *meta-domain* M is any set of domains such that every domain in M is defined on the same alphabet. A *meta-algorithm* for M is an algorithm that takes as input the specification of any domain $D \in M$ and computes as output an algorithm for D.

Example: Let $\Sigma = (0,1,\$)$. For any boolean function Φ of n variables, let $\Gamma(\Phi)$ denote the following function from Σ^* to $(0,1)$.

$\Gamma(\Phi)(x) = \Phi(x)$ *if* $x = y\$, y \in (0{+}1)^n$

=0 otherwise.

Let o_1, o_2 be functions from Σ^* to $(0{+}1)$ given by

$o_1(x) = x0\$y$, x of the form $x\$ay, xy \in (0{+}1)^*$,

$a \in (0{+}1)$.

$=x$ otherwise

$o_2(x) = x1\$y$, x of the form $x\$ay, xy \in (0{+}1)^*$,

$a \in (0{+}1)$.

$=x$ otherwise

Let M be the collections of all domains of the form (G,O) where $G = \Gamma(\Phi)$ for some boolean function Φ and O is the two operators defined above.

It is easy to see, that constructing an algorithm for an arbitrary domain $D \in M$ is equivalent to deciding the satisfiability of boolean formulae. Hence, if $P \neq NP$, M does not have a polynomial-time meta-algorithm. We break here for a definition. •

An *example* for a domain D is a pair (x,σ_x), $x \in \Sigma^*$, $\sigma_x \in O^k$, $k \leq |x|$ such that $G(\sigma_x(x)) = 1$.

Example:(continued) Returning to our example, we see that if the meta-algorithm were allowed to see a single example for its input domain, its task is trivial. •

The point behind the example is as follows. Given a domain D, it might be computationally intractable to compute an efficient algorithm for D, even if we knew that such existed. Yet, seeing solved examples for the input domain allows an efficient algorithm to be constructed quickly. The examples serve to improve the computational efficiency of the meta-algorithm, and hence we view this as learning in the sense of improving efficiency as opposed to concept learning.

To furnish the meta-algorithm with examples, we place at its disposal a routine EXAMPLE, similar to the one of Framework 1. At each call, EXAMPLE returns a randomly chosen example for the input domain.

We say a meta-domain M *allows heuristics* if there exists a meta-algorithm A for M such that

(a) A takes as input integers n, h and the specification of a domain $D \in M$. Let t be the least upper bound of the running time on inputs of length n of the programs in the specification of D.

(b) A computes for time polynomial in n, h, the length of its input, and t. A may call EXAMPLE, which returns examples for D, chosen according to some unknown distribution P over the solvable subset of Σ^n.

(c) For all $D \in M$ and all distributions P over Σ^n, with probability $(1-1/h)$ A outputs a program H_n that approximates an algorithm for D in the sense that

$$\sum_{x \in S} P(x) \leq 1/h$$

where

$S = \{x \mid |x|=n, H_n \text{ is incorrect on } x\}$.

(d) For any two inputs (l,h_1,D) and (m,h_2,D), $l \geq m$, let A output H_l and H_m respectively. Then

$$\frac{H_l\text{'s run time on } \Sigma^l}{H_m\text{'s run time on } \Sigma^m} \leq (l/m)^k$$

where k is a constant that depends only on D.

Conditions (a) through (c) in the definition above are as in Framework 1, and have the same purpose. Condition (d) is a uniformity condition requiring that the run time of the algorithm output by A grows polynomially with the length of the strings it is useful on.

Let $D=(G,O)$ be a domain. For each operator $o \in O$, and integer $i \geq 1$, consider the set

$U_i(o) = \{x \mid o(x)$ has a solution sequence of length $|x|-i$ or less$\}$.

We call the $U_i(o)$ the *preimages* of o in D, and we call the collection of preimages for all the operators in D the *preimages* of D.

Claim 4: For any domain D, given efficient programs to test membership in the preimages of D, we can construct an efficient algorithm for D.

Proof: Consider the following algorithm

```
Input $x$, $|x| = n$
begin
    $\sigma \leftarrow$ null-sequence ;
    for i = 1 to n do
        pick $o \in O$ such that $x \in U_i(o)$;
        if no such exists, fail;
        $x \leftarrow o(x)$;
        $\sigma \leftarrow \sigma \cdot o$;
    od
    output $\sigma$, a solution sequence for $x$.
end
```

Clearly this is an algorithm for D. •

We need one more definition before we can state the second of our main results. Let F be a family of functions from Σ^* to $\{0,1\}$ for some alphabet Σ. With any $f \in F$ we associate the set $S_f = \{x | f(x) = 1\}$. Any string $x \in \Sigma^*$ is a *positive* example for $f \in F$ if $x \in S_f$. We say that F is *well-ordered* if for any set S of strings in Σ^* such that $S \subseteq S_f$ for some $f \in F$, there exists a *least* $g \in F$ such that $S \subseteq S_g$. i.e, for all $g' \in F$, $S \subseteq S_{g'}$ implies that $S_g \subseteq S_{g'}$. An *ordering* for a well-ordered family is similar to an ordering for general families as defined in section 1, except that it takes as input a set of positive examples and outputs the least function consistent with these examples as defined above. For more details on well-ordered families, see [Natarajan 1987a].

Theorem 7: Let M be a meta-domain on an alphabet Σ. If there exists a family of functions F from $\Sigma^* \rightarrow (0,1)$ such that

(a)F contains the preimages of every domain in M,

(b)There exists a polynomial $p(n)$ such that every function in F is computed by some program that runs in time $p(n)$ on inputs of length n,

(c)F is of polynomial-dimension, well-ordered and polynomial time orderable by an algorithm A that outputs the $p(n)$-time bounded programs of (b),

then, M allows heuristics.

Proof: (sketch) Let F be a family as above and let A be ordering for it as in (c) above. We use A to construct a meta-algorithm A' for M as shown below. Essentially, the algorithm uses A to construct good approximations for the preimages of D and then uses these preimages to build an algorithm for D as in Claim 4.

Meta-Algorithm A'

```
Input $n, h, D=(G,O)$

begin
for i = 1 to n do
 for each $o \in O$ do
    Let $F_n$ be of dimension $d$.
    $m \leftarrow n(nh|O|)d$
    $S \leftarrow \emptyset$;
    for j = 1 to $m$ do
      Call EXAMPLE to obtain $(x, \sigma_x)$;
```

```
        for each decomposition of $\sigma_x$
          into $\sigma_1 o \sigma_2$, $|\sigma_2| \le |x|-i$ do
            $S \leftarrow S \cup \{\sigma_1(x)\}$;
            od
      od
      $U_i(o) \leftarrow A(S)$;
  od
od

output the following as the algorithm for $D$;

  Input $x$, $|x| = n$

  begin
      $\sigma \leftarrow$ null-sequence;
      for i = 1 to n do
            pick $o$ such that $x \in U_i(o)$
            if no such exists, fail;
            $x \leftarrow o(x)$;
            $\sigma \leftarrow \sigma \cdot o$;
      od
  output $\sigma$, a solution sequence for $x$.
  end
end
```

In the interest of brevity, we skip a formal proof that A' is a meta-algorithm for M. •

Essentially, Theorem 7 reduces the task of learning in this framework to one of learning boolean valued functions in Framework 1, and then invokes the dimensionality theorem for Framework 1. The reader should not jump to the conclusion that the role of the examples here is therefore the same as that in Framework 1. Even in the absence of examples, the specification of the domain gives the learner sufficient information to construct an algorithm for the domain. The examples serve only to speed up this computation and add no new information. Hence it is not possible here to make a distinction analagous to the distinction between polynomial-time learnability and learnability of Framework 1, a distinction that separated the information complexity of concept learning from the computational complexity. It follows that tightening Theorem 7 to an "only-if" will have to wait until the "only-if" counterpart to Theorem 4 is proved, which in turn waits for a better understanding of the relationship between NP and RP.

5. Conclusion

This paper introduced two new frameworks for learning.

The first framework concerned learning functions or concepts, allowing the learner to see both examples for the function to be learned as well some backgfround information or "theory" on it. We showed that the class of function families learnable in this framework (i.e, from few examples and short theories) is exactly the class learnable in the more established framework of [Valiant 1984] (i.e from few examples and no theories). We believe that this result will better motivate those in the Artificial Intelligence community concerned with building "learning architectures". The proof of the aforementioned result directly relates the length of a piece of information with how useful it is to the learner. Although the relationship is remarkably simple, it required the formalization of a learning framework to permit its interpretation in the context of learning from background information and examples.

The second framework concerned learning in the sense of improving computational efficiency. The framework has sufficient structure to allow a crisp analysis, yet is rich enough to capture the flavour of many practical problems. We proved a theorem identifying some conditions sufficient to allow a learning algorithm within the framework. We believe that this framework and the associated theorem are of significant practical import.

6. Acknowledgements

I thank R. Kannan, T. Mitchell and P. Tadepalli for their help and patience.

7. References

Blumer A., Ehrenfeucht, A., Haussler D., & Warmuth, M., (1986), "Classifying Learnable Geometric Concepts with the Vapnik-Chervonenkis Dimension", ACM Symposium on Theory of Computing, pp273-282.

Karp, R, and Lipton, R, (1980), "Some Connections between Nonuniform and Uniform Complexity Classes", ACM Symposium on Theory of Computing, pp302-309.

Kearns, M., Li, M., Pitt, L., and Valiant ,L.G., (1987), "On the Learnability of Boolean Formulae", ACM Symposium on Theory of Computing, pp285-295.

Mitchell, T.M., Keller, R.M., and Kedar-Cabelli, S.T., (1986), "Explanation Based Generalization: A Unifying View", *Machine Learning*, Vol 1, No 1, January.

Mitchell, T.M., (1983), "Learning and Problem Solving", International Joint Conference on Artificial Intelligence.

Mitchell, T.M., (1987), Private Communication.

Natarajan, B.K., (1987a) "On Learning Boolean Functions", ACM Symposium on Theory of Computing, pp296-304.

Natarajan, B.K., (1987b), "Learning Functions from Examples", Tech. Report, Robotics Insitute, Carnegie-Mellon U., CMU-RI-TR-87-19.

Laird, J.E., Newell, A., Rosenbloom, P.S., "Soar: An Architecture for General Intelligence", Tech. Report, Computer Science, Carnegie-Mellon U., CMU-CS-86-171.

Valiant, L.G., (1984) "A Theory of the Learnable", ACM Symposium on Theory of Computing, pp436-445.

Vapnik, V.N., and Chervonenkis, A.YA., (1971), "On the Uniform Convergence of Relative Frequencies of Events to their Probabilities", *Theory of Probability and its Applications*, vol16, No. 2, pp264-280.