Great Ideas in Computing

University of Toronto CSC196
Fall 2023

Week 4: October 2 - October 6 (2023)

Week 4 slides

Announcements:

- Tutorial this Friday, October 6. Monday, October 9 is Thanksgiving so no class.
- My office hours are Wednesdays 11:30-12:30, Fridays 1:30-2:30 in SF2303B
- Assignment 1 is due Friday, October 6 at 9AM. Late submissions will be significantly penalized ; 10% for up to 24 hours late and an additional 20% if up to 49 hours late. I suggest submitting something well before the due date. You can ressubmit as often as you like but you cannot submit beyone the 48 hours after the djue date.

This weeks agenda

- Followup on presentation by Colin Raffel. I will provide my notes and on quercus I have posted links to three articles on large language models.
- New topic: What is computable? The Church-Turing thesis.

My brief notes from Colin Raffel's presentation

Of course, I am paraphrasing what was said and hopefully capturing the presentation.
Profdessor Raffel started by saying there are 5 ingrediants that have led to the great success of large language models.

The first ingrediemt

There is a technical sense in which general purpose language models are equivalent to predicting the next token (e.g., the next word). "Generative ML is all you need". A few examples were given:

- The cat ...? a number of possible next words are possible
- The French word for cat is ...? chat
- $87192 \times 153219 = ...?$ This is highly unlikely to bne predicted by using previous internet documents.

n-grams refers to predictions based on word frequencies.

The second ingredient

Deep learning

- The computational model is basically a combination of simple function; e.g., a threshold of a linear function. (We discussed this last Friday in our preparation for Raffel's presentation.)
- There is a loss function measuring for example how well we are predicitng the next word.
- It is possible to adaptively adjust the step size. If the step size is too small we may not be making sufficient progress fast enough (and we will possibly freeze) ; if the step size is too large we risk missing when to change directions (and possibly fall of a cliff). (*Including my additions to the analogy.*)

Ingredients 3, 4, and 5

The transformer model This is a particular architecture using lots of parallelism for example make predictions based on more examples. See the first link I provided on quercus.

Scale Using more data , more compute power, more parallelism . 10X Scaling law: For X factor more, performance improves by factor of 10X

Learning from feedback and demonstrations
Just "learning" from existing web textural material doesn't always give us what we want. Use all of the above to pre-train and then
(1) Train and demonstrations by trusted source.
(1) Train on feedback from users.

Ingredients 3, 4, and 5

The transformer model This is a particular architecture using lots of parallelism for example make predictions based on more examples. See the first link I provided on quercus.

Scale Using more data , more compute power, more parallelism . 10X Scaling law: For X factor more, performance improves by factor of 10X

Learning from feedback and demonstrations
Just "learning" from existing web textural material doesn't always give us what we want. Use all of the above to pre-train and then
(1) Train and demonstrations by trusted source.
(1) Train on feedback from users.

What is your feedback on the topic!

New topic: What is and what isn't computable

When we see the rather spectacular ways in which computer algorithms can perform, it is natural to ask whether or not there is anything that eventually we cannot do by computers.

Watching this evolution of computation and communication over say the last 80 years (since the earliest general purpose computers) and, in particular some of the most recent applications of machine learning, one can be forgiven for perhaps believing that there are no ultimate limitations.

But if we are going to ask about the limitations of computation in a precise way, well then we will need a precise mathematical framework.

This will lead is to the seminal 1930s work of Alan Turing (and independently Alonzo Church). To appreciate the seminal (and I would even say surprising) nature of this work, we consider Hilbert's 10th problem.

Computer Science as a mathematical science

David Hilbert was one of the great mathematicians of the late 19th and early 20th centuries. He asked the following question in **1900** known as Hilbert's 10^{th} problem:

"Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers"

Here is a more familiar way to ask this question:

Given a polynomial $P(x_1, \ldots, x_n)$ with integer coefficients in many variables, decide if P has an integer root. That is, do there exist integers $i_1, \ldots, i_n$ such that $P(i_1, \ldots, i_n) = 0$?

As an example, $P(x) = x - 2$ clearly has an integer root whereas $P(x) = x^2 - 2$ does not have an integer (or rational) root.

What is computable? What is decidable?

Hilbert's question was essentially to ask if there is an algorithm that could decide whether or not a given multivariate polynomial has an integer root. Hilbert didn't mention the words "algorithm" or "computer" but he did articulate the need to solve the problem in a finite number of "steps".

Hilbert believed there was such a decision procedure but did not formalize what it meant to say that a problem solution is *computable*.

Terminology: If the problem is a decision problem (i.e., where the solution is to output YES or NO) then we usually say decidable rather than computable.

Following a series of intermediate results over 21 years, in 1970 Matiyasevich gave the first proof that Hilbert's 10th problem was undecidable (in a precise sense we will next discuss).

Note: The problem is decidable for polynomials in one variable.

A precise definition for the meaning of "decidable"

We have studied the von Neumann model as a model of computation but we never gave a precise definition but more or less relied on our prior knowledge of how we think computers work. And we didn't give a definition for what is an *algorithm*.

Computers are continually getting faster and have larger memories so must our concept of what is computable also be constantly changing? Could Hilbert's problem become decidable tomorrow?

We also briefly touched upon the complexity of operations with respect to the data structures for the dictionary data type. Must the complexity of operations and the complexity of algorithms also change constantly?

This raises a fundamental question: Is there an ultimate precise model of computation with respect to which we would then have a precise meaning of a computable function? Or must we continually be changing our understanding of what is and what is not computable?

A precise definition for the meaning of computable (decidable) continued

High level models such as the von Neumann model provide a good intuition for what we have in mind when we say a function f is decidable. **But we really need a precise mathematical model if we want to prove mathematical results.**

Independently in 1936, Alonzo Church and Alan Turing published formal definitions for what it meant to be computable. These papers were very influential for the von Neumann model which comes about 10 years later.

Church's definition was based on a formalism in logic called the lambda calculus where one starts with some basic functions and then specifies ways to compose new functions from existing functions.

Alan Turing proposed a precise model of computation which we will only briefly describe. Turing also went on to show that these two very different models are provably equivalent in the sense that they result in the same set of computable functions.

But are there other models?

For a number of years other models were considered and all turn out to be equivalent (and sometimes weaker) than the Church-Turing models.

This led to the following Church-Turing hypothesis. Every plausible model of computation is equivalent to (or weaker than) Turing's very basic computational model. This is not to say that Turing machines are as easy to program or will lead to the same complexity analysis. **But the meaning of computable does not change.**

In particular, what about quantum computing?

But are there other models?

For a number of years other models were considered and all turn out to be equivalent (and sometimes weaker) than the Church-Turing models.

This led to the following Church-Turing hypothesis. Every plausible model of computation is equivalent to (or weaker than) Turing's very basic computational model. This is not to say that Turing machines are as easy to program or will lead to the same complexity analysis. **But the meaning of computable does not change.**

In particular, what about quantum computing? It could very well be that quantum computing will substantially change our sense of what is "efficiently computable" but it does not enlarge the meaning of "computable".

Note: The Church Turing hypothesis in NOT a theorem. It is an almost universally believed statement about the nature of digital computing. Could someday we come to believe that there are more inclusive models?

But are there other models?

For a number of years other models were considered and all turn out to be equivalent (and sometimes weaker) than the Church-Turing models.

This led to the following Church-Turing hypothesis. Every plausible model of computation is equivalent to (or weaker than) Turing's very basic computational model. This is not to say that Turing machines are as easy to program or will lead to the same complexity analysis. **But the meaning of computable does not change.**

In particular, what about quantum computing? It could very well be that quantum computing will substantially change our sense of what is "efficiently computable" but it does not enlarge the meaning of "computable".

Note: The Church Turing hypothesis in NOT a theorem. It is an almost universally believed statement about the nature of digital computing. Could someday we come to believe that there are more inclusive models? Yes but so far our experience leads us to believe that the hypotheis will continue to be (almost) universally accepted.

A pictorial representation of a Turing machine

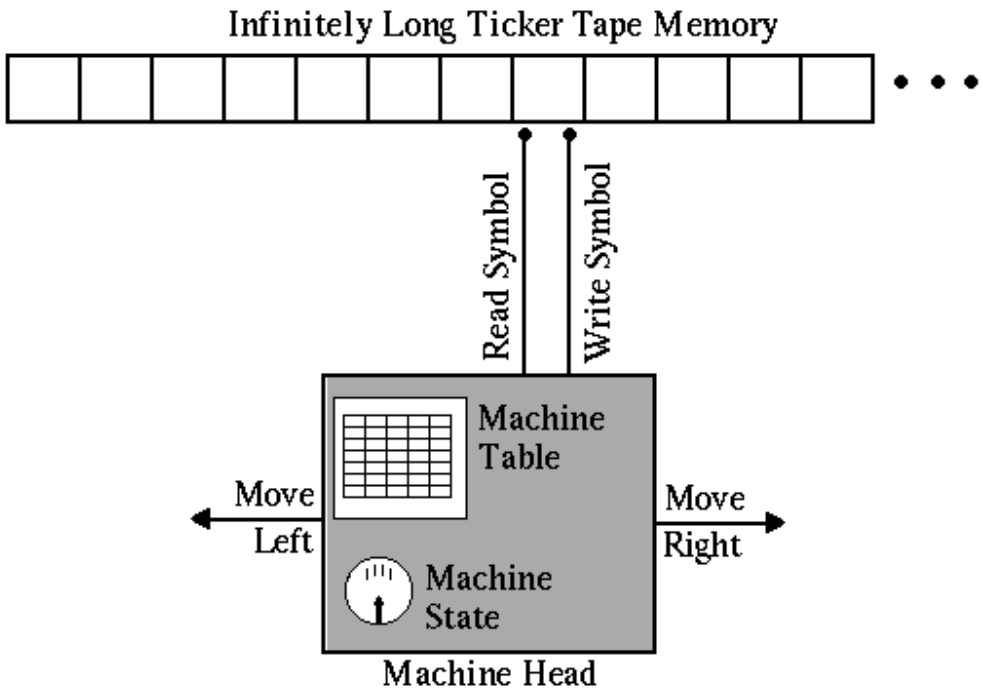

Figure: Figure taken from Michael Dawson "Understanding Cognitive Science"