On Universal Learning Algorithms*

Oded Goldreich† Dana Ron‡

February 13, 1998

Abstract

We observe that there exists a universal learning algorithm that PAC-learns every concept class within complexity that is linearly related to the complexity of the best learning algorithm for this class. This observation is derived by an adaptation, to the learning context, of Levin's proof of the existence of optimal algorithms for NP.

*Presented in the Impromptu Session of *COLT96*.

†Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, ISRAEL. E-mail: oded@wisdom.weizmann.ac.il. On sabbatical leave at LCS, MIT.

‡Laboratory for Computer Science, MIT, Cambridge 02139, USA. E-mail: danar@theory.lcs.mit.edu. Research was supported in part by an NSF Postdoctoral Fellowship.

1 Introduction

In his seminal paper on NP-completeness [9], in addition to proving (independently of Cook [4] and Karp [6]) the existence of NP-complete problems, Levin presented an optimal algorithm for solving any NP-complete problem. That is,

Theorem 1 ([9] – see Appendix for terminology): *For every NP-relation,* R*, there exists an algorithm* A *that solves* R*, and a polynomial* $p(\cdot)$ *so that for every* A′ *that solves* R*, there exists a constant* c *such that for every* $x \in \mathcal{L}(\mathrm{R})$*:* $\mathrm{time}_{\mathrm{A}}(x) < c \cdot \mathrm{time}_{\mathrm{A}'}(x) + p(|x|)$.

More than a decade later, the same underlying idea was used by Levin to prove the following:

Proposition 2 [10]: *There exists a polynomial-time computable function* F *that is one-way, unless no one-way functions exist.* (The function F is explicitly given!)

Here we employ the same idea to prove

Theorem 3 (a universal learning algorithm – informal version): *There exists a universal learning algorithm* U*, so that for any concept class* $\mathcal{C}$*, if* $\mathcal{C}$ *can be* PAC*-learned in polynomial-time then* U PAC*-learns* $\mathcal{C}$ *in polynomial-time.*

We wish to stress that this result has no practical significance since the hidden constants are huge (see Theorem 4 and its proof). On the other hand, the result holds also for many extensions of the PAC-learning model [12] such as learning with queries [1], learning under a fixed distribution (e.g., [3]), learning with statistical queries [7], etc. In general, the result holds in any model that enables efficient hypothesis testing (i.e., given a hypothesis h one should determine whether h approximates the target concept within a given approximation parameter).

2 Formal Setting

2.1 Definition of the PAC learning model

We first define distance between boolean functions. Let f and f' be two boolean functions over some domain X, and let D be a distribution over X. We define the *distance* between f and f' with respect to D as follows:

$$\mathrm{dist}_{\mathrm{D}}(f, f') \stackrel{\mathrm{def}}{=} \Pr_{x \sim \mathrm{D}}[f(x) \neq f'(x)] \ .$$

Let $\mathcal{C} = \{\mathcal{C}_n\}$ be any class of boolean functions (referred to as the *concept class*), where the domain of functions in $\mathcal{C}_n$ is $\{0,1\}^n$. Similarly, let $\mathcal{H} = \{\mathcal{H}_n\}$ be a class of boolean function, which we refer to as the *hypothesis* class. (In particular, $\mathcal{H}$ may be $\mathcal{C}$.) A PAC Learning Algorithm for the class $\mathcal{C}$ using $\mathcal{H}$ is given a source of examples distributed according to an arbitrary unknown distribution D_n over $\{0,1\}^n$, and labeled according to an unknown *target* concept $c \in \mathcal{C}_n$. In addition, the learning algorithm is given a *confidence* parameter $0 < \delta \leq 1/2$, and an *error* parameter $0 < \epsilon \leq 1$.

We require of the learning algorithm that for any choice of input dimension n, distribution D_n and target c, and for any setting of δ and ϵ, the algorithm output a hypothesis function $h \in \mathcal{H}_n$ such that with probability $1-\delta$ over the random examples the algorithm observes, $\mathrm{dist}_{\mathrm{D}_n}(h, c) \leq \epsilon$. The distance between h and c is also referred to as the *error* of h (with respect to c and D_n). The sample complexity and running time of the learning algorithm are measured as functions of n, $1/\delta$, $1/\epsilon$, and size(c), where size(c) is the size of the representation of c in a standard encoding of

concepts in $\mathcal{C}$. We say that a class $\mathcal{C}$ is *efficiently learnable* if it can be learned in time polynomial in the above parameters using a *polynomially evaluatable* hypothesis class $\mathcal{H}$. A function class is polynomially evaluatable if there exists a polynomial-time procedure for computing $h(x)$ for every n, $h \in \mathcal{H}_n$ and $x \in \{0,1\}^n$.

An additional issue that we need to address is the representation of hypotheses. As stated above, a learning algorithm is defined with respect to a particular hypothesis class $\mathcal{H}$, and hence the hypothesis it outputs is assumed to be represented in some standard encoding of functions in $\mathcal{H}$. Since different learning algorithms may use different hypothesis classes, we ask that learning algorithms also output an evaluation algorithm $\text{EVAL}_{\mathcal{H}}$ for the hypothesis class they use, together with their hypothesis. Namely, for any dimension n, hypothesis $h \in \mathcal{H}_n$, and input $x \in \{0,1\}^n$, given n, h and x as input, the algorithm $\text{EVAL}_{\mathcal{H}}$ outputs $h(x)$. Thus we make the hypothesis class (and its corresponding evaluation algorithm) explicit instead of implicit. Note that this only requires of a "conventional" learning algorithm (using a particular hypothesis class $\mathcal{H}$) to tag-on to its output a fixed string (encoding $\text{EVAL}_{\mathcal{H}}$).

2.2 Main Result

A formal statement of Theorem 3 follows. This theorem is stated in the PAC-learning model as defined above. Extensions are discussed later. In this theorem we bound the expected running time of the universal learning algorithm. We note that for any k, the probability that our algorithm does not halt after time k times its expected running time, is at most δ^k.

Theorem 4 (a universal learning algorithm – formal version): *There exists a universal learning algorithm* U*, so that for any concept class* $\mathcal{C} = \{\mathcal{C}_n\}$*, if some algorithm* PAC*-learns* $\mathcal{C}$ *using hypothesis class* $\mathcal{H} = \{\mathcal{H}_n\}$ *in time* $t(n, \text{size}(c), \epsilon, \delta)$*, and using expected sample size* $s(n, \text{size}(c, \epsilon, \delta)$*, then algorithm* U PAC*-learns* $\mathcal{C}$ *in expected time* $\text{O}(t(n, \text{size}(c), \epsilon/2, \delta/2) + \frac{\log(1/\delta)}{\epsilon} \cdot t_{\text{EVAL}_{\mathcal{H}}}(n))$*, and using sample size* $\text{O}(s(n, \text{size}(c), \epsilon/2, \delta/2) + \frac{\log(1/\delta)}{\epsilon})$*, where* $t_{\text{EVAL}_{\mathcal{H}}}$ *is the running time (as a function of* n*) of an evaluation algorithm for the hypothesis class* $\mathcal{H}$*.*

We remark that for any "non-trivial" concept class[1] $\mathcal{C}$, any PAC-learning algorithm must observe $\Omega(\frac{\log(1/\delta)}{\epsilon})$ many examples [2]. Therefore, $s(\cdot, \cdot, \epsilon, \delta) > \frac{\log(1/\delta)}{\epsilon}$ and the additive $\frac{\log(1/\delta)}{\epsilon}$ term in the sample complexity can be omitted. Furthermore, it is typically the case that the time to evaluate a hypothesis on a given input is much smaller compared to the time it takes to construct the hypothesis. Hence the second term in the time complexity of the universal algorithm is usually negligible.

2.3 Overview of Proof

The basic idea in the proof of Theorem 1 is to simply try all possible algorithms, "in parallel", and rely on the fact that it is easy to recognize a correct solution. Here we follow the same idea.

A technical problem is that there exist infinitely many potential algorithms. Following Levin, we classify algorithms according to their length (w.r.t. any standard encoding of algorithms) and emulate an additional step of an algorithm of length ℓ only after we've emulated many more steps of all algorithms of length $\ell - 1$. More precisely, the universal learning algorithm, U, proceeds in *rounds*. In the j^{th} round U utilizes only algorithms of length at most $j - 2\log_2 j$. For every integer $\ell \leq j - 2\log_2 j$, algorithm U allows each algorithm of length ℓ to make $\lfloor \frac{2^j}{\ell^2 \cdot 2^\ell} \rfloor$ additional steps.

[1] A concept class is *trivial* if it consists of a single concept or of two disjoint concepts whose union equals the instance space.

But what does U do with all these algorithms? This depends on the specific learning model. For sake of the presentation we first consider *exact* learning with *equivalence queries* [1], which we define below. In Section 2.4 we show how the universal algorithm for this model can be modified so that it works in the PAC-learning model. In the exact-learning model a learning algorithm is required to output a hypothesis h that is exactly correct with respect to the target function c, i.e., $h \equiv c$. Furthermore, the learning algorithm is required to succeed with probability 1. In the course of learning, the algorithm is allowed access to an equivalence oracle which answers queries of the form: "is (a particular hypothesis) h equivalent to c?".[2]

Given access to an equivalence oracle, whenever an (emulated) algorithm A halts with a hypothesis, the universal algorithm U checks this hypothesis using an equivalence query. (This may not be needed since A might have checked its output by itself.) If the hypothesis is correct, then U outputs this hypothesis and halts. Otherwise, U continues emulating the other algorithms.

We first observe that, for every concept class $\mathcal{C}$, if some algorithm A exactly learns $\mathcal{C}$ then so does U. The reason being that eventually U will emulate sufficiently many steps of A, (where A, by definition, always halts with a correct hypothesis), whereas wrong hypotheses output by other algorithms emulated by U will be rejected by U.

The question is how many steps does U perform until it fully emulates a "good" algorithm for the concept class $\mathcal{C}$. We first observe that the time consumed by U in round j is

$$\sum_{\ell=1}^{\lfloor j-2\log_2 j\rfloor} 2^\ell \cdot \left\lfloor \frac{2^j}{\ell^2 \cdot 2^\ell} \right\rfloor \quad < \quad 2^j \cdot \sum_{\ell=1}^{\infty} \frac{1}{\ell^2} \quad < \quad 2 \cdot 2^j$$

During this round each algorithm of length ℓ was allowed to make $\lfloor \frac{2^j}{\ell^2 \cdot 2^\ell} \rfloor > \frac{2^j}{2\ell^2 \cdot 2^\ell}$ additional steps. Thus, each such algorithm is slowed down by at most a factor of $4\ell^2 \cdot 2^\ell$ (once it starts being emulated). That is, to start emulating a particular algorithm A of length ℓ, the universal algorithm U needs to reach the first stage j such that $j - 2\log_2 j \geq \ell$. This takes $2^{\ell + O(\log \ell)}$ time. From this stage on, to emulate T steps of algorithm A, we need $4\ell^2 2^\ell \cdot T$ steps of algorithm U. It follows that in order to fully emulate a good algorithm A for $\mathcal{C}$, the universal algorithm runs c_A times longer, where c_A is a constant depending on A. This constant is admittedly huge (i.e., it is exponential in the length of the description of A). Still it is a constant and so Theorem 4 follows in a model allowing equivalence queries.

Remark: The sequence $\langle 1/\ell^2 \rangle$ used above can be replaced by any sequence $\langle a_\ell \rangle$ which is easy to compute and has a sum bounded by a constant. Similarly, the sequence $\langle 1/4i^2 \rangle$ used below can be replaced by any sequence $\langle b_i \rangle$ which is easy to compute and has a sum bounded by 1/2.

2.4 Proof of Theorem 4

In this section we show how to modify the universal algorithm described in the previous section so that it works in the PAC model. Instead of being given access to an equivalence oracle, here the algorithm is given access to random examples labeled by the target concept, and is required to output a hypothesis that is only approximately correct (i.e., has error at most ϵ) with respect to the target concept c (and the underlying distribution on examples). Furthermore, the algorithm is allowed to fail (that is, output a hypothesis having large error) with probability δ (were this probability is taken over the choice of random examples).

[2] An equivalence oracle is usually assumed to return a *counter-example* in case $h \not\equiv c$. Thus the oracle we define is sometimes referred to as a *weak* equivalence oracle.

As in the previous section, the universal algorithm proceeds in rounds, where in the j^{th} round it allows each algorithm of length ℓ to perform $\lfloor \frac{2^j}{\ell^2 \cdot 2^\ell} \rfloor$ steps. All emulated algorithms are given $\delta/2$ and $\epsilon/2$ as input (and random examples as required). Since the universal algorithm does not have access to an equivalence oracle, it needs an alternative way to test the correctness of hypotheses output by the emulated algorithms. It is well-known (and easy to verify by applying a multiplicative Chernoff bound), that one can test with confidence δ whether a given (single) hypothesis has error greater than ϵ or at most $\epsilon/2$, using $O(\frac{\log(1/\delta}{\epsilon})$ random examples. This is simply done by evaluating the hypothesis on this many examples and comparing the values computed to the labels of the target concept. If the hypothesis differs from the target on at most $\frac{3}{4}\epsilon$ of the examples then it is accepted by the test, otherwise it is rejected.

One difficulty which arises is that the universal algorithm is allowed to fail only with probability δ while it might need to test *many* hypotheses, where the number of hypotheses is not even known in advance. To overcome this difficulty we vary the confidence in the different tests performed so that the accumulated probability that *some* test returns an incorrect answer (i.e., "accept" when the error is greater than ϵ, or "reject" when the error is at most $\epsilon/2$), is bounded by δ. In particular, we set the confidence of the i^{th} test to $\delta/(4i^2)$. (The i^{th} test is the one associated with the i^{th} algorithm, see details below.)

There are two additional questions that we still need to answer. The first question is how do we account for the time required to test the hypotheses output by the emulated algorithms. The second question is how do we deal with the fact that even a "good" learning algorithm might fail with probability $\delta/2$ (and similarly, that an approximately correct hypothesis might be rejected), in which case the universal algorithm could continue indefinitely. The solution for the first problem is to *augment* each algorithm being emulated so that *it* tests its output hypothesis with the adequate confidence parameter. Details follow.

Consider an enumeration of all possible learning algorithms according to their length. Let A_i denote the i^{th} algorithm (i.e., its length is $\log_2 i - 1$). We augment A_i so that before outputting a hypothesis h it tests h with approximation parameter $\epsilon/4$ and confidence parameter $\delta/(4i^2)$. This means that the running time of A_i is increased by an additive factor of $O(\frac{\log(4i^2/\delta)}{\epsilon} \cdot t_{\text{EVAL}_{\mathcal{H}}})$. Thus, if the running time of a PAC-learning algorithm A for $\mathcal{C}$ is $t_{\mathrm{A}} = t(n, \text{size}(c), \epsilon, \delta)$, then by applying the argument described in Section 2.3 we get that A (together with its augmentation) is fully emulated by U in $c_{\mathrm{A}} \cdot \left(t_{\mathrm{A}} + c'_{\mathrm{A}} \cdot \frac{\log(1/\delta)}{\epsilon} \cdot t_{\text{EVAL}_{\mathcal{H}}} \right)$ steps where c_{A} and c'_{A} are constants depending on A.

In order to solve the second problem, whenever an emulated algorithm outputs a hypothesis that fails the (augmented) test, or simply does not output a hypothesis, the algorithm is returned to the pool of emulated algorithms, and is restarted. However, in its "re-incarnation" it is augmented by a test that runs with a confidence parameter that is half the previous value. Thus, since $\delta \leq 1/2$, the expected number of times that a good algorithm is emulated before it outputs an approximately correct hypothesis and this hypothesis passes the test, is at most 2. Furthermore, the probability that a good algorithm does not output an approximately correct hypothesis after it is emulated k times is δ^k. On the other hand, the probability that the universal algorithm outputs a hypothesis with error greater than ϵ is still bounded by δ. Theorem 4 follows. ■

Remark: Enumerating all possible algorithms should not be confused with enumerating all possible hypotheses (e.g., as done explicitly in [11]).

2.5 Extensions

As stated in the introduction, Theorem 4 applies also to many extensions and variants of the basic PAC-learning model provided that these extensions/variants allow efficient single-hypothesis-testing. Models in which the condition holds include learning with equivalence queries [1] (which we described in Section 2.3), learning under a fixed distribution (e.g., [3]), learning with statistical queries [7], etc.

Theorem 4 also holds with respect to proper (representation dependent) learning, provided that membership in the hypothesis class can be easily decided. The reason for this additional condition is that the universal algorithm should check not only that the hypothesis (output by an emulated algorithm) is probably approximately correct but also that it belongs to the hypothesis class. This introduces an additive term into the running-time of the universal algorithm.

Theorem 4 also holds with respect to other (single) complexity measures such as query complexity. The construction of the universal algorithm has to be modified so that in the j^{th} round each algorithm of length ℓ is allowed complexity $\frac{2^j}{\ell^2 \cdot 2^\ell}$ (according to the measure of interest).

3 Discussion

Universality versus Optimality: Theorem 4 demonstrates that universality does not have to come at the cost of efficiency. The universal learning algorithm can be used to learn any learnable concept class and it is optimal up to a constant factor. That is, any algorithm which is tailored for a specific concept class cannot perform much better (on this class!) than the universal algorithm.

Abuse of Asymptotic Analysis: Theorem 4 abuses the paradigm of asymptotic analysis. The constant hidden in its O-notation is not only huge (in any reasonable application of the theorem), but is rather not fully specified. That is, the theorem as well as its proof do not specify the constant which relates the running-time of a good algorithm for the class and the running-time of the universal algorithm when learning this class. Instead, this constant depends on the encoding of such an unknown good algorithm.

References

[1] D. Angluin. Queries and Concept Learning. *Machine Learning*, 2(4):319–342, 1988.

[2] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4):929–965, 1989.

[3] G. M. Benedek and A. Itai. Learnability with Respect to Fixed Distributions. *Theoretical Computer Science*, 86(2):377–389, 1991.

[4] S. A. Cook. The Complexity of Theorem-Proving Procedures. In *3rd STOC*, pages 151–158, 1971.

[5] A. Ehrenfeucht, D. Haussler, M. J. Kearns and L. G. Valiant. A General Lower Bound on the Number of Examples Needed for Learning. *Information and Computation*, 82(3):247–251, 1989.

[6] R.M. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, (Raymond E. Miller and James W. Thatcher, eds.), Plenum Press, pages 85–103, 1972.

[7] M. J. Kearns. Efficient noise-tolerant learning from statistical queries. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 392–401, 1993.

[8] M. J. Kearns and U.V. Vazirani. *An introduction to Computational Learning Theory.* MIT Press, 1994.

[9] L. Levin. Universal'nyĭe perebornyĭe zadachi (Universal Search Problems: in Russian). *Problemy Peredachi Informatsii*, 9 (3), pages 265–266, 1973.

[10] L. Levin. One-Way Function and Pseudorandom Generators. *Combinatorica*, 7 (4), pages 357–363, 1987.

[11] N. Linial, Y. Mansour, and R. L. Rivest. Results on Learnability and the Vapnik-Chervonenkis Dimension. *Information and Computation*, 90(1):33–49, 1991.

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

A Details on Theorem 1

Theorem 1 refers to the following notions and notations:

- An NP-relation is a polynomial-time recognizable set of pairs, $\mathrm{R} \subseteq \{0,1\}^* \times \{0,1\}^*$, so that $(x,y) \in \mathrm{R}$ implies that $|y| = \mathrm{poly}(|x|)$. The corresponding NP-language is $\mathcal{L}(\mathrm{R}) \stackrel{\mathrm{def}}{=} \{x : \exists y \text{ s.t. } (x,y) \in \mathrm{R}\}$.
- An algorithm A is said to solve R if, for every $x \in \mathcal{L}(\mathrm{R})$, on input x algorithm A outputs y such that $(x,y) \in \mathrm{R}$.
- The running-time of algorithm A on input x is denoted $\mathrm{time}_{\mathrm{A}}(x)$.

The additional $\mathrm{poly}(|x|)$ term mentioned in the theorem can be eliminated if one postulates that every solver must "check" its output by running a fixed (polynomial-time) decision procedure for R.

A proof of Theorem 1 can be easily derived from Section 2.