What does learning mean?

Jeffrey Heinz

October 22, 2024

1 Programs and Algorithms

A procedure, or a program, is a well-defined set of steps one can follow to produce an answer to some question. The question is called the *input* to the program and the answer is called the *output*. In colloquiual use, the term "algorithm" is synonmous with procedure or program. However, I find it useful to think of an algorithm as something more than a procedure. *An algortihm is a procedure which solves a problem.*

A problem is a map from instances of the problem to a solution. The problems themselves could be stated informally as in the following examples.

1. What is the value of $x + y$?
2. What is the list obtained by sorting the elements of the list x?
3. Given a maze M, will execution of the strategy S allow one to exit the maze?
4. What grammar generated the finite set of strings D?

An instance of the problem is a specification of the variables. For example an instance of the first problem is given by letting $x = 4$ and $y = 5$. An instance of the second problem is given by considering $x = [2, 9, 3, 5, 3, 1, 7]]$. An instance of the third problem is given by letting M equal the Labyrinth of Versailles and letting the strategy S be "While touching the wall with your right hand, follow it." An instance of the last problem is given by $D = \{baba, bababa, abba\}$.

Formalizing the notion of a problem means considering the *range* of values that the variables can take. The range of values determines the *instance space* of the problem.

1. Let x, y be any integers.
2. Let x be any list of integers.
3. Let M be ..., let S be ...
4. Let D be any finite set of strings.

What are possible mazes M? What are possible strategies S?

An algorithm is a procedure which provably solves a problem. This means for *any instance* ι of the problem, execution of the algorithm on ι leads to the correct answer.

It is relatively easy to come up with procedures and programs that do things. It is more difficult to come up with algorithms. To come up with an algorithm, one needs to think about the problem one is trying to solve. This means thinking about the how to formulate the problem, and the range of values the variables in the problem can take on. One must then show that the algorithm you are proposing correctly solves any instance of the defined problem.

With that in mind, what problems do learning algorithms solve?

2 A First Approximation

THe figure illustrates a general conception of learning over time. Suppose there is a language L describable with grammar G. Let us explain the notation in the figure. The notation

Time t	1	2	3	4	...	n	...
Evidence at time t	$t(1)$	$t(2)$	$t(3)$	$t(4)$	...	$t(n)$	...
Input to Learning Algorithm at time t	t_1	t_2	t_3	t_4	...	t_n	...
Output of Learning Algorithm at time t	G_1	G_2	G_3	G_4	...	G_n	...

Figure 1: A schema for learning

$t(n)$ means the evidence presented at time n. This notation suggests that evidence can be understood as a function with domain $\mathbb{N}$.

The notation t_n refers to the sequence of evidence up to the nth one. For example, t_3 means the finite sequence $t(1), t(2), t(3)$. In mathematics, angle brackets are sometimes used to denote sequences so some would write this sequence as $\langle t(1), t(2), t(3)\rangle$.

The notation G_n refers to the program output by the algorithm with input t_n. If A is the algorithm and n is any point in time, we can write $A(t_n) = G_n$.

1. What constitutes evidence?
2. What constitutes grammars?
3. Call the sequence $t(1), t(2), \ldots$ a data presentation. What constitutes a data presentation?
4. If this process unfolds over time, what constitutes success in the limit?
5. If this process unfolds over time, what relation holds at time n between G_n and the target G?
6. Is it enough if the algorithm succeeds for the one grammar G?

Answering these questions is important to define learning problems.

3 Computational Learning Theory

There is not a single definition of “learning”. Rather, there are several, and each provides a window of insight into any proposed learning system. Despite seminal work and insights from the past, developing useful learning criteria is ongoing, important endeavor. This document is informed by much prior research including those discussed by [1, 2, 3, 4, 5, 6, 7, 8].

Generally, learning algorithms can be thought of as functions that relation *data presentations* to grammars. Grammars themselves can be thought of as representations of functions that classify strings, or assign structures to strings. The set of functions represented by these those output grammatical representations are called the *concept class*. A formalization will be offered below, but here I provide a broad view of what computational learning theories are about and what the main conclusions are.

In general, it is desirable to understand the behavior of learning systems. This includes being able to ascertain something about the quality of the output, usually in relation to something about the data presentation it received as input. It also includes something about the resources, in time and space, that the algorithm consumes in various scenarios, such as the worst case, the “typical” case, and so on.

Generally, we want algorithms that are feasible (only consume “reasonably” much resources), and that can return outputs across a broad spectrum of possibilities (can learn any pattern). The main results of computational learning theory are that the broadest criteria of “what learning means” reveals it is not possible to have both. Algorithms that have been shown to learn any pattern in principle cannot do so feasibly.

When a computational problem is shown to be intractable, a common approach is to identify tractable sub-problems or related problems. There are at least two ways of constraining computational learning problems.

One is to constrain the *concept class* of targets. Instead of trying to learn “any pattern,” what if we just tried to learn patterns from a specific class C?

Another approach is to modify the kinds of *data presentations* that the learning criteria asks learning algorithms to succeed on. One form of modification is to *provide additional information* in the presentation. For example, it is known that, in some in some classification tasks that allowing both positive and negative examples – as opposed to allowing only positive examples – facilitates learning. Another kind of modification is to *reduce which presentations* algorithms are expected to succeed on. For example, it is known that requiring learners to succeed on representative data presentations which are computable – as opposed to any logically possible representative data presentation, also facilitates learning.

Even though there are these two general approaches, to my knowledge the only non-trivial results which lead to *feasible learning* of interesting concept classes are ones that constrain the concept class. Approaches that change the learning problem by adjusting what constitutes an *admissible* data presentation have been used to show that any *computable* language can be learned, but those learning procedures are not feasible. Apart from some trivial instances of the learning problem, it remains an open question how to obtain feasible learning of broad classes by constraining the data presentations.

Regarding the specific problem of humans learning language, these results imply that constraining the concept class is *necessary* for feasible learning. In addition, it is an open question how constraining which data presentations are admissible can facilitate this learning problem (and others).

References

[1] Daniel Osherson, Scott Weinstein, and Michael Stob. *Systems that Learn.* MIT Press, Cambridge, MA, 1986.

[2] M. Anthony and N. Biggs. *Computational Learning Theory.* Cambridge University Press, 1992.

[3] Michael Kearns and Umesh Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, 1994.

[4] Vladimir Vapnik. *The nature of statistical learning theory.* Springer, New York, 1995.

[5] Sanjay Jain, Daniel Osherson, James S. Royer, and Arun Sharma. *Systems That Learn: An Introduction to Learning Theory (Learning, Development and Conceptual Change).* The MIT Press, 2nd edition, 1999.

[6] Partha Niyogi. *The Computational Nature of Language Learning and Evolution.* Cambridge, MA: MIT Press, 2006.

[7] Alexander Clark and Shalom Lappin. *Linguistic Nativism and the Poverty of the Stimulus.* Wiley-Blackwell, 2011.

[8] Meyhar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning.* MIT Press, 2nd edition, 2018.