PAC-LEARNING IS UNDECIDABLE

A PREPRINT

Sairaam Venkatraman* **S Balasubramanian** **R Raghunatha Sarma**

Department of Mathematics and Computer Science,

Sri Sathya Sai Institute of Higher Learning, Andhra Pradesh, India

August 21, 2018

ABSTRACT

The problem of attempting to learn the mapping between data and labels is the crux of any machine learning task. It is, therefore, of interest to the machine learning community on practical as well as theoretical counts to consider the existence of a test or criterion for deciding the feasibility of attempting to learn. We investigate the existence of such a criterion in the setting of PAC-learning, basing the feasibility solely on whether the mapping to be learnt lends itself to approximation by a given class of hypothesis functions. We show that no such criterion exists, exposing a fundamental limitation in the decidability of learning. In other words, we prove that testing for PAC-learnability is undecidable in the Turing sense. We also briefly discuss some of the probable implications of this result to the current practice of machine learning.

***K*eywords** PAC-learning, Turing Machines, Machine Learning

1 Introduction

The existence of a criterion, preferably efficient in time and space, for deciding whether a function is learnable by a given family of hypotheses can prove to be useful to both practitioners and users of machine learning. For example, such a criterion can help make crucial decisions based on the feasibility of applying machine learning to problems of interest, potentially saving time and resources. Furthermore, if such a criterion could also allow for a comparison of learning algorithms and models, it would only be all the more useful.

In order that a meaningful investigation into the existence of the intuitive aforementioned criterion be made, we must assume a particular, defined model for learnability and use a formal, mathematical framework for studying the problem of learning. In this study, we restrict ourselves to the well-known PAC definition of learning [1] and use the mathematics of Turing machines to treat the problem of learnability in a systematic manner.

The use of Turing machines in this setting is by no means accidental; showing that no Turing machine can decide the learnability of every (computable) function can be seen as evidence of the non-existence of any algorithm to do the same by use of the Church-Turing thesis [2]. Further, this undecidability can also be seen as proof of fundamental limitations in learning, similar to the limitation on computing proved by the existence of incomputable functions.

We now give a brief outline of the article. The next section mentions some of the work in the literature that shares similarity with this article in philosophy and scope. We then introduce concepts necessary to the development of the central result of this paper in the subsequent section. We then state and prove the main undecidability result. We conclude with a discussion on a few hypothesized implications of our result.

*vsairaam@sssihl.edu.in

2 Related Work

In this section, we mention and briefly discuss a few works in the literature that bear some resemblance to our central result. The first, of course, is the undecidability of the halting problem [3]. The halting problem, simply put, is the problem of determining from the description of a program and an input, if the program would halt on that input. More accurately, the halting problem asks whether a given Turing machine would halt on a specific input. The halting problem has been proved to be undecidable in the general case and has been dealt with at length in several books, for example [4]. Our result is inspired from this theorem and can be seen as its analogue for learning computable functions.

There are other undecidability results such as that of testing for the equivalence of Turing machines and some others pertaining to the properties of the languages generated by formal grammars. These can be derived from the halting problem. Since we use the framework of Turing machines, we can obtain such results by following a similar line of reasoning as in these corollaries. However, we do not actively pursue these, and instead focus on only the undecidability of testing for learnability.

A result that establishes a surprising restriction in the field of (unsupervised) learning is by Kleinberg in [5]. In his article, he establishes that it is impossible for a clustering algorithm to satisfy, simultaneously, three intuitive and desirable properties. While this is not an undecidability, it still exposes some bounds on what is possible in learning.

3 Preliminaries

In this section, we introduce several concepts and definitions that are necessary to develop the central theorem.

3.1 PAC-Learnability

Before a formal definition is given, it would do to present an informal description and motivation for PAC-learning. The setting we assume for learning reflects what is the general understanding of (supervised) machine learning; we seek to approximate (or learn) the true, unknown mapping between data and labels using a family of functions that defines the hypothesis model and a finite, labeled training set. Learning may be seen as a search in the family of hypothesis functions we use, say, neural networks to find a function that is deemed to be the best approximator of the true mapping.

Various models of learning can be defined based on the demands we make on the learning process. One notion would be to require that the learning algorithm always outputs a function that exactly matches the true mapping. However, this is impractical on at least two counts: multiple consistent hypotheses and possible misleading training examples [6]. A more practical and relaxed model would be to assume that the learner must simply output an approximate hypothesis most of the time. In other words, we require that the learning algorithm returns a hypothesis that is *probably, approximately correct* (PAC). With this informal description, the definition is as follows.

Consider a class C of functions, an algorithm L and a family of functions H that we call the concept class, the learner and the hypotheses class, respectively. Let X be the common domain for all the functions in C and H. Let $err_D(c, h)$ denote the true error, defined as the probability of $c(x) \neq h(x)$ when x is sampled from X using distribution D. Further, let M be a function that takes a tuple of numbers belonging to $[0, 1]$ and returns a whole number.

PAC-learnability: C is said to PAC-learnable by L with respect to H and M if the following is true. $\forall\ c \in C$, distributions D over X, ϵ and δ such that $0 < \epsilon < \frac{1}{2}$ and $0 < \delta < 1$, given $M(\epsilon, \delta)$ examples sampled from D of the form $(x, c(x))$ as input, L outputs $h \in H$, with probability at least 1 - δ, such that $err_D(c, h) \leq \epsilon$.

We note that this definition is different from the usual definition of PAC-learning in that no restriction is placed on the number of examples or on the running time of the learner. Thus, the only requirement is that the learner be able to output an approximation of c most of the time, as opposed to doing this efficiently. As we shall see, even under this relaxation, testing for learning is impossible.

3.2 Turing Machines

In the definition above, it was assumed that the learner L denoted an algorithm, a concept that is well-understood and used widely. However, for our purposes, a more formal definition is required. Further, as the objective is to study

the decidability of PAC-learning, a formal model for a decision making engine is required. In other words, we need a sufficiently expressive mathematical machine whose internals are open to specification, yet whose input and output behaviour are known. As Turing machines and their probabilistic counterparts satisfy these requirements, they shall be used as formal model for the criterion and the learner, respectively.

Informally, a Turing machine is a mathematical abstraction of a computing machine. It consists of an input/output (infinite) tape, a finite alphabet and a finite set of states the Turing machine can be in. The functioning of a Turing machine is governed by a transition function that decides the course of action on a particular input and the current state. These actions can be: writing to the tape, reading the 'next' or 'previous' alphabet or halting. The halting state can be further qualified by being either 'accepting' or 'rejecting'. Under the assumption of an accepting (or halting) state, we can define the language accepted by a Turing machine as being the set of strings from its alphabet that guide its computation to an accepting (or halting) state.

Turing machines have been extended to the so-called probabilistic Turing machines (PTM) to represent randomized computation. The key difference between the formulations of regular Turing machines and PTMs is that the transition function is probabilistic in the case of PTMs. Thus, a PTM can have several computational paths for same input, each with its own probability.

The expressivity of Turing machines may be summarized by the Church-Turing thesis that, informally put, posits that any computation can be transferred to an equivalent computation by a Turing machine. Further, any Turing machine may be seen as a PTM with its transition function 'making moves' with probability either 0 or 1.

The knowledgeable reader is reminded of the distinction between PTMs and Non Deterministic Turing Machines (NDTM). While both PTMs and NDTMs use non-deterministic transitions and are both equivalent to Turing machines from a computability point of view, there are several fundamental differences. In NDTMs, the transitions are modelled by a relation instead of a function as was presented earlier. Thus, an NDTM could move to from one configuration to any one of a finite set of configurations. Thus, an NDTM can have multiple computational paths for a single input. Informally, an NDTM is said to accept an input if even one of these paths reach the accept state. However, there is no notion of probabilistic acceptance or rejection. NDTMs merely use non-determinism as a powerful way to explore multiple computational paths to find an accepting path, if it exists. Thus, the non-determinism is only in the path search and not does not play a conceptual role in the acceptance or rejection of an input.

A property of importance to our theorem is the fact that Turing machines can be encoded as strings of symbols. A description of this process may be found in[4]. The same can be extended to PTMs quite easily. Thus, we may now treat Turing machines (and PTMs) as finite strings that can be given as input to another, 'universal' Turing machine (PTM) U. Using the encoding of a Turing machine, U can replicate its behaviour on any given input. More formal definitions and details of the above concepts may be found in [4],[7] and [8].

3.3 Turing-PAC learning

In this section, we shall describe the reformulation of PAC-learning in terms of Turing machines. We assume that C - the family of concept functions that we seek to learn and H - the family of hypothesis functions are comprised of functions that are each computable by a PTM and a Turing machine, respectively. We also assume that the true error of a hypothesis with respect a concept function and a distribution is also Turing-computable. These are quite reasonable assumptions and is based on the hypothesis that any computation can be represented by a Turing machine. An important point to note is that in practice we may not actually know the form of the concept class or the true error. What we do know is that (based on our assumptions) they can all be represented by Turing machines. This knowledge is sufficient.

In the first definition of PAC-learning, we had defined L to be an algorithm that conducts a search in the hypothesis space. In practice, this search process is usually done by gradient-based methods. Most of the search processes in practice have some degree of randomness involved by, say, choosing a set of arbitrary initial parameters. Based on this observation, we decide to model the learner as a PTM. In the next few lines, we give an relatively informal, yet sufficiently detailed description of this PTM and its working.

Let $\tau = (\epsilon, \delta, D, M, H)$, where $0 < \epsilon < \frac{1}{2}$, $0 < \delta < 1$, D is a distribution over some fixed set X, M is a function from $[0,1] \times [0,1]$ to the set of natural numbers and H is a family of Turing-computable functions. The learner L_τ is a PTM that works as follows. L_τ takes as input "c" - the PTM-encoding of a PTM-computable function c that is defined over

X. L_τ then generates $M(\epsilon, \delta)$ tuples of the form $(x, c(x))$ by sampling from the distribution D. It then performs some finite computation (that depends on the algorithm used) to return "h" - the encoding of a function in H. Both "c" and "h" are then passed to a Turing machine that computes $err_D(c, h)$. If this Turing machine returns a representation of a number that is less than ϵ, L_τ moves to the reject state, otherwise L_τ moves to the accept state. L_τ is guaranteed to either reject or accept the input based on our assumptions on c, H, err_D and the finiteness of the search procedure.

Based on this description of a learner, we can now formally define the Turing version of PAC learning.

Turing PAC Learnable: We say that C, a family of PTM-computable functions defined over domain X is Turing-PAC learnable by a learner L_τ, where τ is as defined above, if $\forall\ 0 < \epsilon < \frac{1}{2}$, $0 < \delta < 1$, distributions D over X, L_τ accepts "c" with probability at least 1 - δ.

Note that in the above formulation, the dependence on H and M have been captured by τ. With this final definition, we now move to the central result of our paper.

4 Main Result

Based on the definition of Turing-PAC learnability (TPAC), the existence of a test for learnability depends on our ability to decide, given a number $p \in [0,1]$, if the learner L_τ would accept with probability p. Our theorem shall show that this task is impossible.

We use the notation 'p-accepts' to denote that L_τ accepts with probability p and the notation 'at least p-accepts' to denote that L_τ accepts with probability at least p. We shall also use the common notation of 'recursive' to qualify a language that is computable or 'decided' by a Turing machine. A language X is said to be decided by a Turing machine M if and only if the following holds. $x \in X$ if and only if x is accepted by M[4].

Theorem 1. *Let H be a family of Turing computable functions and c be another Turing computable function. Let X be the common domain of c and every function in H. Let D be a fixed distribution over X and M be a function from $[0, 1] \times [0, 1]$ to the set of natural numbers. Let L_τ be a learner as defined previously. Then, H_τ = {"L_τ""c":L_τ at least 1 - δ accepts "c"} is not recursive.*

Proof. Let us assume, by way of contradiction, that H_τ is indeed recursive. Then, there exists a Turing machine M_0 that accepts H_τ. By this assumption, $H_{1,\tau}$ = { "L_τ": L_τ at least 1 - δ accepts "L_τ"} is also recursive. This is true as we can consider M_1 that operates as follows. M_1 takes "L_τ" as input and replicates it once. M_1 then passes this as input to M_0. Since M_0 decides H_τ, M_1 decides $H_{1,\tau}$.

We now use the fact that the recursive property is preserved by complementation. Thus, $H_{2,\tau}$ - the complement of $H_{1,\tau}$ is also recursive and thus, there exists M_2 that decides $H_{2,\tau}$. We now make the observation that as M_2 is Turing machine, it can also be reformulated as a PTM and encoded as such. Let "M_2" denote the PTM-encoding of M_2. Further, "M_2" either belongs to $H_{2,\tau}$ or it does not.

First, we see that $H_{2,\tau}$ = {w:w is not a PTM-encoding or w = "M" $\ni M$ accepts "M" with a probability less than 1 - δ}. Thus, we see that M_2 accepts "M_2" if and only if "M_2" $\in H_{2,\tau}$ if and only if M_2 does not at accept "M_2" with probability at least 1 - δ if and only if M_2 does not accept "M_2" with probability 1 if and only if M_2 does not accept "M_2". The penultimate in the chain of equivalences is due to the fact that M_2 is actually a regular Turing machine and, therefore, except for one computational path it takes, all of its paths have probability 0.

Thus, we have shown that M_2 accepts "M_2" if and only if it does not accept "M_2". This, of course is a contradiction. Thus, we cannot find a Turing machine that can decide "$H_{2,\tau}$". This means that $H_{2,\tau}$ is not recursive. Therefore, $H_{1,\tau}$ is also not recursive. This implies that H_τ is not recursive. □

With this result and the definition of TPAC, we see that it is not possible to have a universal test for learnability.

5 Implications and Extensions

In this section, we present a brief discussion on a few of several possible implications of, as well as extensions to, our result. Given the interest in deep neural networks today and their remarkable performance on several tasks, it would be of some importance to consider our result in connection to them.

Deep neural networks are provably universal approximators, for example [9] and [10]. Thus, any concept function can be approximated to an arbitrarily high degree by them. However, our result still holds even if the family of hypothesis was fixed; the proof is virtually the same. In other words, the undecidability is inherent to the testing process. Regardless of our choice of the learner and the hypothesis family, no universal criterion for learnability can be found.

Despite the above, practice has shown that some family of hypothesis outperform other families consistently. Thus, more generally, it would be useful if a universal comparator for learners be studied. Such a comparator could decide, for a specific concept function, which learner would be more suitable.

Another area of progress in research is the use of transfer learning. Transfer learning allows for practitioners of machine learning to use pre-trained models to act as feature extractors for a different task. This offers the advantage of allowing for better performance using lesser computation. However, little is understood theoretically about the feasibility or validity of applying transfer learning in practical scenaios. Thus, it would be useful to consider a criterion for deciding this. A formal PAC-style model has been proposed in [11]. Based on our result, we hypothesize that no such criterion would exist. However, we have considered only hard criteria in our work. An investigation into the existence of a universal, probabilistic criterion for both learnability as well as transfer-learning might yield interesting and positive results.

As a last remark and a possible extension, we propose that we can achieve decidability by restricting the type of concept classes. We hypothesize that a family of 'useful' and 'practical' concept functions may allow for decidability. One such class can be the family of compositional functions. These are interesting not least because deep learning methods are thought to learn these relations. Further, many concept classes such as facial identity, document structure etc. show a compositional structure. Thus, a positive result for such a family could have some use for practical applications.

6 Conclusions

Our work has used the mathematics of Turing machines and PTMs to study the PAC model for learnability. We reformulated the original PAC model in terms of Turing machines in order to study the decidability of testing for learnability. The rationale for using such a model was based on the expressivity of Turing machines, based on the Church-turing thesis. Further, Turing machines also allow for a formal study of notions such as decidability. We showed that, in general, testing for TPAC-learnability is undecidable. Thus, we ruled out the possibility of a universal, hard criterion for learning.

We then discussed a few extensions and made some hypotheses based on our undecidability. We discussed the undecidability with respect to universal approximators and hypothesized the existence of a universal comparator for learners. Further, we proposed a theoretical study of transfer learning along the lines of this article. Lastly, we conjectured that by restricting the family of concept functions we could achieve decidability.

References

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

[2] Alan Church. Abstract no. 204. *Bull. Amer. Math. Soc*, 41:332–333, 1935.

[3] Alan M Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

[4] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1997.

[5] Jon M Kleinberg. An impossibility theorem for clustering. In *Advances in neural information processing systems*, pages 463–470, 2003.

[6] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997.

[7] Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996.

[8] https://www.encyclopediaofmath.org/index.php/Probabilistic_Turing_machine, 2018. [Online; accessed 28-July-2018].

[9] Nicolas Le Roux and Yoshua Bengio. Deep belief networks are compact universal approximators. *Neural Computation*, 22:2192–2207, 2010.

[10] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[11] Tomer Galanti, Lior Wolf, and Tamir Hazan. A theoretical framework for deep transfer learning. *Information and Inference: A Journal of the IMA*, 5(2):159–209, 2016.