Turing Machines and RNNs

Andy Yang

LAST TIME: DFAs and Regular Languages

Things that DFAs can do:

- PARITY - e.g. keeping track if the door is opened or closed
- Dyck-1 of depth 2 - e.g. ensuring a room of occupancy 2 is empty at the start and end of the day
- $(aa)^*$ - e.g. checking you can arrange your plants in two rows
- $\Sigma^* a \Sigma^* \$ \Sigma^* a \Sigma^*$ - e.g. listening to an entire speech and then attempting to repeat it from memory but only getting one word right

THIS TIME: Beyond DFAs and Regular Languages

Things that DFAS cannot do. **How would you perform these tasks?**:

- $\{w \mid w \in \Sigma^*, \#a \geq \#b\}$ - e.g. keeping track of football game score to see who won
- Unbounded Dyck-1 - e.g. ensuring a room (of infinite occupancy) is empty at the start and end of the day
- a^{n^2} - e.g. checking if you can arrange your plants in a square grid
- $\{w\$w \mid w \in \Sigma^*\}$ - e.g. listening to an entire speech and then repeating the entire thing from memory

Alan Turing

Important paper: On computable numbers, with an application to the Entscheidungsproblem (Turing 1936)

Figure 1: Alan Turing pondering the Entscheidungsproblem

Computers

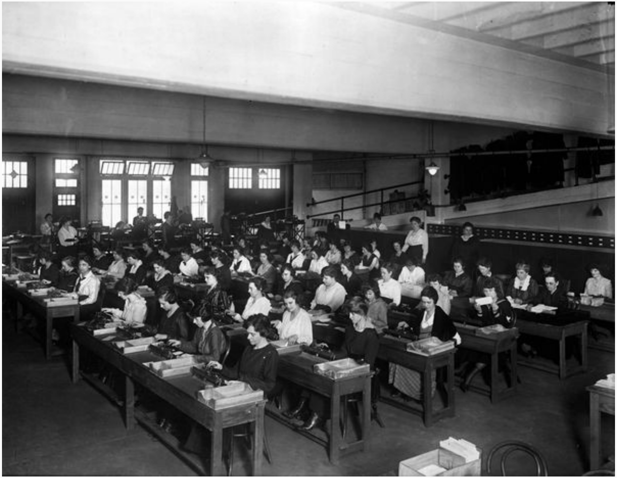


Figure 2: a room full of computers

Turing Machines

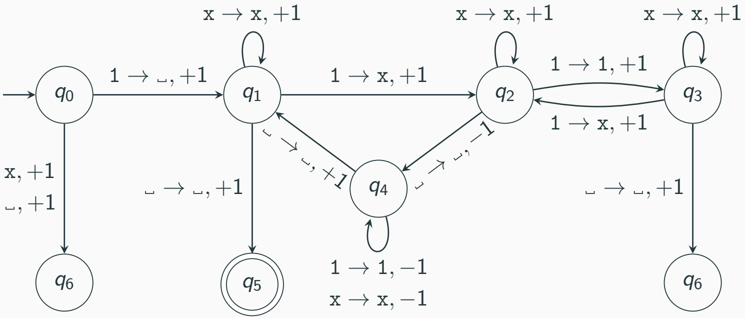
Definition
A Turing machine is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$, where

- Q is a finite set of states
- Σ is a finite input alphabet, where $\textvisiblespace \notin \Sigma$
- Γ is a finite tape alphabet, where $\Sigma \cup \{\textvisiblespace\} \subseteq \Gamma$
- $\delta \colon Q \times \Gamma \to Q \times \Gamma \times \{-1, +1\}$ is the transition function.

The tape has a left end and extends infinitely to the right. On input $\mathbf{w} \in \Sigma^*$, the tape is initialized to $\mathbf{w}\textvisiblespace\textvisiblespace\cdots$. If the current state is q, the current tape symbol is a, and $\delta(q, a) = (r, b, m)$, then the machine enters state r, writes a b, and moves left if $m = -1$, right if $m = +1$. If the machine enters state q_{accept}, it halts and accepts $\mathbf{w}$; if it enters state q_{reject}, it halts and rejects $\mathbf{w}$.

Turing Machines

Here's an example Turing machine [2], with $q_{\text{start}} = q_0$, $q_{\text{accept}} = q_5$ (marked with a double circle), $q_{\text{reject}} = q_6$. It decides the language $\{1^{2^m} \mid m \geq 0\}$.



The reject state q_6 appears twice to reduce clutter.

RNNs

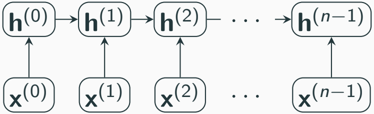


Figure 3: A simple RNN yearning to become a Turing machine

Rational-Weight RNNs Simulate Turing Machines

Theorem (1)

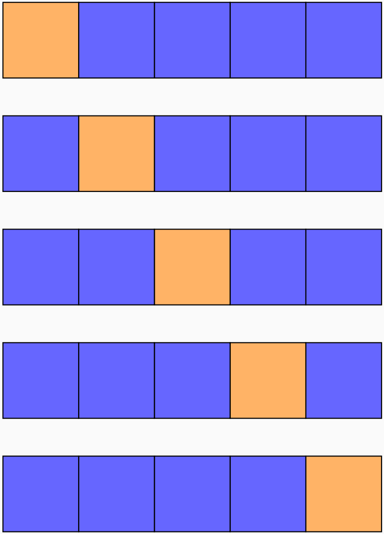
For any Turing machine M with input alphabet Σ, there is a network $f = out \circ rec$, where rec is a simple RNN with rational weights and ReLU *activation functions, and out is a linear layer, that is equivalent to M in the following sense: for any string $\mathbf{w} \in \Sigma^*$,*

- *If M halts and accepts on input $\mathbf{w}$, then there is a T such that for all $t \in [T]$, $f(\mathbf{w} \cdot \mathrm{BOS} \cdot \mathrm{NUL}^t) = \mathbf{e}_{\mathrm{NUL}}$ and $f(\mathbf{w} \cdot \mathrm{BOS} \cdot \mathrm{NUL}^T) = \mathbf{e}_{\mathrm{ACC}}$.*
- *If M halts and rejects on input $\mathbf{w}$, then there is a T such that for all $t \in [T]$, $f(\mathbf{w} \cdot \mathrm{BOS} \cdot \mathrm{NUL}^t) = \mathbf{e}_{\mathrm{NUL}}$ and $f(\mathbf{w} \cdot \mathrm{BOS} \cdot \mathrm{NUL}^T) = \mathbf{e}_{\mathrm{REJ}}$.*
- *If M does not halt on input $\mathbf{w}$, then for all $t \geq 0$, $f(\mathbf{w} \cdot \mathrm{BOS} \cdot \mathrm{NUL}^t) = \mathbf{e}_{\mathrm{NUL}}$.*

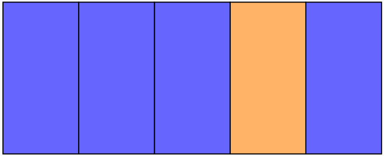
Integers vs Rationals

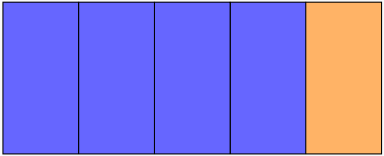
Last time, because $\mathrm{SLU}\colon \mathbb{R} \to [0,1]$, when rounded to integer weights it becomes $\mathbb{Z} \to \{0,1\}$. So there are finitely many states $\mathbf{h}^{(i)}$. This is not the case if weights are rational.

Stack Encoding

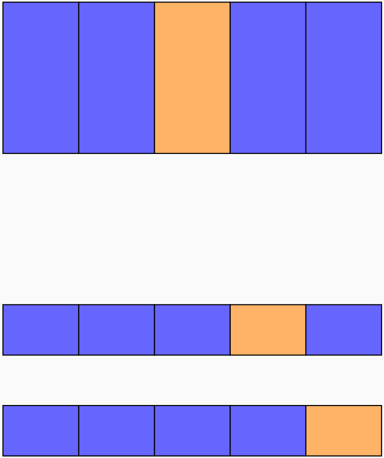


Stack Encoding

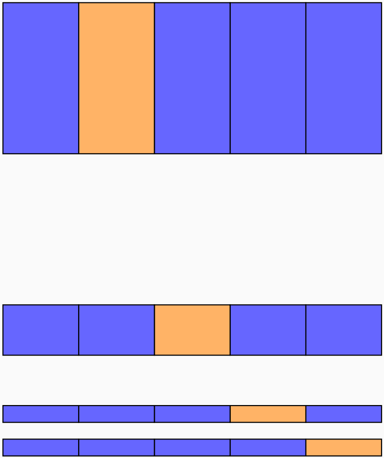




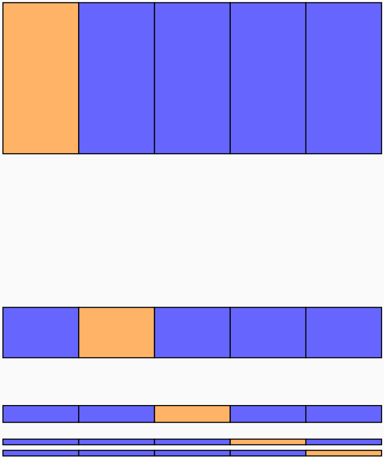
Stack Encoding



Stack Encoding



Stack Encoding



Stack Encoding

Let $\Gamma = \{a_1, a_2, \ldots, a_{|\Gamma|}\}$ be the alphabet of stack symbols. We encode a stack as a vector of $|\Gamma|$ rational numbers using the following mapping:

$$\begin{aligned} \text{stack} \colon \Gamma^* &\to \mathbb{Q}^{|\Gamma|} \\ \text{stack}(\epsilon) &= \mathbf{0} && (1) \\ \text{stack}(a_j \cdot \mathbf{z}) &= \tfrac{2}{3}\mathbf{e}_j + \tfrac{1}{3}\text{stack}(\mathbf{z}). && (2) \end{aligned}$$

For each $a \in \Gamma$, this encoding puts a "margin" between stacks without an a on top and stacks with an a on top, so that a SLU network can distinguish them:

Then the basic stack operations can be implemented as follows:

$$\begin{aligned} \text{push}(\mathbf{z}, a_j) &= \tfrac{2}{3}\mathbf{e}_j + \tfrac{1}{3}\mathbf{z} && (3) \\ \text{top}(\mathbf{z}) &= \text{SLU}(3\mathbf{z} - 1) && (4) \\ \text{pop}(\mathbf{z}) &= 3\mathbf{z} - 2\,\text{top}(\mathbf{z}). && (5) \end{aligned}$$

References

[1] Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50 (1):132–150, 1995. doi: https://doi.org/10.1006/jcss.1995.1013.

[2] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 3rd edition, 2013.