Turing Machines and RNNs

Andy Yang

Rational-Weight RNNs

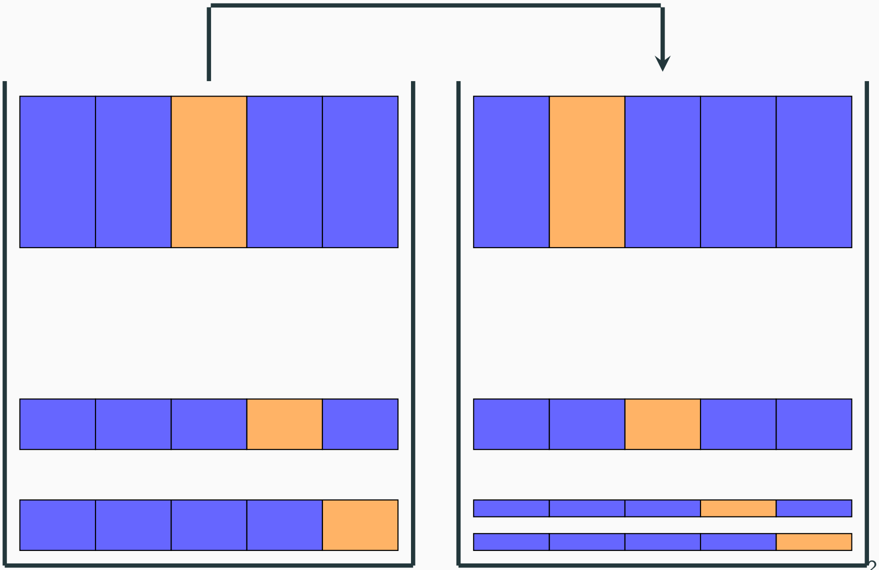
Rational-Weight RNNs Simulate Turing Machines

Theorem (3)

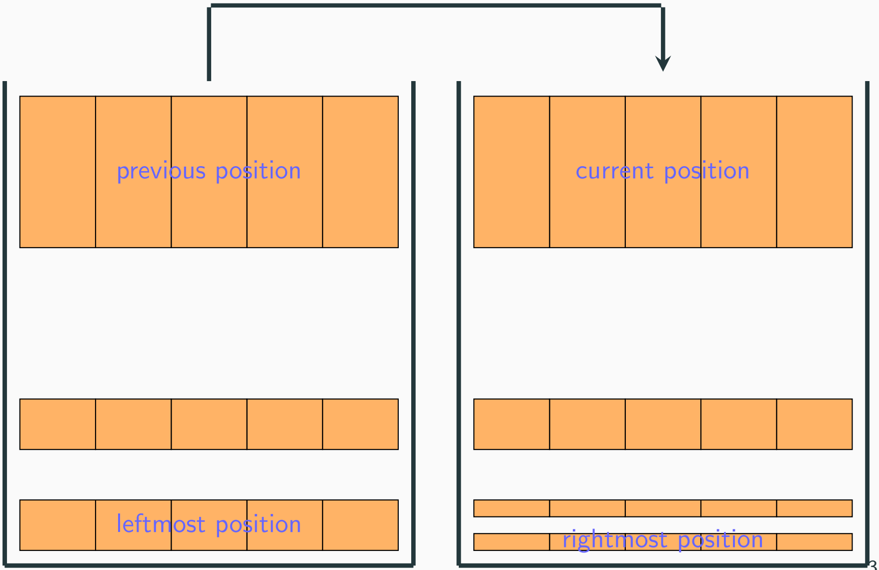
For any Turing machine M with input alphabet Σ, there is a network $f = out \circ rec$, where rec is a simple RNN with rational weights and ReLU *activation functions, and out is a linear layer, that is equivalent to M in the following sense: for any string $\mathbf{w} \in \Sigma^*$,*

- *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 $t \geq 0$, $f(\mathbf{w} \cdot \mathrm{BOS} \cdot \mathrm{NUL}^t) = \mathbf{e}_{\mathrm{NUL}}$.*

Stacks



Stacks


previous position
leftmost position
current position
rightmost position

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:

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

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:

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

State

The hidden vectors of the RNN are

$$\mathbf{h}^{(i)} = \begin{bmatrix} \mathbf{q}^{(i)} \\ \mathbf{l}^{(i)} \\ \mathbf{c}^{(i)} \\ \mathbf{r}^{(i)} \end{bmatrix} \tag{6}$$

where $\mathbf{q}^{(i)}$ is the one-hot vector of the current state, $\mathbf{l}^{(i)}$ and $\mathbf{r}^{(i)}$ are the left and right stacks, respectively, and $\mathbf{c}$ is the one-hot vector of the current symbol. For clarity (I hope), let's write q_i in place of $\mathbf{e}_i$.

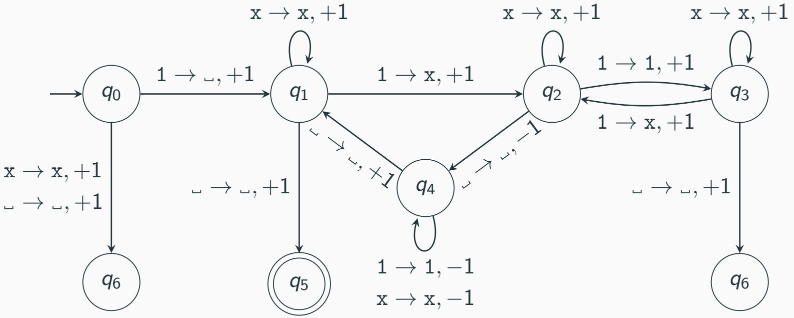
The initial vector is

$$\mathbf{h}^{(-1)} = \begin{bmatrix} q_0 \\ \text{stack}(\$) \\ \text{␣} \\ \text{stack}(\$) \end{bmatrix}. \tag{7}$$

We initialized both stacks with a $ on the bottom and a blank as the current symbol.

Turing Machines

Here's an example Turing machine [4], 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.

What do we need?

- Simulate tape
- Simulate state
- Simulate transitions

State Control

We've shown previously how to define Boolean operators using ReLUs; we also need

$$\mathrm{if}(b, t) = \mathrm{ReLU}(t + b - 1) \tag{8}$$

Assuming $b \in \{0, 1\}$ and $t \in [0, 1]$, this means "if $b = 1$, then t, else 0."

Simulation

The recurrent step is a big piecewise linear function. We break it up into four pieces:

$$
\begin{aligned}
\mathit{step}\left(\begin{bmatrix}\mathbf{q}\\\mathbf{l}\\\mathbf{c}\\\mathbf{r}\end{bmatrix}, \mathbf{x}\right) &= \mathit{load}\left(\begin{bmatrix}\mathbf{q}\\\mathbf{l}\\\mathbf{c}\\\mathbf{r}\end{bmatrix}, \mathbf{x}\right) + \mathit{rewind}\left(\begin{bmatrix}\mathbf{q}\\\mathbf{l}\\\mathbf{c}\\\mathbf{r}\end{bmatrix}\right) && (9)\\
&\quad + \mathit{left}\left(\begin{bmatrix}\mathbf{q}\\\mathbf{l}\\\mathbf{c}\\\mathbf{r}\end{bmatrix}\right) + \mathit{right}\left(\begin{bmatrix}\mathbf{q}\\\mathbf{l}\\\mathbf{c}\\\mathbf{r}\end{bmatrix}\right). && (10)
\end{aligned}
$$

Key: At any time, only one piece is ever nonzero

Loading

The first term initially loads the input string onto the tape, from left to right:

$$\mathit{load}\left(\begin{bmatrix}\mathbf{q}\\\mathbf{l}\\\mathbf{c}\\\mathbf{r}\end{bmatrix},\mathbf{x}\right)=\text{if}\left(\mathbf{q}=q_0\wedge\mathbf{x}\neq\mathrm{EOS},\begin{bmatrix}q_0\\\text{push}(\mathbf{l},\mathbf{x})\\\text{␣}\\\mathbf{r}\end{bmatrix}\right)+\text{if}\left(\mathbf{q}=q_0\wedge\mathbf{x}=\mathrm{EOS},\begin{bmatrix}q_1\\\mathbf{l}\\\mathbf{c}\\\mathbf{r}\end{bmatrix}\right).\qquad(11)$$

Rewinding

The second term just rewinds the head back to the left end of the tape:

$$\textit{rewind}\left(\begin{bmatrix}\mathbf{q}\\ \mathbf{l}\\ \mathbf{c}\\ \mathbf{r}\end{bmatrix}\right) = \text{if}\left(\mathbf{q} = q_1 \wedge \text{top}(\mathbf{l}) \neq \$, \begin{bmatrix} q_1 \\ \text{pop}(\mathbf{l}) \\ \text{top}(\mathbf{l}) \\ \text{push}(\mathbf{r}, \mathbf{c})\end{bmatrix}\right) + \text{if}\left(\mathbf{q} = q_1 \wedge \text{top}(\mathbf{l}) = \$, \begin{bmatrix} q_2 \\ \mathbf{l} \\ \mathbf{c} \\ \mathbf{r}\end{bmatrix}\right). \tag{12}$$

Left Transitions

The third term handles all the left-moving transitions:

$$
\begin{aligned}
\mathit{left}\left(\begin{bmatrix}\mathbf{q}\\ \mathbf{l}\\ \mathbf{c}\\ \mathbf{r}\end{bmatrix}\right) = &\sum_{\substack{q,q'\in Q\\ a,a'\in\Gamma\\ \delta(q,a)=(q',a',-1)}} \text{if}\left(\mathbf{q}=q\wedge\mathbf{c}=a\wedge\mathrm{top}(\mathbf{l})\neq\$,\ \begin{bmatrix}q'\\ \mathrm{pop}(\mathbf{l})\\ \mathrm{top}(\mathbf{l})\\ \mathrm{push}(\mathbf{r},a')\end{bmatrix}\right)\\
&+\sum_{\substack{q,q'\in Q\\ a,a'\in\Gamma\\ \delta(q,a)=(q',a',-1)}} \text{if}\left(\mathbf{q}=q\wedge\mathbf{c}=a\wedge\mathrm{top}(\mathbf{l})=\$,\ \begin{bmatrix}q'\\ \mathbf{l}\\ a'\\ \mathbf{r}\end{bmatrix}\right).
\end{aligned}
\tag{13}
$$

Right Transitions

The last term handles all the right-moving transitions:

$$\begin{aligned} \mathit{right}\left(\begin{bmatrix}\mathbf{q}\\ \mathbf{l}\\ \mathbf{c}\\ \mathbf{r}\end{bmatrix}\right) &= \sum_{\substack{q,q'\in Q\\ a,a'\in\Gamma\\ \delta(q,a)=(q',a',+1)}} \text{if}\left(\mathbf{q}=q\wedge\mathbf{c}=a\wedge\text{top}(\mathbf{r})\neq\$,\begin{bmatrix}q'\\ \text{push}(\mathbf{l},a')\\ \text{top}(\mathbf{r})\\ \text{pop}(\mathbf{r})\end{bmatrix}\right)\\ &+ \sum_{\substack{q,q'\in Q\\ a,a'\in\Gamma\\ \delta(q,a)=(q',a',+1)}} \text{if}\left(\mathbf{q}=q\wedge\mathbf{c}=a\wedge\text{top}(\mathbf{r})=\$,\begin{bmatrix}q'\\ \text{push}(\mathbf{l},a')\\ \llcorner\!\lrcorner\\ \mathbf{r}\end{bmatrix}\right). \end{aligned} \tag{14}$$

Output

Finally, we define an output layer such that

$$\mathit{out}\left(\begin{bmatrix}\mathbf{q}\\\mathbf{l}\\\mathbf{c}\\\mathbf{r}\end{bmatrix}\right)=\begin{cases}\mathbf{e}_{\mathrm{ACC}} & \text{if } \mathbf{q}=q_{\text{accept}}\\\mathbf{e}_{\mathrm{REJ}} & \text{if } \mathbf{q}=q_{\text{reject}}\\\mathbf{e}_{\mathrm{NUL}} & \text{otherwise.}\end{cases}$$

Notes

If the number of intermediate steps is bounded by $T(n)$, then the number of simulated steps of the Turing machine is also bounded by $T(n)$.

The presented proof did not result in a simple RNN (multiple ReLU layers) but the notes discuss how to squash it down to 1.

Proof idea be extended to simulating probabilistic Turing machines, with rational weights Nowak et al. [1].

Real-Weight RNNs

Theorem Sttaement

The last result we look at is purely theoretical: If we allow an RNN to have real weights, how powerful is it?

Theorem (Siegelmann and Sontag 2)
For any language L over Σ, there is a network $f = out \circ rec$, where rec is a simple RNN with real weights and ReLU *activation functions, and out is a linear layer, that decides L in the following sense: for any string $\mathbf{w} \in \Sigma^*$,*

- *If $\mathbf{w} \in L$, 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 $\mathbf{w} \notin L$, 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 $f(\mathbf{w} \cdot \mathrm{BOS} \cdot \mathrm{NUL}^T) = \mathbf{e}_{\mathrm{REJ}}$.*

There exists an RNN that always halts

Encoding a Language

We have already seen how to encode a string over Σ as a vector of $|\Sigma|$ rational numbers. Under the same encoding, we can encode an *infinite* string as a vector of real numbers. Let's think of an infinite string over Σ as a mapping $w \colon \mathbb{N}_{>0} \to \Sigma$, that is, $w(i)$ is the symbol at position i. Then

$$\text{stack}(w) = \sum_{i=1}^{\infty} \frac{2}{3^i} \mathbf{e}_{w(i)}. \tag{15}$$

Given a language L, we can enumerate the (finite) strings of L in order of increasing length, as $\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \ldots$. Then we can concatenate them into a single infinite string, $\langle L \rangle = \mathbf{w}^{(0)}\texttt{\#}\mathbf{w}^{(1)}\texttt{\#}\cdots$. For example, if $L = \{\mathbf{ww} \mid \mathbf{w} \in \{\mathtt{a}, \mathtt{b}\}^*\}$, then

$$\langle L \rangle = \texttt{\#aa\#bb\#aaaa\#abab\#baba\#bbbb\#aaaaaa\#}\cdots.$$

Proof

The construction in the proof of Theorem 1 can be modified to simulate a Turing machine with two tapes (or a tape and a stack is enough): the first tape is as before, while the second tape is initialized with $\langle L \rangle$. Then construct an RNN that simulates the Turing machine $M =$ "On input $\mathbf{w}$:

1. Compare the string on the first tape ($\mathbf{w}$) with the string on the second tape, starting from the current position up to (but not including) #.
2. If they are equal, *accept*.
3. If $\mathbf{w}$ is shorter than the other string, *reject*.
4. Otherwise, move the first head back to the left end, move the second head immediately to the right of the #, and goto 1.

References

[1] Franz Nowak, Anej Svete, Li Du, and Ryan Cotterell. On the representational capacity of recurrent neural language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

[2] Hava T. Siegelmann and Eduardo D. Sontag. Analog computation via neural networks. *Theoretical Computer Science*, 131(2):331–360, 1994. doi: 10.1016/0304-3975(94)90178-3.

[3] 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.

References ii

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