Computational Learning Theory

For CGSC270

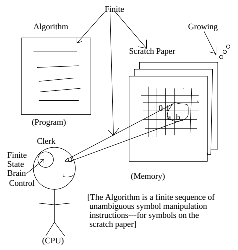
By
John Case
Department of Computer and Information Sciences
101A Smith Hall
University of Delaware
Newark, DE 19716

Email: case@cis.udel.edu
Url: http://www.cis.udel.edu/∼case

These slides at url:
http://www.cis.udel.edu/∼case/slides/colt-handout.ps

More on computational learning theory at url:
http://www.cis.udel.edu/∼case/colt.html

Turing's Analysis of **Human** Clerical Computation



Turing's Analysis [Cas92]:

- Provides a model or **mathematical definition** of **computable** (by human clerk or otherwise).
- The model is **mathematically provably equivalent** to what can be accomplished by modern electronic digital computers.
- Makes it possible to **prove mathematical theorems** about the **limits of computability**.*
- Combined with mathematical definitions of **computational learning** (later slides), makes it possible to **prove mathematical theorems** about the **limits of computational learning** (human and artificially intelligent).

*In high school geometry, one **proves theorems** about triangles, circles and the like; in **computability theory** one proves theorems about which things are computable, how hard are they, and which things are not computable!

Kind of Learning Generally Considered in Computational Learning Theory

- **Not**: memorizing lists, tables, phone books, ... kind of learning.
- **Yes**: learning a **concept** from positive and/or negative examples. E.g., in infancy/childhood we see animals which are birds and those which are not; before the end of childhood we have learned (subconscious) rules so that, shown a picture of such animals as wrens, robins, and elephants, we can correctly say which are birds and which are not. Our adult-level learned rules also work* on penguins, ostriches, kiwis, bats, and house flies. (◡̈)

*But more slowly as measured by human reaction speed.

- **Yes**: learning a grammar for a language from positive examples of that language (as we do subconsciously in infancy/early childhood).
- **Yes**: learning a rule (or program) for generating a sequence of data (or a function). E.g., we easily learn a rule for generating infinite sequences such as the following (if we are shown sufficiently much of them).

$$1, 3, 5, 7, \ldots \qquad (1)$$

$$3, 4, 7, 12, \ldots \qquad (2)$$

  - **A** rule for Sequence 1: generate successive odd numbers.
  - **A** rule for Sequence 2: start with 3 and add successive odd numbers.

A General Model of Computational Learning [Cas86,OSW86]*

$$p_0, p_1, p_2, \ldots, | \, p_t, \ldots \overset{\textbf{Out}}{\leftarrow} M \overset{\textbf{In}}{\leftarrow} \text{data re } O$$

- O is a computable object such as a data sequence, a function, or a language.
- M is an algorithmic device/machine for learning.
- M, fed data re O, outputs a succession of rules, programs, or grammars $p_0, p_1, p_2, \ldots, p_t, \ \ldots$.
- M is *successful at learning* O means, for **some** t, the rules, programs, or grammars p_t **and beyond** all compute O.

*For **important** "special cases" see also [KV94].

Example M

An M for successfully learning rules for a **wide** class of numerical data sequences including those from Slide 5:

Look for iterated forward differences to become (apparently) constant, then output a rule which uses the (apparent) constant to extrapolate the sequence.

Example Runs of **this** M

Data:	1		3		5		7		**?**
Diffs:		2		2		2			

2 is the apparent constant, so the rule that M outputs is: generate the level 0 sequence, where, at level 0, start with 1; add the apparent constant 2 from level 1 to get successive level 0 data items. This rule generates the data sequence $1, 3, 5, 7, 9, 11, 13, 15, \ldots$.

Data:	3		4		7		12		**?**
1st Diffs:		1		3		5			
2nd Diffs:			2		2				

2 is the apparent constant **after two differencings**, so **this** rule that M outputs is: generate the level 0 sequence, where, at level 0, start with 3; at level 1, start with 1; add 2 from level 2 to get successive level 1 data items; **finally**, add successive level 1 items to get successive level 0 data items. This rule generates the data sequence $3, 4, 7, 12, 19, 28, \ \ldots$.

Data:	3		4		7		18		**?**
1st Diffs:		1		3		11			
2nd Diffs:			2		8				
3rd Diffs:				6					

6 is the apparent constant after **three** differencings, so **this** rule that M outputs is: generate the level 0 sequence, where, at level 0, start with 3; at level 1, start with 1; at level 2, start with 2; add the apparent constant 6 from level 3 to get successive level 2 data items; add successive level 2 items to get successive level 1 data items; **finally**, add successive level 1 items to get successive level 0 data items. This rule generates the data sequence $3, 4, 7, 18, 43, 88, \ \ldots$.

The examples on the just previous slide illustrate that, with **new** data, **sometimes** M may have to output a new rule.

The data sequence $1, 3, 5, 7, 9, 11, 13, 15, \ldots$, is generated by the polynomial

$$(2x + 1), \qquad (3)$$

for $x = 0, 1, 2, 3, 4, 5, 6, 7, \ldots$.

The data sequence $3, 4, 7, 12, 19, 28, \ \ldots$ is generated by the polynomial

$$(x^2 + 3), \qquad (4)$$

for $x = 0, 1, 2, 3, 4, 5 \ldots$.

The data sequence $3, 4, 7, 18, 43, 88, \ \ldots$ is generated by the polynomial

$$(x^2 + 3) + x(x - 1)(x - 2), \qquad (5)$$

for $x = 0, 1, 2, 3, 4, 5 \ldots$.

It turns out (a theorem): our **example** "differencing" M is successful on all and only the polynomials [Hil56]. It fails on $1, 2, 4, 8, 16, 32, 64, 128, \ \ldots$, which is generated by the **non**-polynomial

$$2^x, \qquad (6)$$

for $x = 0, 1, 2, 3, 4, 5, 6, 7, \ldots$.

One **can** handle such exponentials, but this requires using a **different** learner M'. **Furthermore**, one can combine the M for polynomials and the M' for exponentials to get an M'' (more sophisticated than M and M') that is successful on exactly polynomials **and** exponentials!

M'' fails on $1, 2, 4, 8, 61, 23, 46, 821, \ \ldots$, which is generated by reversing the decimal digits in the sequence generated by Formula 6 just above, for $x = 0, 1, 2, 3, 4, 5, 6, 7, \ldots$. (◡̈)

An even more sophisticated learner M''' can handle all that M'' can **and** the decimal digit reversals of all that M'' can ... .

Is there an end to this? Can one always combine learners to get even better learners? See next slide!

Some Theorems from Computational Learning Theory [Cas86,OSW86]*

Theorem (Barzdin & Blums '70s) *There are learning machines M_1 and M_2 which can each successfully learn classes of infinite, computable numerical sequences, but* **NO** *learning machine exists which combines the learning ability of M_1 and M_2 to learn every sequence either of them can!*

Hence, just as in the biological world, there is a need for **diversity** in types of learning machines—sometimes one type will succeed, sometimes another (and one can't know which in advance).

Corollary (Gold '67) NO *learning machine exists which can successfully learn* **every** *infinite, computable numerical sequence!*

This corollary makes computational learning theory non-trivial—there is no universally successful learning machine for the entire class of infinite, computable numerical sequences.

*They are about **people** too if people are machines.

Physicists sometimes employ rules/theories for **approximately** extrapolating data from experiments, e.g., the classical "explanation" for anomalous dispersion in physical optics. Can suitably clever learning machines learn more infinite, computable data sequences if their outputs can make a few mistakes? Lets say that a machine M learns a sequence $S = s_0, s_1, s_2, \ \ldots$ **successfully**^{n} just in case, on input S it **eventually** outputs nothing but rules/programs/... which compute this sequence **except** perhaps for up to n data items s_i (where they are allowed to fail).

Theorem (Case& Smith '78) *Suppose n is a non-negative integer. Then there is a learning machine M which can successfully^{n+1} learn* **some** *class of infinite, computable numerical sequences, but* **NO** *learning machine can successfully^n learn this same class!*

N.B. The example machine M^1 described next successfully^1 learns a class of sequences $\mathcal{C}^1$, but **NO** machine successfully^0 learns $\mathcal{C}^1$.*

*N.B. successful^0 is the same as just plain successful.

Example M^1:

Numerically code the (potential) sequence-generating programs in some standard programming language.

Let M^1, on a numerical data sequence $S = s_0, s_1, s_2, \ \ldots$ look for the first **non**-zero data item, s_i, and, if it finds such s_i, it outputs the rule/program whose code number is s_i.

Let $\mathcal{C}^1$ be the class of all infinite, computable numerical data sequences $S = s_0, s_1, s_2, \ \ldots$ such that S has a first non-zero data item s_i and s_i codes a rule/program which generates S itself **except** that it may fail to compute one of the data elements in S. So, each S in $\mathcal{C}^1$ looks like $0, 0, 0, 0, \ \ldots, s_i, \ \ldots$, where $s_i \neq 0$ and s_i codes a rule for generating S **or** a rule which almost does, but it messes up on computing one of the data elements. Clearly, M^1 successfully^1 learns each S in $\mathcal{C}^1$.

Should each rule/program conjectured by a learner M be consistent with the data seen by M at the point it makes the conjecture? The obvious answer would **seem** to be Yes. Yet we have:

Theorem (Barzdin & Blums '70s) *There is a learning machine M which can successfully learn* **some** *class $\mathcal{C}$ of infinite, computable numerical sequences, but* **NO** *learning machine, which both* **eventually** *outputs a single, final conjecture on each sequence in $\mathcal{C}$* **and** *which* **never** *outputs a conjecture which contradicts the data on which the conjecture is based, can successfully learn $\mathcal{C}$!*

Let $\mathcal{C}$ be the class of all infinite, computable numerical data sequences $S = s_0, s_1, s_2, \ \ldots$ such that S has a first non-zero data item s_i and s_i codes a rule/program which **correctly** generates S itself. Clearly, M^1 from Slide 13 successfully learns each sequence in $\mathcal{C}$, but **NO** learning machine, both reaching a single final conjecture on the sequences in $\mathcal{C}$ and always outputting conjectures consistent with the data on which they are based, successfully learns all of $\mathcal{C}$.

In the physical sciences it is generally believed that one should always employ rules/theories for extrapolating data which are as **parsimonious** as possible. One can ask, though, if the requirement of learning **parsimonious** rules/programs restricts learning power.

In some cases it does not (see the next theorem), but in other cases (see the last theorem on this slide), it does.

Theorem (Chen '81) *Some M successfully learns $\mathcal{C}$ from Slide 14, where the final and correct conjectures of M on sequences S from $\mathcal{C}$ are only computably far from minimal size.*

Theorem (Kinber '77) *Let $\mathcal{C}'$ be the class of all infinite numerical sequences such that all but finitely many of their data items are 0. Then $\mathcal{C}'$ is successfully learnable, but* **NO** *M successfully learns $\mathcal{C}'$,* **where** *the final and correct conjectures of M on sequences S from $\mathcal{C}'$ are only computably far from minimal size.**

*$\mathcal{C}$ from Slide 14 & $\mathcal{C}'$ from this slide witness the (first) theorem on Slide 11: each is successfully learnable, but **NO** single machine learns both.

References

[Cas86] J. Case. Learning machines. In W. Demopoulos and A. Marras, editors, *Language Learning and Concept Acquisition*. Ablex Publishing Company, 1986.

[Cas92] J. Case. Turing machine. In Stuart Shapiro, editor, *Encyclopedia of Artificial Intelligence*. John Wiley and Sons, New York, NY, second edition, 1992.

[Hil56] F. Hildebrand. *Introduction to Numerical Analysis*. McGraw-Hill: New York, 1956.

[KV94] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, 1994.

[OSW86] D. Osherson, M. Stob, and S. Weinstein. *Systems that Learn, An Introduction to Learning Theory for Cognitive and Computer Scientists*. MIT Press, Cambridge, Mass., 1986.