On Learning from Exercises

B. K. Natarajan

CMU-RI-TR-89-4

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

February 1989

Abstract

This paper explores a new direction in the formal theory of learning – learning in the sense of improving computational efficiency as opposed to concept learning in the sense of Valiant. Specifically, the paper concerns algorithms that learn to solve problems from sample instances of the problems. We develop a general framework for such learning and study the framework over two distinct random sources of sample instances. The first source provides sample instances together with their solutions, while the second source provides unsolved instances or "exercises". We prove two theorems identifying conditions sufficient for learning over the two sources, our proofs being constructive in that they exhibit learning algorithms. To illustrate the scope of our results, we discuss their application to a program that learns to solve restricted classes of symbolic integrals.

1. Introduction

In [1], Valiant introduced a rich framework for the analysis of algorithms that learn to approximate sets from randomly chosen elements within and without the sets. This framework and its extensions has been analyzed by a number of authors, [2, 3, 4, 5] amongst others. In this paper, we present a new framework concerning algorithms that learn to solve problems approximately. instances. Early steps in this direction were taken in [4]. In a sense, this can be viewed as learning to improve computational efficiency as opposed to concept learning in the sense of Valiant. We believe that this is an important new direction in the formal theory of learning.

Consider the problem of symbolic integration. Given the definition of the problem and a standard table of integrals, we have complete information on how to solve the problem. Yet, although we are capable of solving instances of symbolic integration immediately, we are by no means efficient in our methods. It appears that we need to examine sample instances, study solutions to these instances, and based on these solutions build up a set of heuristics that will enable us to solve the problem fast. In this sense, the learning process has helped improve our computational efficiency. Similarly, given some other problem, say Rubik's cube, and the instructions concerning its solution, we would like to become proficient at it just as quickly. In essence, we would like to behave in the following manner: given the specification of a problem, we quickly learn to be efficient at solving the problem. Stated more abstractly: Consider a class of problems, such that each problem in the class is known to possess an efficient algorithm. We are interested in a meta-algorithm for the class – an algorithm that takes as input the specification of a problem drawn from the class as well as sample instances of the problem, and produces as output an efficient algorithm for the problem. As we will see, the sample instances play a crucial role in the process, as in their absence, constructing an algorithm for the input problem can be computationally intractable. In this paper, we are interested in examining learning in the aforementioned sense. Specifically, we inquire into the conditions under which such learning is possible. Our methods of analysis are probabilistic in flavour, akin to those of Valiant [1].

In Section 2, we present a formal definition of the learning framework. The framework formalizes learning in the above sense, demanding that the learner learn to solve a problem, given a source of randomly chosen solved instances of the problem. We prove a theorem identifying conditions sufficient to allow such learning. In Section 3, we consider an application of our theorem to a restricted version of symbolic integration. In particular, we show how to construct an algorithm that is capable of learning to solve such restricted classes of integrals from randomly chosen examples. In Section 4, we change the source of sample instances to one that provides unsolved instances that are chosen in a random but slightly benevolent manner. Specifically, rather than present the learning algorithm with randomly chosen solved instances of the problem, the learning algorithm is only allowed randomly chosen "exercises" on the problem – unsolved instances of the problem, chosen according to a probability distribution measuring their importance to the learner. This is very much the same as the exercises in a work-book, such as one might find at the end of a book dealing with say symbolic integration or differential equations. We are

able to prove that the conditions sufficient for learning from solved instances are sufficient for learning here as well. The proof is constructive in that we give a general learning algorithm that learns by solving the exercises, solving them in order of least difficult to most difficult. This theorem constitutes our main result.

2. Learning From Solved Instances

Let Σ be the $\{0,1\}$ boolean alphabet.

Defn: A *problem* D is the pair (G, O), where
(a)The *goal* $G:\Sigma^* \rightarrow \{0,1\}$ is 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 function computable in polynomial time.
A *specification* of a problem $D = (G,O)$ is a set of programs for G and O that run in polynomial time.

Defn: We say an instance $x \in \Sigma^*$ of a problem $D =(G,O)$ is *solvable* if there exists a sequence of operators σ such that $G(\sigma(x)) = 1$. The sequence σ is a *solution sequence* for x. The *sequence length* of a solution sequence is the number of operator applications in it, i.e., the length of $\sigma = |\sigma|$. Unless demanded by context, we use the term length to refer to the sequence length of a solution sequence. A solution sequence σ is *optimal* for x if its length is as short as that of any solution sequence for x.

Defn: Let $\sigma = p_1p_2p_3...p_t$ be a solution sequence to x, where the p_i are operators in O. We say x, $p_t(x)$, $P_{t-1}p_t(x)$,... are *steps* in the solution of x and that $p_t(x)$, $P_{t-1}p_t(x)$,... are *intermediate steps* in the solution of x. The *step-length* of σ with respect to x is the maximum of $\{|x|, |p_t(x)|, |P_{t-1}p_t(x)|, ...\}$, i.e., it is the length of the longest instance encountered in using σ to solve x.

Defn: An algorithm for a problem D is a program that takes as input a string $x \in \Sigma^*$ and produces as output a solution sequence for x, if such exists.

A *family* of problems M is simply any set of problems. We are interested in an algorithm that is useful over a family of problems, in that it is capable of learning to solve any of the problems in the family. To this end, we define the notion of a meta-algorithm for a family. Loosely speaking, a *meta-algorithm* for a family M is an algorithm that takes as input the specification of a problem D in M and attempts to construct an algorithm for D. Given the scope of our definition of a family of problems, it is easy to see that the task of the meta-algorithm will be *NP*-hard for most non-trivial families. See [4]. This is true, even if we guarantee that every problem in the family has a polynomial-time algorithm – the difficulty lies in finding such an algorithm, given the specification of the problem. In order to reduce this complexity and thereby aid the meta-algorithm in its task, we provide the meta-algorithm with sample instances of the problem specified in its input. Specifically, we consider two distinct sources of such sample instances,

one providing the meta-algorithm with randomly chosen solved instances, and the other providing unsolved instances that are randomly chosen, although in a slightly more benevolent manner than the first source. The first source is the simpler to analyze and will be the subject of the remainder of this section. The second source is considered in Section 4.

We place at the disposal of the mera-algorithm a subroutine INSTANCE which acts as a random source of solved instances. We may view INSTANCE as a black box with a button, such that at each push of the button, INSTANCE outputs a randomly chosen solved instance of the input problem D. Specifically, at each call, INSTANCE returns a pair (x,σ). The string $x \in \Sigma^*$ is randomly drawn according to an arbitrary and unknown probability distribution P on Σ^*. The operator sequence σ is a randomly chosen optimal solution sequence for x, being the null-sequence if x is not solvable or if x is solved as it is. By randomly chosen, we mean that at any stage in the solution of x, the next operator used by INSTANCE is picked randomly from among those that are useful. In order to make this precise, we need the following definition.

Defn: Let $D=(G,O)$ be a problem. For each operator $o \in O$, consider the set

$U(o) = \{x \mid \exists$ an optimal solution of the form $\sigma \cdot o$ for $x\}$

We call $U(o)$ the *projection* of o, and $U(D) = \{U(o) \mid o \in D\}$ the *projections* of D.

For any x in Σ^*, let O_x be the set of operators useful on x, i.e.,

$O_x = \{o \mid o \in O, x \in U(o)\}$.

When solving x, the first operator used by INSTANCE is picked at random from O_x. Specifically, if there are p operators in O_x, each is picked with probability $1/p$[1]. Similarly, the second operator is picked at random from O_y, where y is the result of applying the first operator to x. And so on.

With these definitions in hand, we attempt to make precise our notion of a meta-algorithm. In essence, a meta-algorithm A for a family of problems M will take as input an *error parameter* h and the specification of a problem D in M. A will then compute for time polynomial in various parameters and output a program H that efficiently approximates an algorithm for D. By this we mean that we mean that H will behave like an algorithm for D with probability $(1-1/h)$. A formal definition follows.

Defn: An algorithm A is a *meta-algorithm* for a family of problems M if there exists an integer k such that

(a) A takes as input an integer h and the specification of a problem $D \in M$. Let l be the string length of this input.

(b) A may call INSTANCE. INSTANCE returns examples for D, chosen according to some unknown distribution P over Σ^*. Let n be the longest step-length and m the longest sequence length of the solutions so provided by INSTANCE. For inputs of length n, let $t(n)$ be the sum of the running

[1] It is sufficient if each is picked with probability at least $1/poly(n)$, where $n = |x|$ and $poly(n)$ denotes a polynomial in n.

times of the programs in the specification of D. A computes in time $(lhm\iota(n))^k$, i.e., in time polynomial in the length of its input l, the error parameter h and the time required to evaluate the programs in the specification of D on the examples seen. A may be a randomized algorithm.

(c)For all $D \in M$ and all distributions P over Σ^*, with probability $(1-1/h)$ A outputs a (possibly randomized) program H that runs in time $t(n)^k$ on inputs of length n and approximates an algorithm for D in the sense that

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

where $S = \{x |$ H fails on $x\}$

Since H may be randomized, by "H fails on x", we mean that H fails to solve x with probability greater than 1/2, although x is solvable.

We now inquire into the conditions under which a family of problems posseses a meta-algorithm. Theorem 1 identifies conditions sufficient to guarantee the existence of a meta-algorithm. Necessary conditions appear to be much harder to obtain, perhaps requiring a greater understanding of learning with "advice" as explored in [4]. The statement and proof of Theorem 1 are based on previous results on learning sets with one-sided error [3]. These results are reviewed briefly in Appendix A. We refer the unfamiliar reader to that section before proceeding to the theorem.

Theorem 1: A family of problems M possesses a meta-algorithm if there exists a family of sets F such that

(a)F contains the projections of every problem D in M.

(b)F is polynomial-time learnable with one-sided error. (See Appendix A for details.)

Proof: (sketch) For a given problem D, if we can test membership in the projections of D efficiently, then we can construct an efficient algorithm for D. The following is such an algorithm.

```
input x: string;
begin
    σ ← null-sequence ;
    While G(x) ≠ 1 do
        pick o ∈ O such that x ∈ U(o);
        if no such exists, halt; ---x is not solvable---
        x ← o(x);
        σ ← o·σ;
    end
    output σ as solution for x;
end
```

The key idea in the proof is as follows: Given a problem D, the meta-algorithm will construct approximations to the projections of D using the solved instances. It will then substitute these approximations in the above algorithm to obtain an approximate algorithm for D. If the conditions of the theorem are satisfied, this can be carried out in random polynomial-time, yielding a good approximation of an algorithm for D.

The rest of the proof deals with the details. Specifically, we will exhibit a meta-algorithm for M. We

need the following definition. Let D be a problem in M. We define the quantity $I_D(n)$ to be the set of all instances in D that possess optimal solutions of step-length less than n.

$I_D(n) = \{x | x$ has an optimal solution in D of step-length at most $n\}$

When the problem D is clear from the context, we will simply write $I(n)$. Also, for $\delta \in (0,1)$ define the quantity n_δ as the least integer n such that

$$\sum_{x \in I(n)} P(x) \geq 1-\delta$$

That is, n_δ is the least integer such that the probability of occurrence of an optimal solution of step-length greater than n_δ is less than δ. In what follows, we will arrange for the meta-algorithm to learn approximations to the projections of D that are good for strings of length n_δ or less, for a value of δ that will be appropriately chosen.

Let F be a family as in the statement of the theorem. By Theorem A of Appendix A, F must possess a polynomial-time ordering Q. We use Q to construct a meta-algorithm A for M as shown below. The algorithm uses Q to construct good approximations for the projections of D and then uses these projections to build an algorithm for D.

Meta-Algorithm A_1
input h, $D=(G,O)$
Let F be of dimension $d(n)$;
Let $O = \{o_i | i = 1..k\}$;
Let $S(o_1),...S(o_k)$, $V(o_1),...V(o_k)$ be sets, initially empty;
begin
Section 1:
---This section estimates $n_{1/3h}$ with confidence $(1-1/3h)$---
call INSTANCE $3h \cdot log(3h)$ times.
Let n be the longest step-length amongst those seen.
Section 2:
---This section generates examples for projections -----
repeat $3h(kd(n))+log(3h))$ **times**
call INSTANCE to obtain (x,σ);
let σ be the sequence $o_{x_1}o_{x_2}...o_{x_r}$;
$S(o_{x_1}) \leftarrow S(o_{x_1}) \cup \{x\}$
$S(o_{x_2}) \leftarrow S(o_{x_2}) \cup \{o_{x_1}(x)\}$
.....
$S(o_{x_r}) \leftarrow S(o_{x_r}) \cup \{o_{x_{r-1}}...o_{x_1}(x)\}$
end
Section 3:
---This section constructs approximations of projections---
repeat $i=1..k$ **times**
$V(o_i) \leftarrow Q(S(o_i))$;
if Q is randomized, repeat to confidence of $1-1/3h$;
end
Section 4:
Output the following as an approximate algorithm for D
Algorithm H
input x: string;
begin
$\sigma \leftarrow$ null-sequence ;
While $G(x) \neq 1$ **do**
let $O_x = \{o | x \in V(o)\}$;
If O_x is empty **then** halt
else pick o in O_x uniformly randomly.
$x \leftarrow o(x)$;
$\sigma \leftarrow o \cdot \sigma$;
end
output σ as solution for x;
end

end

We now show that the above is indeed a meta-algorithm for M. Consider Section 1 of the algorithm. We need to show that drawing $3h \cdot log(3h)$ instances will produce a step-length m such that $n_{1/3h} \leq n$. For any single call of INSTANCE, the probability of a step-length of less than $n_{1/3h}$ occurring is $(1-1/3h)$ by definition. In t calls of INSTANCE, the probability of all the step-lengths being less than $n_{1/3h}$ is hence $(1-1/3h)^t$. We only need pick t such that

$(1-1/3h)^t \leq 1/3h$

Which inequality is satisfied by choosing $t = 3h \cdot log(3h)$.

We will consider Sections 2, 3, and 4 of the algorithm simultaneously. With respect to strings of length n or less, each set $V(o)$ can be chosen in $|F_n|$ ways in Section 3 of the algorithm. Hence, the number of distinct algorithms that can be constructed in Section 4 is $|F_n|^k$. Let S be the set of algorithms so constructible. If $n \geq n_{1/3h}$, at least one of these algorithms will approximate an algorithm for D within $1/3h$. This is because the statement of the theorem demands that F contains the projections of D. Now, the aim of Sections 2 and 3 is to eliminate those algorithms in S that are bad approximations. Consider algorithms in S that do not approximate an algorithm for D within $1/3h$. Call such algorithms "bad". The probability that a particular bad algorithm will correctly solve a randomly chosen instance is $(1-1/3h)$, and the probability that the algorithm will correctly solve all of r randomly chosen instances is $(1-1/3h)^r$. The probability that any bad algorithm in S will correctly solve r random instances is at most $|S|(1-1/3h)^r$. To eliminate all bad algorithms in S with confidence $(1-1/3h)$, we only need to make the above quantity less than $1/3h$. That is,

$|S|(1-1/3h)^r \leq 1/3h$

Since, $|S| \leq |F_n|^k$ and $|F_n| \leq 2^{d(m)}$, we have,

$2^{kd(n)}(1-1/3h)^t \leq 1/3h$

or

$r \geq 3h(kd(m) + log(3h))$.

This is exactly the number of instances employed by Sections 2 and 3 to eliminate the bad algorithms in S. Since Sections 1, 2 and 3 are each carried out to a confidence of $(1-1/3h)$, the overall confidence is $(1-1/h)$. Furthermore, the elimination of bad algorithms from S constructs an algorithm that approximates an algorithm for D within $(2/3h)$. This is so because the best approximation within S need only be within $1/3h$ owing to our choice of m, and the elimination process will construct an algorithm within $1/3h$ of this best algorithm.

In all, with probability $(1-1/h)$ the meta-algorithm constructs an algorithm for the input problem D that is within $2/3h$ in accuracy. Hence, A is a meta-algorithm for M and the theorem is proved. •

3. An Application to Symbolic Integration

In this section we discuss an application of Theorem 1 to the domain of symbolic integration. There have been reports in the AI literature of programs that learn to carry out restricted forms of symbolic integration. See [6] for instance. We will show how this can be achieved by a straightforward application of Theorem 1.

Consider the class of integrals that can be solved by the following standard integrals.

$$\int kf(x)dx = k\int f(x)dx$$

$$\int f(x)-g(x)dx = \int f(x)dx - \int g(x)dx$$

$$\int f(x)+g(x)dx = \int f(x)dx + \int g(x)dx$$

$$\int x^n dx = \frac{x^{n+1}}{n+1}$$

$$\int sinxdx = -cosx$$

$$\int cosxdx = -sinx$$

$$\int ud(v) = uv - \int vd(u)$$

Suppose we wish to construct an algorithm that can solve this class of integrals.

Consider the following grammar Γ.

prob $\rightarrow \int$ *exp var* | *d*(*exp*)

exp $\rightarrow$ *term* |*term* + *exp*| *term* – *exp* | *term* / *term* |

term *ightarrow p–term* | *p–term* * *term*

p–term $\rightarrow$ *const var* | – *term* | *trig power prob* | *exp*

power $\rightarrow$ *var* ** *term*

trig $\rightarrow$ *SIN var* | *COS var*

const $\rightarrow$ *int* | *a* | *k*

var $\rightarrow$ *x* | *y* | *z*

int *rightarrow* 1|2|3|4|5|6|7|8|9|0

This grammar generates a superset of the strings that will be seen as input to the integration algorithm. Let α be any sentential form in the grammar Γ. Define $L(\alpha)$ to be the set of strings derivable in Γ from α. That is,

$L(\alpha) = \{x | \alpha \rightarrow_{\Gamma} x\}$.

Let F be the family of all such sets, i.e.,

$F = \{L(\alpha) | \alpha$ is a sentential form in $\Gamma\}$.

It is easy to see that F is polynomial-time learnable with one sided error. To do so, we only need invoke Theorem A of Appendix A and check that (a) F is closed under intersection. We show the equivalent condition [3] that for any set of strings, there exists a "least" sentential form that generates them. By least, we mean that any other sentential form that generates these strings will be a super set of the least sentential form. To see this, given a set of strings we can efficiently compute the least sentential form that generates them as follows. Construct the parse trees for these strings in Γ, and then march up these parse trees simultaneously to pick off points common to all of them. Since the parse trees are unique in Γ, the claim follows. (b) F posseses a polynomial-time ordering. Indeed, we will exhibit a deterministic linear time ordering for F. For any set of strings, compute the least sentential form that generates them as described above. Once we have this least sentential form, it is a simple matter to

output a program that recognizes strings that can be generated from it. (c) Since the number of sentential forms of length n is at most c^n for some constant c, F is of dimension $n \cdot log(c)$.

We now hope that F contains the projections of all the standard integrals listed earlier. (To be honest, it does contain them.) We can then invoke the meta-algorithm of Theorem 1, and provide it with randomly chosen solved instances of these integrals. By Theorem 1, the output of the meta-algorithm will indeed be a good algorithm for the class of integrals in question. Tadepalli, in [4] implemented this algorithm and verified this to be the case.

4. Learning From Exercises

In the foregoing, we considered a model of learning wherein the external agent INSTANCE provided solved instances of the problem of interest. In this section, we consider a model of learning wherein the external agent provides unsolved instances of the problem of interest, although these instances are chosen a little more carefully than in the previous model. The unsolved instances are exercises, in much the same sense as those that may be found at the end of a text book on symbolic integration. Note that the exercises in the back of the book are not representative of the "natural" distribution of problem instances, but are chosen to reinforce the techniques required to solve them. In this section, we formalize the notion of learning from exercises and prove a theorem similar to that of Theorem 1.

We now replace the routine INSTANCE of the previous section with a routine EX. The key idea is to provide the learning algorithm with a source of unsolved instances of varying difficulty. This will permit the learning algorithm to consider increasingly difficult instances, improving its capabilities as it progresses. Let P be a probability distribution on Σ^*, and let INSTANCE be defined according to P as described earlier. We can best describe EX in terms of INSTANCE, as shown below. In essence, EX takes as argument an integer l and returns an instance x such that the optimal solution of x has length l. The probability that a particular instance x will be returned by any call of EX is the probability that x will be used in a solution by INSTANCE. This is a measure of the importance of knowing how to solve x, with respect to the natural distribution P.

function EX(l)
begin
    call INSTANCE to obtain (x,σ);
    if $|\sigma| < l$, output the null instance.
    else
    let $\sigma = \sigma_1\sigma_2$, where $|\sigma_1| = l$
    output $\sigma_1(x)$.
end

We now define the notion of a meta-algorithm for a family of problems in this setting. This definition is largely identical to that of Section 2, except for the use of EX instead of INSTANCE.

Defn: An algorithm A is a *meta-algorithm* for a family of problems M if there exists an integer k such that

(a)A takes as input integer h and the specification of a problem $D \in M$. Let l be the string length of this input.

(b)A may call EX. EX returns instances of D drawn according to some unknown distribution P over Σ^*. Let n be the least integer such that all the instances so produced by EX are in $I(n)$, and let m be the largest integer used as argument to EX. For inputs of length n, let the sum of the running times of the programs in the specification of D be $t(n)$. A computes for time less than $(lhmt(n))^k$, i.e., in time polynomial in the length of its input l, the error parameter h, the length m of the optimal solutions of the instances seen, and the time required to evaluate the programs in the specification of D on the instances seen. A may be a randomized algorithm.

(c)For all $D \in M$ and all distributions P over Σ^*, with probability $(1-1/h)$ A outputs a (possibly randomized) program H that runs in time $(t(r))^k$ on inputs of length r and approximates an algorithm for D in the sense that

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

where $S = \{x |$ H fails on $x\}$

Since H may be randomized, by "H fails on x", we mean that H fails to solve x with probability greater than 1/2, although x is solvable.

We now inquire into the conditions under which a family of problems possesses a meta-algorithm in this model. As it happens, the theorem we prove for this model is identical in its statement to Theorem 1.

Theorem 2: A family of problems M possesses a meta-algorithm if there exists a family of sets F such that

(a)F contains the projections of every problem D in M.

(b)F is polynomial-time learnable with one-sided error. (See Appendix A for details.)

Note that this pertains to the model wherein the meta-algorithm seeks unsolved instances from EXERCISE.

Proof: (Sketch) The key idea in this proof is similar to that of Theorem 1 – the meta-algorithm constructs approximations to the projections of D. The catch is that it must provide solutions to the instances on its own. To do so, the meta-algorithm iteratively learns to solve problems with increasingly longer solution sequences. Specifically, the meta-algorithm first learns to solve problems with solution sequences of length one. Knowing how to solve problems with solution sequences of length i, it learns to solve problems with solutions of length $i+1$. In order to describe such an algorithm, we need the following definition.

Defn: For $D \in M$ and $\delta \in (0,1)$ define the quantity m_δ to be the least integer such that

$$\sum_{x \in S} P(x) \geq 1-\delta$$

where $S = \{x | x$ has a solution of length m or less in $D\}$.

Meta-Algorithm A_2
input h, $D=(G,O)$
Let F be of dimension $d(n)$;
Let $O = \{o_i | i = 1..k\}$;
Let $S(o_1),...S(o_k)$, $V(o_1),...V(o_k)$ be sets, initially empty;
begin
Section 1:
let $\alpha = 1/4h$.
Estimate $m \geq m_\alpha$ to a confidence of $(1-\alpha)$.
Let $\varepsilon = 1/(2hm^2)$.
Estimate $n \geq n_\varepsilon$ to a confidence of $(1-\varepsilon)$.
Substitute the null sets for the $V(o)$'s in the algorithm of Section 3
to obtain the algorithm H_0.
Section 2:
for $l = 1, 2, ...,m$ **do**
pick t, such that $t/ln(t) \geq 1/\varepsilon(kd(n) + ln(1/\varepsilon)) + ln(1/\varepsilon)$
call EX(l) t times
let E be the set of instances so obtained;
for each $o \in O$ and each $x \in E$ **do**
run H_{l-1} on $o(x)$, repeating to a confidence of $(1-\varepsilon/kt)$.
if H_{l-1} solves $o(x)$ in $l-1$ steps **then**
$S(o) = S(o) \cup \{x\}$
od
for each $o \in O$ **do**
$V(o) = Q(S(o))$;
if Q is randomized, repeat to confidence of $(1-\varepsilon)$
od
construct the algorithm of section 3 using the newly computed values of
the $V(o)$'s. Call this algorithm H_l.
od

Section 3:
Algorithm H
input x: string;
begin
$\sigma \leftarrow$ null-sequence ;
While $G(x) \neq 1$ **do**
let $O_x = \{o | x \in V(o)\}$;
if O_x is empty **then** halt and report failure.
else pick o in O_x uniformly randomly.
$x \leftarrow o(x)$;
$\sigma \leftarrow o \cdot \sigma$;
end
output σ as solution for x;
end

Output H_m as an approximate algorithm for D
end

We will prove the above meta-algorithm correct in stages. First we consider Section 1. The estimation here is to be done exactly as in Section 1 of Meta-Algorithm 1, and the corresponding proof holds.

We now consider Sections 2 and 3 simultanously. We proceed by induction, with the following being our inductive hypothesis. To simpify the proof, let us assume that our estimate n for n_ε is to a confidence of unity. We will account for this at a later stage.

Inductive Hypothesis: In any run of the meta algorithm, with probability $(1-\varepsilon)^{4l}$

$$\sum_{x \in S} P_l(x) \geq (1-\varepsilon)^l \qquad eqn(1)$$

where $S = \{x | H_l \text{ is correct}^2 \text{ on } x\}$ and P_l is the conditional distribution given by

$P_l(x) = Pr\{x \text{ is produced by any call of EX}(l) \mid x \in I(n)\}$.

Basis: For $l = 0$: H_0 produces the empty sequence as solution for the set $\{x|, G(x) = 1\}$ and fails on all other inputs. Hence $\sum_{x \in S_0} P_0(x) = 1$, and the inductive hypothesis is satisfied for $l = 0$.

Induction: Assume that the inductive hypothesis is true for $(l-1)$ and prove true for l.

Let $S_l(o), S_{l-1}(o), V_l(o), V_{l-1}(o)$ represent the sets $S(o)$ and $V(o)$ for operator o at the end of iterations l and $l-1$ respectively of the outer **for** loop in the meta-algorithm. Now, consider the following algorithm.

Algorithm H^*_l.
Input x: string;
begin
  let $O_x = \{o | x \in V_l(o)\}$;
  If O_x is empty **then** halt and report failure.
  else pick o in O_x uniformly randomly.
    $x \leftarrow o(x)$.
    run H_{l-1} on x.
    If H_{l-1} solves x with solution σ
      output σo and halt.
    else report failure.
    end

H^* is different from H_l in that it uses the V_l's for deciding only on the first operator in the solution of an input instance x. After that it runs H_{l-1}. By the inductive hypothesis, H_{l-1} can be as inaccurate as $(1-\varepsilon)^l$. Hence, H^* cannot do better than that. The important thing is that it is possible to choose the $V_l(o)$'s from F so that this accuracy is attained. To see this, recall that F contains the projection of O - the $U(o)$'s. And choosing $V(o) = U(o)$ for each o will satisfy our demands. Furthermore, since the probability distribution P_l is non-zero only on instances of length n (and the null instance), it follows that we could just as well pick $V(o) = U(o) \cap \Sigma^n$. That is, we could pick $V(o)$ from F_n rather than from F.

---

[2]By this we mean that H_l solves x with probability $\geq 1/2$ if x is solvable.

We will now show how to construct good approximations to the $U(o)\cap\Sigma^n$'s so that the inductive hypothesis may stand. Consider H^*. For a given H_{l-1}, there are $|F_n|$ ways to choose each of the k sets $V_l(o)$, and hence there are at most $|F_n|^k$ choices for H^*. Call a choice "bad" if it does not satisfy eqn(1) of the inductive hypotnesis. We wish to eliminate the bad choices. To do so, we will call EX(l), so that if our current choice is bad, EX(l) will produce a witness to this with high probability. That is, EX(l) will produce an instance x such that x is not in $V_l(o)$ for any o, and yet there exists o_i such that $o_i(x)$ can be solved by H_{l-1} in $l-1$ steps. Now, at any call of EX(l), given that the call resulted in an instance $x \in I(n)$, the probability that a bad choice of H^* will be correct on the instance produced is at most $(1-\varepsilon)^l$. If we make s calls of EX(l), given that all of them resulted in instances from $I(n)$, the probability that a bad choice of H^* will be correct on all s instances is at most $(1-\varepsilon)^{ls}$. Hence, the probability that any bad choice of H^* will be correct on all s instances is bounded by $(1-\varepsilon)^{ls}|F_n|^k>$. We choose s so that the probability of the above event is at most ε. That is, we choose s so that

$(1-\varepsilon)^{ls}|F_n|^k \le \varepsilon$.

It certainly suffices to pick s to satisfy

$s \ge 1/(\varepsilon)(kd(n) + ln(1/\varepsilon))$, where $d(n)$ is the dimension of F.

But by our choice of n, the probability that any call of EX(l) will result in an instance from $I(n)$ is only $(1-\varepsilon$. Hence, we will call EX(l) t times, for some $t>s$ so that with probability $(1-\varepsilon)$, these t calls will result in at least s instances from $I(n)$. A simple Chernoff estimate yields that if t should satisfy $t/ln(t) \ge s+ln(1/\varepsilon)$. Such a choice would imply that with probability $(1-\varepsilon)^2$, we have eliminated the bad choices for H^*, i.e, with probability $(1-\varepsilon)^2$, H^* satisfies eqn(1), given that H_{l-1} satisfies eqn(1).

We also have to account for verifying these witnesses. That is, given an instance x, for each operator o, we must run H_{l-1} on $o(x)$. Since H_{l-1} is randomized, it has a certain probability of failure and this must be accounted for. To do so, we run H_{l-1} sufficiently many times so that our confidence in the result is $(1-\varepsilon/kt)$. This will require $O(ln(kt/\varepsilon))$ repetitions. Since we must run H_{l-1} on kt inputs, our simultaneous confidence in the results of all the kt computations is $(1-\varepsilon/kt)^{kt}$, which is bounded by $(1-\varepsilon)$. Finally, we note that picking a candidate $V(o)$ from F_n is done with the ordering Q, which may be randomized. We carry out this computation to a confidence of $(1-\varepsilon/k)$ for each operator O, leading to a confidence of $(1-\varepsilon/k)^k \ge (1-\varepsilon)$ for all the k operators. Combining the above estimates with the result of the last paragraph, we conclude that with probability $(1-\varepsilon)^4$, H^* satisfies eqn(1), given that H_{l-1} satisfies eqn(1). By the inductive hypothesis, H_{l-1} satisfies eqn(1) with probability $(1-\varepsilon)^{4(l-1)}$. Therefore, H^* satisfies eqn(1) with probability

$(1-\varepsilon)^{4(l-1)}(1-\varepsilon)^4 = (1-\varepsilon)^{4l}$.

Then, since $S_{l-1}(o) \subseteq S_l(o)$ for each o, it follows from the definitions of Appendix A[3]that $V_{l-1}(o) \subseteq V_l(o)$. This directly implies that the set of instances solved by H^* is a subset of the set of problems solved by H_l. Therefore, H_l satisfies the inductive hypothesis as well.

---

[3]Condition (b) of the definition of ordering Q, Appendix A.

We now seek to bound the error of H_m with respect to the natural distribution P. Specifically, we seek a lower bound on the following quantity.

$$\sum_{x \in S_m} P(x)$$

where $S_m = \{x | H_m \text{ is correct on } x\}$.

Let N be the set of instances that are not solvable.

$N = \{x | x \text{ is not solvable}\}$.

We define the following sets, parametric in l, with respect to H_l.

$X_l = \{x | x \in I(n), \text{optimal solution of } x \text{ has } l \text{ steps}, H_l \text{ solves } x\}$

$Y_l = \{x | \text{optimal solution of } x \text{ has fewer than } l \text{ steps or } x \text{ is not solvable}\}$.

$Z_l = \{x | \text{optimal solution of } x \text{ has more than } l \text{ steps}\}$.

Also, for an instance x, define the event $B(x)$ as follows.

$B(x) = \{x \text{ is an intermediate step in the solution produced by INSTANCE}\}$

Now consider the sum $\sum_{x \in S_l} P_l(x)$. We can decompose this sum as follows.

$$\sum_{x \in S_l} cP_l(x) = \sum_{x \in X_l} P(x) + \sum_{x \in X_l} Pr\{B(x)\} + \sum_{x \in Y_l} P(x).$$

In the above, c is a normalization factor to account for the fact that P_l is conditional on those instances that are in $I(n)$. By our choice of $n \geq n_\varepsilon$, (recall that we are still under the assumption that our estimate of n_ε is of confidence unity), this normalization factor satisfies $c \leq (1-\varepsilon)$. To see this, simply note that $\sum_{x \in I(n_\varepsilon)} P(x) \geq 1-\varepsilon$, by the definition of n_ε. By the definitions of $B(x)$, X_l and Z_l,

$$\sum_{x \in X_{l-1}} Pr\{B(x)\} \leq \sum_{x \in Z_l} P(x) \qquad eqn(2)$$

Therefore,

$$\sum_{x \in X_{l-1}} Pr\{B(x)\} + \sum_{x \in Y_l} P(x) \leq \sum_{x \in Z_l} P(x) + \sum_{x \in Y_l} P(x) \leq 1. \qquad eqn(3)$$

Summing $\sum_{x \in S_l} cP_l(x)$ over $l = 0,1,2...m$ and substituting eqn (3) in the sum $(m-1)$ times we obtain,

$$\sum_{l=0}^{l=m} \sum_{x \in S_l} cP_l(x) \leq \sum_{l=0}^{l=m} \sum_{x \in X_l} P(x) + \sum_{x \in X_m} Pr\{B(x)\} + (m-1) + \sum_{x \in N} P(x)$$

Using eqn(2) to replace the second term on the right, we get

$$\sum_{x \in S_l} cP_l(x) \leq \sum_{l=0}^{l=m} \sum_{x \in X_l} P(x) + \sum_{x \in Z_{m+1}} P(x) + (m-1) + \sum_{x \in N} P(x)$$

But by our choice of m, with probability $(1-\alpha)$, $\sum_{x \in Z_{m+1}} P(x) \leq \alpha$. Therefore we can rewrite our inequality thus, to hold with probability $(1-\varepsilon)$.

$$\sum_{l=0}^{l=m} \sum_{x \in S_l} cP_l(x) \leq \sum_{l=0}^{l=m} \sum_{x \in X_l} P(x) + \alpha + (m-1) + \sum_{x \in N} P(x) \qquad eqn(4)$$

Now, by the inductive hypothesis, with probability $(1-\varepsilon)^{4l}$

$$\sum_{x \in S_l} P_l(x) \geq (1-\varepsilon)^l$$

Hence,

$$\sum_{l=0}^{l=m} \sum_{x \in S_l} P_l(x) \geq m(1-\varepsilon)^m \qquad eqn(5)$$

Noting that eqn(4) and eqn(5) hold with probability $(1-\alpha)$ and probability $(1-\varepsilon)^{4m}$ respectively, we can substitute eqn(5) in eqn(4) to write: With probability $(1-\varepsilon)^{4m}(1-\alpha)$

$$cm(1-\varepsilon)^m \leq \sum_{l=0}^{l=m} \sum_{x \in X_l} P(x) + \alpha + (m-1) + \sum_{x \in N} P(x) \qquad eqn(6)$$

Grouping the first and last terms on the right hand side and substituting $c \geq (1-\varepsilon)$, we get,

$$\sum_{x \in S} P(x) \geq m(1-\varepsilon)(1-\varepsilon)^m - \alpha - (m-1) \qquad eqn(7)$$

Where $S = \{x | H_m$ is correct on $x\}$. We desire the quantity on the right hand side to be greater than $(1-1/h)$. Simplifying, we find that $\varepsilon \leq 1/(2hm^2)$ suffices.

Finally, we estimate our confidence that eqn(7) holds. Under the assumption that our estimate n for n_ε was to unit confidence, we obtained the confidence estimate of $(1-\alpha)(1-\varepsilon)^{4m}$ as noted with eqn(6). Since the confidence in our estimate of n_ε is only $(1-\varepsilon)$, the overall confidence that eqn(7) holds is $(1-\varepsilon)^{4m+2}$). We need to check whether our choice of $\varepsilon \leq 1/(2hm^2)$ is sufficient to ensure that this confidence level exceeds $(1-1/h)$. As it happens, this is the case.

We have therefore proved that A is indeed a meta-algorithm for M. •

5. Conclusion

This paper explored a new direction in the formal theory learning – algorithms that learn to solve problems from sample instances of the problems. Two random sources of sample instances are considered, one providing solved instances and the other providing unsolved instances or exercises. For both sources, general theorems are proved identifying conditions sufficient to permit learning. To illustrate the scope of these results, the are applied to the construction of an algorithm that learns to perform a restricted versions of symbolic integration.

6. References

[1] Valiant, L.G., "A Theory of the Learnable", Symposium on Theory of Computing, 1984.

[2] Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. ,"Learning Geometric Concepts and the Vapnik-Chervonenkis Dimension", Symposium on Theory of Computing, 1986.

[3] Natarajan, B.K., "On Learning Boolean Functions", Symposium on Theory of Computing, 1987.

[4] Natarajan, B.K., and Tadepalli, P., , "Two New Frameworks for Learning", Int. Conf on Machine Learning, 1988.

[5] Kearns, M., Li, M., Pitt, L., and Valiant, L.G., "On Learning Boolean Formulae", Symposium on Theory of Computing", 1987.

[6] Mitchell, T.M., Keller, R., Kedar-Cabelli, S., Machine Learning, Vol1, 1986.

Appendix A

This section reviews some necessary definitions and results on learning families of sets with one-sided error as presented in [3].

Let f denote a subset of Σ^* and F be a family (a set) of such sets.

Defn: A family of set F is polynomial-time learnable with one-sided error if there exists an algorithm A and an integer k such that

(a)A takes as input integer h, the error parameter.

(b)A may call EXAMPLE, where EXAMPLE returns randomly drawn elements of some set f in F. These elements are drawn according to an arbitrary and unknown probability distribution P on f. A computes in time $(hl)^k$, where l is the length of the longest example produced by EXAMPLE. A may be randomized.

(c)For all f in F and all probability distributions P on these sets f, with probability $(1-1/h)$ A outputs a program C that runs in time n^k on inputs of length n and accepts a set g in F such that $g \subseteq f$ and $Prob\{f-g\} \leq 1/h$.

Defn: Let $f \subseteq \Sigma^*$. For natural number n, the induced set f_n is defined by $f_n = \{x | x \in f, |x| \leq n\}$. Similarly $F_n = \{f_n | f \in F\}$.

Defn: The *dimension* of a family F is $d(n)$ if for all n, $|F_n| \leq 2^{d(n)}$. If $d(n)$ is a polynomial in n, we say F is of polynomial dimension.

Defn: An algorithm Q is said to be a *polynomial-time ordering* for family F if there exists an integer k such that

(a)Q takes as input a set of strings S. Q outputs a program C such that C accepts a set f in F, $S \subseteq f$. Also, for all g in F, $S \subseteq g$ implies $f \subseteq g$.

(b)Both Q and C run in (possibly randomized) time l^k on inputs of length l.

Theorem A: A family F is polynomial-time learnable with one-sided error if and only if F is of

polynomial dimension, F is closed under intersection, and F possesses a polynomial-time ordering.

Proof: See [3] for details. •