On Universal Learning Algorithms*

Oded Goldreich[†] Dana Ron[‡]

July 5, 1996

Abstract

We observe that there exists a universal learning algorithm which PAC-learns every concept class within complexity which is linearly related to the complexity of the best learning algorithm for this class. This observation is derived by a straightforward 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, which solves R, and a polynomial $p(\cdot)$ so that for every A' which solves R, there exists a constant c such that for every $x \in \mathcal{L}(R)$:* $\text{time}_A(x) < c \cdot \text{time}_{A'}(x) + p(|x|)$.

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 which is one-way, unless no one-way functions exist.* (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. Furthermore, if some algorithm PAC-learns $\mathcal{C}$ in time $t(\cdot)$ then U PAC-learns $\mathcal{C}$ in time $O(t(\cdot))$.*

We wish to stress that this result has no practical significance since the constant hidden in the O-notation is huge (see the 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 which 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

A formal statement of Theorem 3 follows. We first stress that by PAC-learning we mean the distribution-free model of learning from examples as introduced by Valiant in [12]. (Extensions will be discussed later.) Another two changes with respect to Theorem 3 are the introduction of the additive $\log(1/\delta)/\epsilon$ term (which can be eliminated in most cases– see below) and the specification of the (standard) parameters on which the time complexity depends (cf., [8]).

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 $\mathcal{C}$ in time $t(n, \text{size}(c), \epsilon, \delta)$ then U PAC-learns $\mathcal{C}$ in time $O(t(n, \text{size}(c), \epsilon, \delta) + \frac{\log(1/\delta)}{\epsilon})$, where n, $\text{size}(c)$, ϵ and δ are the usual dimension of the instance space, size of the target concept $c \in \mathcal{C}_n$, approximation parameter and confidence parameter, respectively.*

We remark that for any "non-trivial" concept class[1] $\mathcal{C}$, any PAC-learning algorithm must take $\frac{\log(1/\delta)}{\epsilon}$ many examples [2], and so $t(\cdot, \cdot, \epsilon, \delta) > \frac{\log(1/\delta)}{\epsilon}$ and the additive $\frac{\log(1/\delta)}{\epsilon}$ term can be omitted.

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

2.1 Proof in a model allowing Weak Equivalence Queries

The basic idea in the proof of Theorem 1 is to just 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 $\ell - 1$.

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 $\ell \leq j - 2\log_2 j$, algorithm U allows each algorithm of length ℓ to make $\frac{2^j}{\ell^2 \cdot 2^\ell}$ additional steps.

But what does U do with all these parallel learning algorithms? This depends on the specific learning model. For sake of simplicity we first consider a model of learning-from-examples augmented by *weak equivalence queries*. In this model, one may determine in unit cost whether a given hypothesis approximates the target concept up to a given approximation parameter. In this model, whenever an (emulated) algorithm halts with an hypothesis, we check this hypothesis using an equivalence query. (This may not be needed since the emulated algorithm might have checked its output by itself.) Thus, the universal algorithm halts with a good hypothesis whenever any algorithm halts doing so.

We first observe that, for every concept class $\mathcal{C}$, if some algorithm A learns $\mathcal{C}$ then so does U. The reason being that eventually U will emulate sufficiently many steps of A, whereas wrong hypotheses output by other algorithms emulated by U will be rejected by U (since it tests each hypothesis output by any emulated algorithm). The question is how does the complexity of U relate to the complexity of a "good" algorithm for the concept class $\mathcal{C}$. We first observe that the time consumed by U in round j is

$$\sum_{\ell=1}^{j-2\log_2 j} 2^\ell \cdot \frac{2^j}{\ell^2 \cdot 2^\ell} < 2 \cdot 2^j$$

During this round each algorithm of length ℓ was allowed to make $\frac{1}{\ell^2 \cdot 2^\ell} \cdot 2^j$ additional steps. Thus, each such algorithm is slowed down by only a factor of $2\ell^2 \cdot 2^\ell$. That is, to emulate T steps of a particular algorithm of length ℓ we need $2\ell^2 2^\ell \cdot T$ steps of algorithm U. It follows that T steps of a good algorithm A for $\mathcal{C}$ are emulated by $c_A \cdot T$ steps of U, where c_A is a constant depending on A. This constant is admitably 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 weak 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 $\langle 1/2i^2 \rangle$ 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 Managing without Equivalence Queries

It is well-known that equivalence queries can be simulated by examples, yet we have to be slightly more careful here since the number of equivalence queries used by our universal algorithm is not a-priori known (akin the running-time of the universal algorithm on a specific concept class). Thus, whenever we implement an equivalence query for the universal algorithm we set the allowed error probability so that the sum of all errors incurred is bounded; for example, if we are allowed error δ then we may implement the i^{th} equivalence query while allowing for error $\delta/2i^2$.

The above analysis of running-time did not account for the time required to test the hypotheses output by the emulated algorithms. In case we are allowed weak equivalence queries (as assumed above), testing each hypothesis takes unit time and can thus be ignored. Otherwise, we need to implement a test which rejects, with probability at least $1-\delta'$, any given (single!) hypothesis which deviates by more than ϵ from the target concept. Such a test can be implemented in time $\frac{\log(1/\delta')}{\epsilon}$. Recall however that our invocations of the hypothesis tester require δ' to be smaller than the confidence parameter δ given to algorithm U. A host of minor technical problems arises and they are all resolved by augmenting each algorithm being emulated so that it tests its output hypothesis with an adequate confidence parameter. Details follows.

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 ϵ and confidence parameter $\delta/2i^2$, where ϵ and δ are the corresponding parameters given to algorithm U. This means that the running time of A_i is increased by an additive factor of $\frac{\log(2i^2/\delta)}{\epsilon}$. Thus, we need to replace T in the above running-time bound by $T + \frac{\log(c_A^2/\delta)}{\epsilon}$. Theorem 4 follows. ■

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

2.3 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. That is, given a hypothesis h one should determine, within time $O(\frac{\log(1/\delta)}{\epsilon})$, whether h approximates the target concept. Models in which the condition holds include learning with (membership and/or equivalence) queries [1], 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 sample complexity and 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.

Appendix: Details on Theorem 1

Theorem 1 refers to the following notions and notations:

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

The additional $\text{poly}(|x|)$ 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.