# Great Ideas in Computing

University of Toronto CSC196
Fall 2022

Week 5: October 10 - October 14 (2022)

## Week 5 slides

Announcements:

- This week we had Thanksgiving on Monday and our guest Professor David Lindell presenting on Wednesday.
- Next week, we have another guest presenter, Professor Fan Long. This had to be rescheduled from November. We will have a class on Monday and Friday, Oct 21 is the first quiz. I will have more information on the quiz on Monday.
- Given all the guest presenters it has become harder for me to create an assignment using our class discussions and slides. If there are no serious objections, I am going to move the due date for Assignment 2 to now be due on Tuesday, November 1.

Todays agenda

- A very brief (mainly without slides) discussion about neural nets.
- Follow up on our two guest presentations
- Returning to computability theory: reductions and transformations, undecidable problems, diagonalization, encoding a Turing machine computation.

## Neural nets

David Lindell showed a depiction of two neural nets (slides 89 and 91) using different activation functions. Every one of his circles (which we can call gates or nodes or neurons), except the input nodes, are of the form $\sigma(c_1 y_1 + c_2 y_2 + \ldots + c_k y_k) + b)$ where sigma is the activation function, the $y_i$ are the inputs to the node, and $b$ is a bias. That is, the node is a composition of a non-linear activation function with linear affine function.

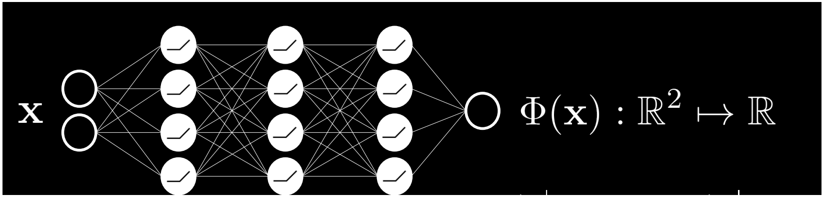


**Figure:** From Lindell's slide 89: a neural net using a linear threshold activation function

## Other activation functions

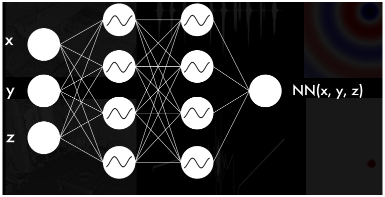


**Figure:** From Lindell's slide 91: a neural net using a sinusoldial activation function

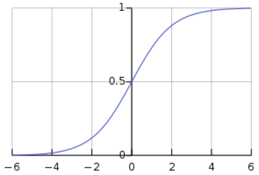
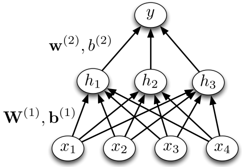


**Figure:** The sigmoid activation function

## One more activation function and a compact network representation

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

**Figure:** The $\{0,1\}$ hard threshold activation function



$y$
$\mathbf{w}^{(2)}, b^{(2)}$
$h_1$
$h_2$
$h_3$
$\mathbf{W}^{(1)}, \mathbf{b}^{(1)}$
$x_1$
$x_2$
$x_3$
$x_4$

**Figure:** Matrix representation of a neural net

## What has made neural nets so successful and what can't neural nets do?

There have been a number of major developments that has led to the rapid and great success of ML and in particular deep neural nets.

- Advances in hardware especially special purpose hardware units first developed with respect to graphics and computer games.
- An abudance of data for supervised training in many applications.
- Methods for training deep neural nets (i.e. back propagation).

What can't neural nets or other ML algorithms do well? The success of ML and neural nets in particular should make anyone hesitant to make any comments as anything I say could be quickly proved wrong.

So what do you think is well beyond today's ML and what are any long term barriers?

## What has made neural nets so successful and what can't neural nets do?

There have been a number of major developments that has led to the rapid and great success of ML and in particular deep neural nets.

- Advances in hardware especially special purpose hardware units first developed with respect to graphics and computer games.
- An abudance of data for supervised training in many applications.
- Methods for training deep neural nets (i.e. back propagation).

What can't neural nets or other ML algorithms do well? The success of ML and neural nets in particular should make anyone hesitant to make any comments as anything I say could be quickly proved wrong.

So what do you think is well beyond today's ML and what are any long term barriers?

What I can say is that there is a lot of activity in "self-learning networks; or at least how to train a neural network with little training data.

How do we interpret the meaning of the nodes at a given level or more generally how to interpret what the neural net is really doing.

## Returning to cmputability theory and undecidable problems

After the remarkable discussions by Rahul Krishnan and David Lindell, it seems like almost a paradox that there are things that are undecidable.

We will pick up where we left off on Monday, Oct 3. We will first show a reduction between two languages in order to illustrate the idea of how we use reductions to establish undecidability of a language given a language already known the be undecidable.

Recall that we use $< M >$ to denote a string repesentation of a TM $M$. Consider the following two languages introduced in the Oct 3 class.

$\mathcal{L}_2 = \{< M > | M$ is a TM that halts on all inputs $w\}$ and
$\mathcal{L}_3 = \{< M > | M$ is a TM that halts on the input string $w = 010\}$

We will show that $\mathcal{L}_3$ reduces to $\mathcal{L}_2$

## Transformations: A very special type of reduction

We will use a very simple type of reduction which I will call a *transformation* and which we can denote by $\leq_{\tau}$. We say that $A \leq_{\tau} B$ if there is a computable function $f$ such that $w \in A$ iff $f(w) \in B$.

It should be easy to see that $A \leq_{\tau} B$ is a special case of $\leq_{T}$. Is this obvious?

When we do some complexity theory, we will further refine this concept by requiring that the transformation function $f$ is computable in polynomial time (by a Turing machine).

## Showing the desired transformtion $\mathcal{L}_3 \leq_\tau \mathcal{L}_2$.

Here is a description of the transformation of $< M >$ to $f(< M >)$ such that $< M_3 > \in \mathcal{L}_3$ iff $f(< M_3 >) \in \mathcal{L}_2$.

Given an encoding of a Turing machine TM $M_3$, we are going to construct a TM $M_2$. That is, $< M_2 >= f(< M_3 >)$. $M_2$ operates on an input string $w$ as follows:
If $w \neq 010$, $M_2$ halts. If $w = 010$, then $M_2$ simulates $M_3$ on input $w$.

Claim: $M_2$ halts on the input string $w = 010$ iff $M_3$ halts on all inputs $w$.
That is, $< M_3 > \in \mathcal{L}_3$ iff $f(< M_3 >) =< M_2 > \in \mathcal{L}_2$

Given that the halting problem is undecidable, many other problems can be proved to be undecidable using reductions. Most of these problems do not mention Turing machines but undecidability comes from the problem "being able to encode the computation of a Turing machine $M$".

## Expanding on the previous slide

A halting computation is a composition of Turing machine *configurations* $C_1, C_2, \ldots, C_t$ such that $C_{i+1}$ is the configuration of the Turing machine that follows from executing one step of the Turing machine when it is in configuration $C_i$ and $C_t$ is in a halting state.

In the transformation we described, we used the fact that a Turing machine can simulate the computation of another Turing machine. This is what Turing called a *universal Turing machine*i (UTM). In modern terms, a UTM is an interpreter.

A universal Turing machine (UTM) $\mathcal{U}$ is a T.M. such that

$$\mathcal{U}(<\mathcal{M}>, w) = \mathcal{M}(w)$$

.

That is, $\mathcal{U}$ simulate exactly what $\mathcal{M}$ does on input $w$. Turing showed how to design a UTM.

## The Entscheidungsproblem

In his seminal paper "On Computable Numbers With an Application to the Entscheidungsproblem (i.e. decision problem), Turing uses his model and the undecidability of the halting problem, to prove the undecidabiliy of the "Entscheidungsproblem" posed by Hilbert in 1928. (Church provided an independent proof within his formalism.)

Sometimes this is informally stated as "can mathematics be decided ?"

The Entscheidungsproblem question refers to the decidability of predicate logic which Church and Turing independently resolved in 1936-1937. It would take a little while to formally define this "Entscheidungsproblem" but here is an example of the kind of question that one wants to answer:
Given a formula such as $\forall x \exists y : y < x$
can we determine if such a formula is always true no matter what what ordered domain $x, y$ and $<$ refer to?
For example, $x < y$ and $y < z$ implies $x \neq z$ is always true.
But $x < y$ implies $\exists z : x < z < y$ is not true of all ordered domains (e.g., consider the integers) but is true of the rationals.

## Repeating the pictorial representation of a Turing machine

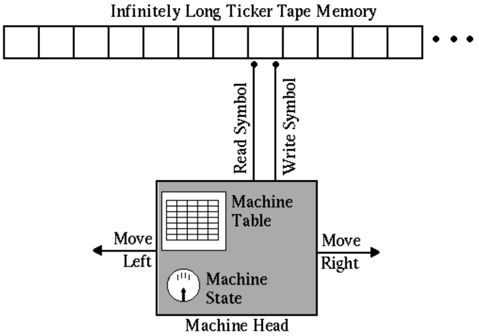


**Figure:** Figure taken from Michael Dawson, “Understanding Cognitive Science”

## Formalization of a Turing machine

- Formally, a Turing machine algorithm is described by the following function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
  $Q$ is a *finite* set of *states*. $\Gamma$ is a finite set of symbols (e.g., $\Gamma = \{\#, 0, 1, a, b, \ldots\}$ and perhaps $\Sigma = \{0, 1\}$)
- Note: Each $\delta$ function is the definition of a single Turing machine; that is, each $\delta$ function is the statement of an algorithm.
- We can assume there is a halting state $q_{halt}$ such that the machine halts if it enters state $q_{halt}$. There is also an initial state $q_0$.
- We view a Turing machine $P$ as computing a function $f_P : \Sigma^* \to \Sigma^*$ where $\Sigma \subseteq \Gamma$ where $y = f(x)$ is the string that remains if (and when) the machine halts. There can be other conventions as to interpreting the resulting output $y$.
- Note that the model is precisely defined as is the concept of a computation step. A *configuration* of a TM is specified by the contents of the tape, the state, and the position of the tape head. A computation of a TM is a sequence of configurations, starting with an initial configuration.
- For decision problems, we can have YES and NO halting states.

## A more general Turing machine model

The Turing machine model has been extended to allow separate read (for the input) and write (for the output when computing a function) tapes and any finite number of work tapes. Here is a figure of a multi-tape TM (but without separate input and output tapes).

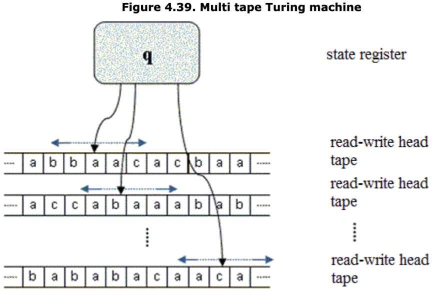


**Figure:** Figure from the Bela Gyires Informatics Curriculum Repository

## Diagonalization

You may have learned in high school why using the diagonalization method it can be proved that the set of real numbers say in $(0, 1)$ is *uncountable* while the set of rationals in $(0, 1)$ is *countable*.

The idea is that since the rationals are countable we can list them and lets say that $r_i = \text{ the } i^{th}$ rational number in $(0, 1)$ (in some agreed uopon listing of these rationals). Suppose as a binary fraction $r_i = .r_i(1)r_i(2)r_I(3)\ldots r_i(m_i)$ for some $m_i$ and $r_i(j) = 0$ for all $j > m_i$.

We can create a non-rational number $x = .s_1\ s_2\ s_3 \ldots$ where $s_i = 1 - r_i(i)$.

## Diagonalization and the halting problem

We will just sketch why the halting problem is undecidable.

Using diagonalization, we can show that the following halting problem is undecidable. Namely, the set of Turing machines is a countable set and let $\mathcal{M}_i$ denote the $i^{th}$ TM. Consider the following function $f(i) = \mathit{YES}$ if $\mathcal{M}_i(i) = \text{ NO or } M_i(i)$ does not halt, and $f(i) =$ NO otherwise.

If the halting problem were decideable, then using a UTM and the claimed decidability of the halting problem, $f$ would be a computable function but that would be a contradiction since $f$ is defined to be different than every TM $\mathcal{M}_j$.

What Turing showed is that we can encode whether or not a Turing machine $M$ accepts input $w$ (i.e. the halting problem) by a statement in predicate logic.