On the Turing Completeness of Recurrent Neural Networks

Francesco Ballerini

Università degli Studi di Firenze
Scuola di Ingegneria

February 27th 2020

Motivations

Why neural networks

Deep learning (deep neural networks) is currently the prevalent machine-learning approach for:

Why neural networks

Deep learning (deep neural networks) is currently the prevalent machine-learning approach for:

- Computer vision

Why neural networks

Deep learning (deep neural networks) is currently the prevalent machine-learning approach for:

- Computer vision
- Natural-language processing

Why neural networks

Deep learning (deep neural networks) is currently the prevalent machine-learning approach for:

- Computer vision
- Natural-language processing
- In general, all perceptual tasks

Why Turing completeness

From a theory-of-computation standpoint:

Why Turing completeness

From a theory-of-computation standpoint:

- It would be nice to know that neural networks are as powerful as Turing machines (TMs)

Why Turing completeness

From a theory-of-computation standpoint:

- It would be nice to know that neural networks are as powerful as Turing machines (TMs)
- So as to ensure that any effectively calculable function is computable by a neural network (Church–Turing thesis)

The result

As it turns out, given a TM, we can build a recurrent neural network (RNN) with rational weights and biases that computes the same function as the TM

The result

This result bridges two worlds:

The result

This result bridges two worlds:

- Symbolic computation (TMs):

- "Neural" computation (RNNs):

The result

This result bridges two worlds:

- Symbolic computation (TMs):
  - Finite alphabets of symbols
- "Neural" computation (RNNs):
  - Continuous values

The result

This result bridges two worlds:

- Symbolic computation (TMs):
  - Finite alphabets of symbols
  - Explicit separation of control (state) and memory (tape)
- "Neural" computation (RNNs):
  - Continuous values
  - No intrinsic separation of state vs memory

The result

This result bridges two worlds:

- Symbolic computation (TMs):
  - Finite alphabets of symbols
  - Explicit separation of control (state) and memory (tape)
  - If–then conditionals used for updates
- "Neural" computation (RNNs):
  - Continuous values
  - No intrinsic separation of state vs memory
  - No intrinsic Boolean logic

Our contribution

The original article

- The original proof is contained in an article named "On the Computational Power of Neural Nets" by Hava Siegelmann and Eduardo Sontag

The original article

- The original proof is contained in an article named "On the Computational Power of Neural Nets" by Hava Siegelmann and Eduardo Sontag
- The article dates back to 1992

Our modernization

What we did was:

Our modernization

What we did was:

- Updating the formalism and terminology according to what is widely used today

Our modernization

What we did was:

- Updating the formalism and terminology according to what is widely used today
- Adding

Our modernization

What we did was:

- Updating the formalism and terminology according to what is widely used today
- Adding
  - Figures

Our modernization

What we did was:

- Updating the formalism and terminology according to what is widely used today
- Adding
  - Figures
  - Tables

Our modernization

What we did was:

- Updating the formalism and terminology according to what is widely used today
- Adding
  - Figures
  - Tables
  - Explicit calculations

Our modernization

What we did was:

- Updating the formalism and terminology according to what is widely used today
- Adding
  - Figures
  - Tables
  - Explicit calculations

in order to leave as little as possible to the imagination of the reader

Our modernization

What we did was:

- Updating the formalism and terminology according to what is widely used today
- Adding
  - Figures
  - Tables
  - Explicit calculations

  in order to leave as little as possible to the imagination of the reader
- Adding an example in which we apply the construction showed in the proof

Our modernization

What we did was:

- Updating the formalism and terminology according to what is widely used today
- Adding
  - Figures
  - Tables
  - Explicit calculations

  in order to leave as little as possible to the imagination of the reader
- Adding an example in which we apply the construction showed in the proof
  - And providing a Python implementation of such example

Our modernization

What we did was:

- Updating the formalism and terminology according to what is widely used today
- Adding
  - Figures
  - Tables
  - Explicit calculations

  in order to leave as little as possible to the imagination of the reader
- Adding an example in which we apply the construction showed in the proof
  - And providing a Python implementation of such example
- Most importantly, giving a more detailed and (hopefully) pleasant structure to the proof itself

Our modernization

What we did was:

- Updating the formalism and terminology according to what is widely used today
- Adding
  - Figures
  - Tables
  - Explicit calculations

  in order to leave as little as possible to the imagination of the reader
- Adding an example in which we apply the construction showed in the proof
  - And providing a Python implementation of such example
- Most importantly, giving a more detailed and (hopefully) pleasant structure to the proof itself
  - Which, in our experience, was pretty frustrating to make sense of

Preliminaries

What this presentation is about

- We will not show the formal proof of the general case

What this presentation is about

- We will not show the formal proof of the general case
- Instead, we will focus on an example of how to build an RNN that simulates a TM

What this presentation is about

- We will not show the formal proof of the general case
- Instead, we will focus on an example of how to build an RNN that simulates a TM
- And, actually, instead of considering a TM, we will start from a double-stack pushdown automaton (double-stack PDA)

What this presentation is about

- We will not show the formal proof of the general case
- Instead, we will focus on an example of how to build an RNN that simulates a TM
- And, actually, instead of considering a TM, we will start from a double-stack pushdown automaton (double-stack PDA)
- This is not a problem, since double-stack PDAs can be proven to be computationally equivalent to single-tape TMs

What this presentation is about

- We will not show the formal proof of the general case
- Instead, we will focus on an example of how to build an RNN that simulates a TM
- And, actually, instead of considering a TM, we will start from a double-stack pushdown automaton (double-stack PDA)
- This is not a problem, since double-stack PDAs can be proven to be computationally equivalent to single-tape TMs
  - That is, a function is computable by a double-stack PDA if and only if it is computable by a single-tape TM

Rational stack encoding

- We want to encode a stack content as a number

Rational stack encoding

- We want to encode a stack content as a number
  - This will turn out to be useful later ...

Rational stack encoding

- We want to encode a stack content as a number
  - This will turn out to be useful later ...
- We will work with PDAs with two binary stacks

Rational stack encoding

- We want to encode a stack content as a number
  - This will turn out to be useful later ...
- We will work with PDAs with two binary stacks
  - Equivalent to binary single-tape TMs

Rational stack encoding

- We want to encode a stack content as a number
  - This will turn out to be useful later ...
- We will work with PDAs with two binary stacks
  - Equivalent to binary single-tape TMs
- We choose the following encoding, where $a_i \in \{0, 1\}$:

$$\begin{array}{|c|} \hline a_1 \\ \hline a_2 \\ \hline \vdots \\ \hline a_n \\ \hline \end{array} \longrightarrow \quad q = \sum_{i=1}^{n} \frac{2a_i + 1}{4^i} \in \mathbb{Q}$$

Rational stack encoding

- We want to encode a stack content as a number
  - This will turn out to be useful later ...
- We will work with PDAs with two binary stacks
  - Equivalent to binary single-tape TMs
- We choose the following encoding, where $a_i \in \{0, 1\}$:

$$\begin{array}{|c|}\hline a_1 \\ \hline a_2 \\ \hline \vdots \\ \hline a_n \\ \hline \end{array} \quad \longrightarrow \quad q = \sum_{i=1}^{n} \frac{2a_i + 1}{4^i} \in \mathbb{Q}$$

- $q \in [0, 1)$

Rational stack encoding

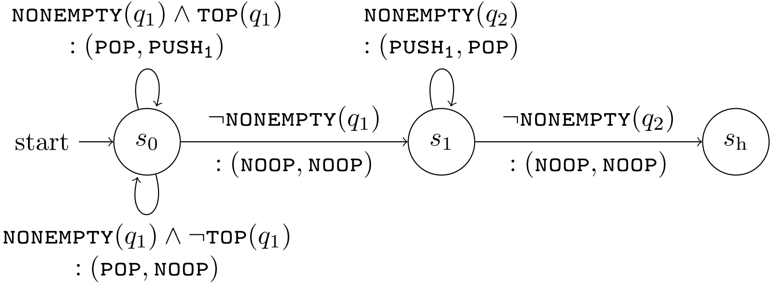
- We want to encode a stack content as a number
  - This will turn out to be useful later ...
- We will work with PDAs with two binary stacks
  - Equivalent to binary single-tape TMs
- We choose the following encoding, where $a_i \in \{0, 1\}$:

$$\begin{array}{|c|} \hline a_1 \\ \hline a_2 \\ \hline \vdots \\ \hline a_n \\ \hline \end{array} \longrightarrow \quad q = \sum_{i=1}^{n} \frac{2a_i + 1}{4^i} \in \mathbb{Q}$$

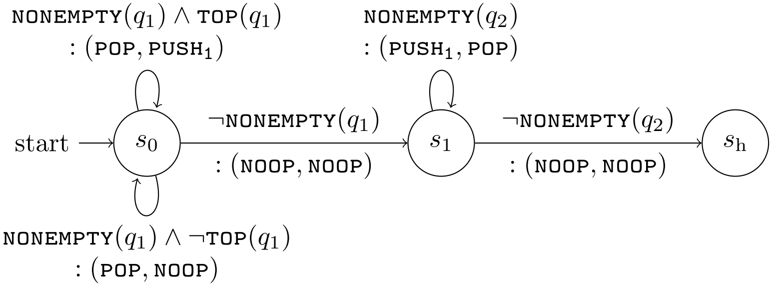
- $q \in [0, 1)$
  - So, it might be a good candidate for a neuron activation ...

PDA $\rightarrow$ RNN

Double-stack PDA computing unary addition

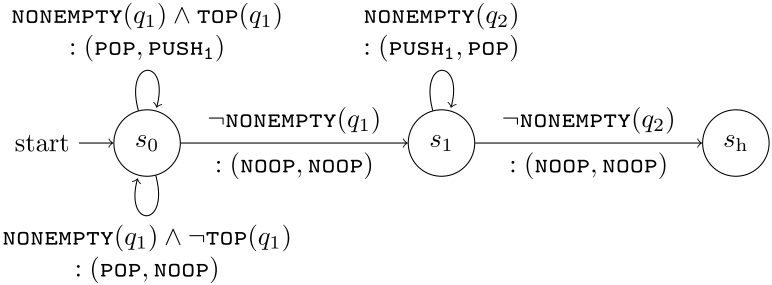


Double-stack PDA computing unary addition



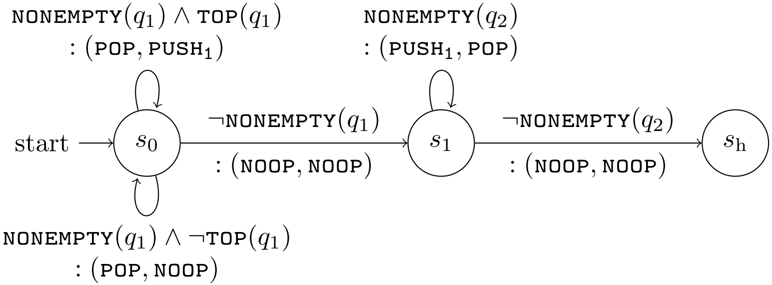
- Stack 1 starts with the two unary numbers to add separated by a zero

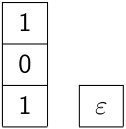
Double-stack PDA computing unary addition



- Stack 1 starts with the two unary numbers to add separated by a zero
- Stack 2 starts empty

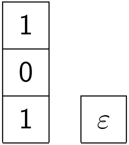
Double-stack PDA computing unary addition



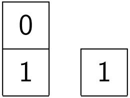
- Stack 1 starts with the two unary numbers to add separated by a zero
- Stack 2 starts empty
  - E.g. $1 + 1 \longrightarrow$ 

Double-stack PDA computing unary addition

Start



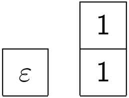
Double-stack PDA computing unary addition



Double-stack PDA computing unary addition

$\boxed{1}$ $\boxed{1}$

Double-stack PDA computing unary addition

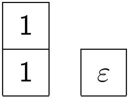


Double-stack PDA computing unary addition

$\boxed{1}$ $\boxed{1}$

Double-stack PDA computing unary addition

Halt



Defining the neural-network input layer

- At a given time t, the PDA is uniquely described by

Defining the neural-network input layer

- At a given time t, the PDA is uniquely described by
  - Its state

Defining the neural-network input layer

- At a given time t, the PDA is uniquely described by
  - Its state
  - The content of stack 1

Defining the neural-network input layer

- At a given time t, the PDA is uniquely described by
  - Its state
  - The content of stack 1
  - The content of stack 2

Defining the neural-network input layer

- At a given time t, the PDA is uniquely described by
  - Its state
  - The content of stack 1
  - The content of stack 2
- We want to be able to express the description of the PDA at time t as a neural-network layer

Defining the neural-network input layer

- At a given time t, the PDA is uniquely described by
  - Its state
  - The content of stack 1
  - The content of stack 2
- We want to be able to express the description of the PDA at time t as a neural-network layer
- To do so, consider the tuple $(x_0, x_1, x_{\mathrm{h}}, q_1, q_2) \in \mathbb{Q}^5$, where:

Defining the neural-network input layer

- At a given time t, the PDA is uniquely described by
  - Its state
  - The content of stack 1
  - The content of stack 2
- We want to be able to express the description of the PDA at time t as a neural-network layer
- To do so, consider the tuple $(x_0, x_1, x_h, q_1, q_2) \in \mathbb{Q}^5$, where:
  - $(x_0, x_1, x_h) = \begin{cases} (1, 0, 0) & \text{if the PDA is in state } s_0 \\ & \\ & \end{cases}$

Defining the neural-network input layer

- At a given time t, the PDA is uniquely described by
  - Its state
  - The content of stack 1
  - The content of stack 2
- We want to be able to express the description of the PDA at time t as a neural-network layer
- To do so, consider the tuple $(x_0, x_1, x_h, q_1, q_2) \in \mathbb{Q}^5$, where:
  - $(x_0, x_1, x_h) = \begin{cases} (1,0,0) & \text{if the PDA is in state } s_0 \\ (0,1,0) & \text{if the PDA is in state } s_1 \\ \end{cases}$

Defining the neural-network input layer

- At a given time t, the PDA is uniquely described by
  - Its state
  - The content of stack 1
  - The content of stack 2
- We want to be able to express the description of the PDA at time t as a neural-network layer
- To do so, consider the tuple $(x_0, x_1, x_h, q_1, q_2) \in \mathbb{Q}^5$, where:
  - $(x_0, x_1, x_h) = \begin{cases} (1,0,0) & \text{if the PDA is in state } s_0 \\ (0,1,0) & \text{if the PDA is in state } s_1 \\ (0,0,1) & \text{if the PDA is in state } s_h \end{cases}$

Defining the neural-network input layer

- At a given time t, the PDA is uniquely described by
  - Its state
  - The content of stack 1
  - The content of stack 2
- We want to be able to express the description of the PDA at time t as a neural-network layer
- To do so, consider the tuple $(x_0, x_1, x_h, q_1, q_2) \in \mathbb{Q}^5$, where:
  - $(x_0, x_1, x_h) = \begin{cases} (1,0,0) & \text{if the PDA is in state } s_0 \\ (0,1,0) & \text{if the PDA is in state } s_1 \\ (0,0,1) & \text{if the PDA is in state } s_h \end{cases}$
  - $q_{1,2} \in [0,1)$ is the rational encoding of the content of stack $1, 2$

Defining the neural-network input layer

- At a given time t, the PDA is uniquely described by
  - Its state
  - The content of stack 1
  - The content of stack 2
- We want to be able to express the description of the PDA at time t as a neural-network layer
- To do so, consider the tuple $(x_0, x_1, x_h, q_1, q_2) \in \mathbb{Q}^5$, where:
  - $(x_0, x_1, x_h) = \begin{cases} (1,0,0) & \text{if the PDA is in state } s_0 \\ (0,1,0) & \text{if the PDA is in state } s_1 \\ (0,0,1) & \text{if the PDA is in state } s_h \end{cases}$
  - $q_{1,2} \in [0,1)$ is the rational encoding of the content of stack $1, 2$
- Therefore, $(x_0, x_1, x_h, q_1, q_2)$ will be the input layer of the neural network we are building

Defining the neural-network output layer

- Let $(x_0^+, x_1^+, x_\mathrm{h}^+, q_1^+, q_2^+)$ be the description of the PDA at time $t+1$

Defining the neural-network output layer

- Let $(x_0^+, x_1^+, x_\mathrm{h}^+, q_1^+, q_2^+)$ be the description of the PDA at time $t+1$
- We would like $(x_0^+, x_1^+, x_\mathrm{h}^+, q_1^+, q_2^+)$ to be the output layer of our network

Defining the neural-network output layer

- Let $(x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$ be the description of the PDA at time $t+1$
- We would like $(x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$ to be the output layer of our network
- Is it possible to go from input to output layer, that is

$$(x_0, x_1, x_h, q_1, q_2) \longrightarrow (x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$$

using only neural-network operations?

Defining the neural-network output layer

- Let $(x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$ be the description of the PDA at time $t+1$
- We would like $(x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$ to be the output layer of our network
- Is it possible to go from input to output layer, that is

$$(x_0, x_1, x_h, q_1, q_2) \longrightarrow (x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$$

  using only neural-network operations?
- Namely, something like $\sigma(Wa + b)$, where:

Defining the neural-network output layer

- Let $(x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$ be the description of the PDA at time $t+1$
- We would like $(x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$ to be the output layer of our network
- Is it possible to go from input to output layer, that is

$$(x_0, x_1, x_h, q_1, q_2) \longrightarrow (x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$$

  using only neural-network operations?
- Namely, something like $\sigma(Wa + b)$, where:
  - σ is a sigmoid function

Defining the neural-network output layer

- Let $(x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$ be the description of the PDA at time $t+1$
- We would like $(x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$ to be the output layer of our network
- Is it possible to go from input to output layer, that is

$$(x_0, x_1, x_h, q_1, q_2) \longrightarrow (x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$$

  using only neural-network operations?
- Namely, something like $\sigma(Wa + b)$, where:
  - σ is a sigmoid function
  - W is a matrix

Defining the neural-network output layer

- Let $(x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$ be the description of the PDA at time $t+1$
- We would like $(x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$ to be the output layer of our network
- Is it possible to go from input to output layer, that is

$$(x_0, x_1, x_h, q_1, q_2) \longrightarrow (x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$$

  using only neural-network operations?
- Namely, something like $\sigma(Wa + b)$, where:
  - σ is a sigmoid function
  - W is a matrix
  - a and b are vectors

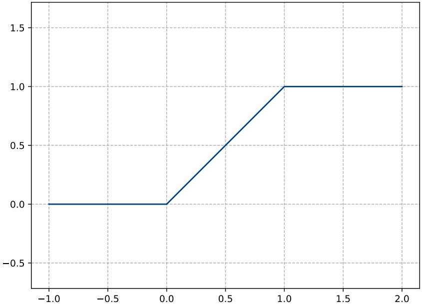
- Yes, it is!

- Yes, it is!
- Let us start from x_0^+:

- Yes, it is!
- Let us start from x_0^+:

$$x_0^+ = \sigma(x_0^+)$$

where σ is

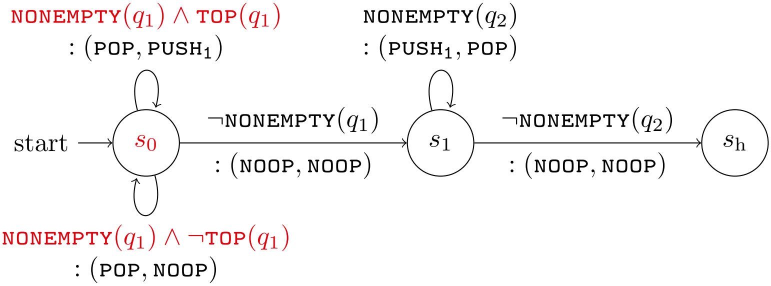


(remember that $x_0^+ \in \{0, 1\}$)

- Yes, it is!
- Let us start from x_0^+:

$$
\begin{aligned}
x_0^+ &= \sigma(x_0^+) \\
&= \sigma(x_0 \cdot \texttt{NONEMPTY}(q_1) \cdot \texttt{TOP}(q_1) + \\
&\qquad x_0 \cdot \texttt{NONEMPTY}(q_1) \cdot \neg\texttt{TOP}(q_1))
\end{aligned}
$$

since



- Yes, it is!
- Let us start from x_0^+:

$$
\begin{aligned}
x_0^+ &= \sigma(x_0^+) \\
&= \sigma(x_0 \cdot \texttt{NONEMPTY}(q_1) \cdot \texttt{TOP}(q_1) + \\
&\qquad x_0 \cdot \texttt{NONEMPTY}(q_1) \cdot \neg\texttt{TOP}(q_1)) \\
&= \sigma(\sigma(x_0 + \texttt{NONEMPTY}(q_1) + \texttt{TOP}(q_1) - 2) + \\
&\qquad \sigma(x_0 + \texttt{NONEMPTY}(q_1) + \neg\texttt{TOP}(q_1) - 2))
\end{aligned}
$$

because of the following lemma:

Lemma

Let $a_1, a_2, \ldots, a_k \in \{0, 1\}$. Then

$$a_1 a_2 \ldots a_k = \sigma(a_1 + a_2 + \cdots + a_k - k + 1)$$

- Yes, it is!
- Let us start from x_0^+:

$$
\begin{aligned}
x_0^+ &= \sigma(x_0^+) \\
&= \sigma(x_0 \cdot \textsc{nonempty}(q_1) \cdot \textsc{top}(q_1) + \\
&\qquad x_0 \cdot \textsc{nonempty}(q_1) \cdot \neg\textsc{top}(q_1)) \\
&= \sigma(\sigma(x_0 + \textsc{nonempty}(q_1) + \textsc{top}(q_1) - 2) + \\
&\qquad \sigma(x_0 + \textsc{nonempty}(q_1) + \neg\textsc{top}(q_1) - 2)) \\
&= \sigma(\sigma(x_0 + \sigma(4q_1) + \sigma(4q_1 - 2) - 2) + \\
&\qquad \sigma(x_0 + \sigma(4q_1) + 1 - \sigma(q_1 - 2) - 2))
\end{aligned}
$$

because of the following theorem:

Theorem

Let q be the rational encoding of a stack content. Then

$$\textsc{top}(q) = \sigma(4q - 2), \quad \textsc{nonempty}(q) = \sigma(4q)$$

- Yes, it is!
- Let us start from x_0^+:

$$
\begin{aligned}
x_0^+ &= \sigma(x_0^+) \\
&= \sigma(x_0 \cdot \texttt{NONEMPTY}(q_1) \cdot \texttt{TOP}(q_1) + \\
&\qquad x_0 \cdot \texttt{NONEMPTY}(q_1) \cdot \neg\texttt{TOP}(q_1)) \\
&= \sigma(\sigma(x_0 + \texttt{NONEMPTY}(q_1) + \texttt{TOP}(q_1) - 2) + \\
&\qquad \sigma(x_0 + \texttt{NONEMPTY}(q_1) + \neg\texttt{TOP}(q_1) - 2)) \\
&= \sigma(\sigma(x_0 + \sigma(4q_1) + \sigma(4q_1 - 2) - 2) + \\
&\qquad \sigma(x_0 + \sigma(4q_1) - \sigma(q_1 - 2) - 1))
\end{aligned}
$$

Input layer $\rightarrow$ output layer

- We did it! We computed x_0^+ using only σ and linear combinations of input-layer activations $(x_0, x_1, x_\mathsf{h}, q_1, q_2)$

Input layer $\rightarrow$ output layer

- We did it! We computed x_0^+ using only σ and linear combinations of input-layer activations $(x_0, x_1, x_\mathrm{h}, q_1, q_2)$
  - In particular, we used x_0 and q_1

Input layer $\rightarrow$ output layer

- We did it! We computed x_0^+ using only σ and linear combinations of input-layer activations $(x_0, x_1, x_\mathrm{h}, q_1, q_2)$
  - In particular, we used x_0 and q_1
- From the expression we got, though, we can see that we will not be able to go from $(x_0, x_1, x_\mathrm{h}, q_1, q_2)$ to x_0^+ in just one step

Input layer → output layer

- We did it! We computed x_0^+ using only σ and linear combinations of input-layer activations $(x_0, x_1, x_\mathrm{h}, q_1, q_2)$
  - In particular, we used x_0 and q_1
- From the expression we got, though, we can see that we will not be able to go from $(x_0, x_1, x_\mathrm{h}, q_1, q_2)$ to x_0^+ in just one step
- Instead, we will need some inner (i.e. intermediate) layers $\ell_{1,2}$:

Input layer $\to$ output layer

- We did it! We computed x_0^+ using only σ and linear combinations of input-layer activations $(x_0, x_1, x_h, q_1, q_2)$
  - In particular, we used x_0 and q_1
- From the expression we got, though, we can see that we will not be able to go from $(x_0, x_1, x_h, q_1, q_2)$ to x_0^+ in just one step
- Instead, we will need some inner (i.e. intermediate) layers $\ell_{1,2}$:

$$x_0^+ = \underbrace{\sigma(\underbrace{\sigma(\overset{\overset{\text{input}}{\uparrow}}{x_0} + \underbrace{\sigma(4\overset{\overset{\text{input}}{\uparrow}}{q_1})}_{\ell_1} + \underbrace{\sigma(4\overset{\overset{\text{input}}{\uparrow}}{q_1} - 2)}_{\ell_1} - 2)}_{\ell_2} + \underbrace{\sigma(\overset{\overset{\text{input}}{\uparrow}}{x_0} + \underbrace{\sigma(4\overset{\overset{\text{input}}{\uparrow}}{q_1})}_{\ell_1} - \underbrace{\sigma(4\overset{\overset{\text{input}}{\uparrow}}{q_1} - 2)}_{\ell_1} - 1)}_{\ell_2})}_{\text{output}}$$

Input layer $\rightarrow$ output layer

- We can do the same for:

Input layer $\rightarrow$ output layer

- We can do the same for:

$$(x_0, x_1, x_h, q_1, q_2) \longrightarrow x_1^+$$

Input layer $\rightarrow$ output layer

- We can do the same for:

$$(x_0, x_1, x_{\mathrm{h}}, q_1, q_2) \longrightarrow x_1^+$$
$$(x_0, x_1, x_{\mathrm{h}}, q_1, q_2) \longrightarrow x_{\mathrm{h}}^+$$

Input layer $\to$ output layer

- We can do the same for:

$$(x_0, x_1, x_h, q_1, q_2) \longrightarrow x_1^+$$
$$(x_0, x_1, x_h, q_1, q_2) \longrightarrow x_h^+$$
$$(x_0, x_1, x_h, q_1, q_2) \longrightarrow q_1^+$$

Input layer $\rightarrow$ output layer

- We can do the same for:

$$
\begin{aligned}
(x_0, x_1, x_h, q_1, q_2) &\longrightarrow x_1^+ \\
(x_0, x_1, x_h, q_1, q_2) &\longrightarrow x_h^+ \\
(x_0, x_1, x_h, q_1, q_2) &\longrightarrow q_1^+ \\
(x_0, x_1, x_h, q_1, q_2) &\longrightarrow q_2^+
\end{aligned}
$$

Input layer $\rightarrow$ output layer

- We can do the same for:

$$(x_0, x_1, x_h, q_1, q_2) \longrightarrow x_1^+$$
$$(x_0, x_1, x_h, q_1, q_2) \longrightarrow x_h^+$$
$$(x_0, x_1, x_h, q_1, q_2) \longrightarrow q_1^+$$
$$(x_0, x_1, x_h, q_1, q_2) \longrightarrow q_2^+$$

- Let us skip the calculations (which are similar to those we have already seen anyway) and just show the final results:

Input layer $\rightarrow$ output layer

$$x_1^+ = \cdots = \sigma(\sigma(x_0 - \sigma(4q_1)) + \sigma(x_1 + \sigma(4q_2) - 1))$$

Input layer $\to$ output layer

$$x_1^+ = \cdots = \sigma(\sigma(x_0 - \sigma(4q_1)) + \sigma(x_1 + \sigma(4q_2) - 1))$$
$$x_h^+ = \cdots = \sigma(\sigma(x_1 - \sigma(4q_2)))$$

Input layer $\rightarrow$ output layer

$$
\begin{aligned}
x_1^+ &= \cdots = \sigma(\sigma(x_0 - \sigma(4q_1)) + \sigma(x_1 + \sigma(4q_2) - 1)) \\
x_h^+ &= \cdots = \sigma(\sigma(x_1 - \sigma(4q_2))) \\
q_1^+ &= \cdots = \sigma\big(\sigma(x_0 + \sigma(4q_1) - \sigma(4q_1 - 2) + 4q_1 - 4) + \\
&\qquad\qquad \sigma(x_0 + \sigma(4q_1) - 3\sigma(4q_1 - 2) + 4q_1 - 3) + \\
&\qquad\qquad \sigma(x_0 - \sigma(4q_1) + q_1 - 1) + \\
&\qquad\qquad \sigma\big(x_1 + \sigma(4q_2) + \tfrac{1}{4}q_1 - \tfrac{5}{4}\big) + \\
&\qquad\qquad \sigma(x_1 - \sigma(4q_2) + q_1 - 1)\big)
\end{aligned}
$$

Input layer $\to$ output layer

$$
\begin{aligned}
x_1^+ &= \cdots = \sigma(\sigma(x_0 - \sigma(4q_1)) + \sigma(x_1 + \sigma(4q_2) - 1)) \\
x_h^+ &= \cdots = \sigma(\sigma(x_1 - \sigma(4q_2))) \\
q_1^+ &= \cdots = \sigma\big(\sigma(x_0 + \sigma(4q_1) - \sigma(4q_1 - 2) + 4q_1 - 4) + \\
&\qquad\quad \sigma(x_0 + \sigma(4q_1) - 3\sigma(4q_1 - 2) + 4q_1 - 3) + \\
&\qquad\quad \sigma(x_0 - \sigma(4q_1) + q_1 - 1) + \\
&\qquad\quad \sigma\big(x_1 + \sigma(4q_2) + \tfrac{1}{4}q_1 - \tfrac{5}{4}\big) + \\
&\qquad\quad \sigma(x_1 - \sigma(4q_2) + q_1 - 1)\big) \\
q_2^+ &= \cdots = \sigma\big(\sigma\big(x_0 + \sigma(4q_1) - \sigma(4q_1 - 2) + \tfrac{1}{4}q_2 - \tfrac{9}{4}\big) + \\
&\qquad\quad \sigma(x_0 + \sigma(4q_1) - \sigma(4q_1 - 2) + q_2 - 2) + \\
&\qquad\quad \sigma(x_0 - \sigma(4q_1) + q_2 - 1) + \\
&\qquad\quad \sigma(x_1 + \sigma(4q_2) - 2\sigma(4q_2 - 2) + 4q_2 - 3) + \\
&\qquad\quad \sigma(x_1 - \sigma(4q_2) + q_2 - 1)\big)
\end{aligned}
$$

The network

- We are done: we built a neural network whose input layer is the description of the PDA at time t, namely,

$$(x_0, x_1, x_{\mathrm{h}}, q_1, q_2)$$

The network

- We are done: we built a neural network whose input layer is the description of the PDA at time t, namely,

$$(x_0, x_1, x_\mathrm{h}, q_1, q_2)$$

and whose output layer is the description of the PDA at time $t+1$, that is,

$$(x_0^+, x_1^+, x_\mathrm{h}^+, q_1^+, q_2^+)$$

The network

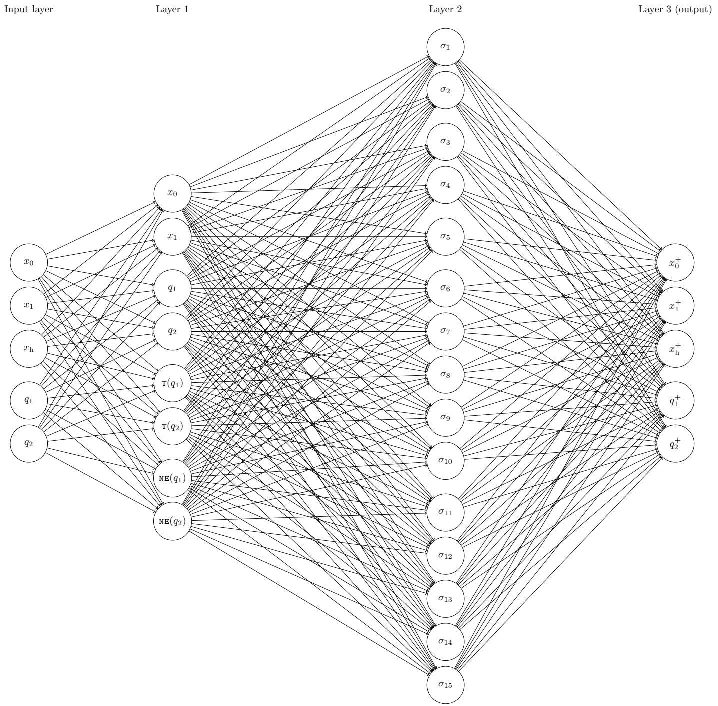
- We are done: we built a neural network whose input layer is the description of the PDA at time t, namely,

$$(x_0, x_1, x_{\mathrm{h}}, q_1, q_2)$$

and whose output layer is the description of the PDA at time $t+1$, that is,

$$(x_0^+, x_1^+, x_{\mathrm{h}}^+, q_1^+, q_2^+)$$

- Let us look at its graphical depiction:



Input layer
Layer 1
Layer 2
Layer 3 (output)
x_0
x_1
x_h
q_1
q_2
x_0
x_1
q_1
q_2
$T(q_1)$
$T(q_2)$
$NE(q_1)$
$NE(q_2)$
σ_1
σ_2
σ_3
σ_4
σ_5
σ_6
σ_7
σ_8
σ_9
σ_{10}
σ_{11}
σ_{12}
σ_{13}
σ_{14}
σ_{15}
x_0^+
x_1^+
x_h^+
q_1^+
q_2^+

Simulation

Now that we have built the network, how do we simulate a PDA execution with it?

Simulation

Now that we have built the network, how do we simulate a PDA execution with it?

Simulation algorithm

1. Initialize the network with input layer

$$(x_0, x_1, x_h, q_1, q_2) = (1, 0, 0, q_1, q_2)$$

Simulation

Now that we have built the network, how do we simulate a PDA execution with it?

Simulation algorithm

1. Initialize the network with input layer

$$(x_0, x_1, x_h, q_1, q_2) = (1, 0, 0, q_1, q_2)$$

   where $q_{1,2}$ is the rational encoding of the content of stack $1, 2$ at the beginning of the PDA execution

Simulation

Now that we have built the network, how do we simulate a PDA execution with it?

Simulation algorithm

1. Initialize the network with input layer

$$(x_0, x_1, x_h, q_1, q_2) = (1, 0, 0, q_1, q_2)$$

   where $q_{1,2}$ is the rational encoding of the content of stack $1, 2$ at the beginning of the PDA execution

2. Compute the output layer $(x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$

Simulation

Now that we have built the network, how do we simulate a PDA execution with it?

Simulation algorithm

1. Initialize the network with input layer

$$(x_0, x_1, x_h, q_1, q_2) = (1, 0, 0, q_1, q_2)$$

   where $q_{1,2}$ is the rational encoding of the content of stack $1, 2$ at the beginning of the PDA execution
2. Compute the output layer $(x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$
3. If $x_h^+ = 0$

Simulation

Now that we have built the network, how do we simulate a PDA execution with it?

Simulation algorithm

1. Initialize the network with input layer

$$(x_0, x_1, x_{\mathrm{h}}, q_1, q_2) = (1, 0, 0, q_1, q_2)$$

   where $q_{1,2}$ is the rational encoding of the content of stack $1, 2$ at the beginning of the PDA execution

2. Compute the output layer $(x_0^+, x_1^+, x_{\mathrm{h}}^+, q_1^+, q_2^+)$
3. If $x_{\mathrm{h}}^+ = 0$
   - Copy the output layer into the input layer

Simulation

Now that we have built the network, how do we simulate a PDA execution with it?

Simulation algorithm

1. Initialize the network with input layer

$$(x_0, x_1, x_{\mathrm{h}}, q_1, q_2) = (1, 0, 0, q_1, q_2)$$

   where $q_{1,2}$ is the rational encoding of the content of stack $1, 2$ at the beginning of the PDA execution
2. Compute the output layer $(x_0^+, x_1^+, x_{\mathrm{h}}^+, q_1^+, q_2^+)$
3. If $x_{\mathrm{h}}^+ = 0$
   - Copy the output layer into the input layer
   - Go back to point 2

Simulation

Now that we have built the network, how do we simulate a PDA execution with it?

Simulation algorithm

1. Initialize the network with input layer

$$(x_0, x_1, x_{\mathrm{h}}, q_1, q_2) = (1, 0, 0, q_1, q_2)$$

   where $q_{1,2}$ is the rational encoding of the content of stack $1, 2$ at the beginning of the PDA execution
2. Compute the output layer $(x_0^+, x_1^+, x_{\mathrm{h}}^+, q_1^+, q_2^+)$
3. If $x_{\mathrm{h}}^+ = 0$
   - Copy the output layer into the input layer
   - Go back to point 2

   If $x_{\mathrm{h}}^+ = 1$,

Simulation

Now that we have built the network, how do we simulate a PDA execution with it?

Simulation algorithm

1. Initialize the network with input layer

$$(x_0, x_1, x_h, q_1, q_2) = (1, 0, 0, q_1, q_2)$$

   where $q_{1,2}$ is the rational encoding of the content of stack $1, 2$ at the beginning of the PDA execution

2. Compute the output layer $(x_0^+, x_1^+, x_h^+, q_1^+, q_2^+)$
3. If $x_h^+ = 0$
   - Copy the output layer into the input layer
   - Go back to point 2

   If $x_h^+ = 1$, then $q_{1,2}^+$ is the rational encoding of the content of stack $1, 2$ at the end of the PDA execution

Implementation (Python 3 + NumPy)

Fragment of class `Network`

```
def __init__(self):
    self.weights = [
        numpy.array(...),  # matrix W1
        numpy.array(...),  # matrix W2
        numpy.array(...)   # matrix W3
    ]
    self.biases = [
        numpy.array(...),  # vector b1
        numpy.array(...),  # vector b2
        numpy.array(...)   # vector b3
    ]
```

Implementation (Python 3 + NumPy)

Fragment of class `Network`

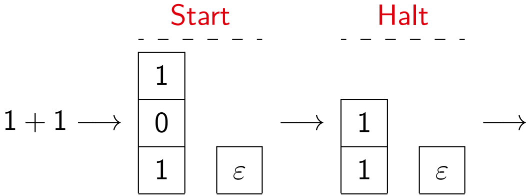
```
def execute(self, stack1):

    def feedforward(a):
        if a[2] == 1:
            return a[3], a[4]
        for w, b in zip(self.weights, self.biases):
            a = sigmoid(numpy.dot(w, a) + b)
        return feedforward(a)

    a = numpy.array(
        [1, 0, 0, Stack(stack1).encoding, Stack([]).encoding]
    )
    return feedforward(a)
```

Test

- What to expect:

Test

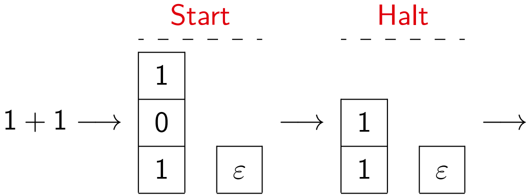
- What to expect:



$$q_1 = \frac{2 \cdot 1}{4} + \frac{2 \cdot 1}{4^2} = 0.9375$$

$$q_2 = 0$$

- What to expect:

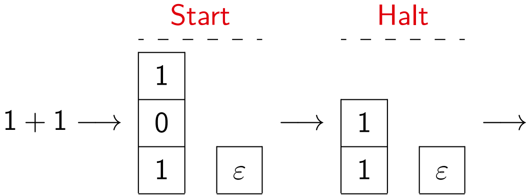


$$q_1 = \frac{2\cdot 1}{4} + \frac{2\cdot 1}{4^2} = 0.9375$$
$$q_2 = 0$$

- What we get:

Test

- What to expect:



$$q_1 = \frac{2\cdot 1}{4} + \frac{2\cdot 1}{4^2} = 0.9375$$
$$q_2 = 0$$

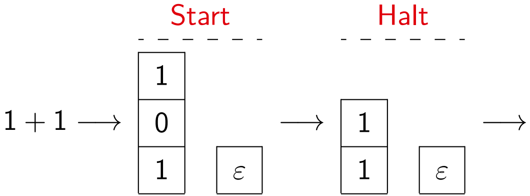
- What we get:

Test

```
>>> Network().execute([1, 0, 1])
```

Test

- What to expect:



$$q_1 = \frac{2 \cdot 1}{4} + \frac{2 \cdot 1}{4^2} = 0.9375$$
$$q_2 = 0$$

- What we get:

Test

```
>>> Network().execute([1, 0, 1])
(0.9375, 0.0)
```

Conclusions

Critical observations and limits of our construction

Let us consider some implementation issues:

Critical observations and limits of our construction

Let us consider some implementation issues:

- In order to be be able to represent any rational number, no matter with how many digits, we would need arbitrary-precision arithmetic and an infinite amount of memory

Critical observations and limits of our construction

Let us consider some implementation issues:

- In order to be be able to represent any rational number, no matter with how many digits, we would need arbitrary-precision arithmetic and an infinite amount of memory
  - But this is just the Turing machine infinite-tape requirement in a different guise

Critical observations and limits of our construction

Let us consider some implementation issues:

- In order to be be able to represent any rational number, no matter with how many digits, we would need arbitrary-precision arithmetic and an infinite amount of memory
  - But this is just the Turing machine infinite-tape requirement in a different guise
  - And it comes up because we encoded stack contents as rational numbers but we didn't put a limit on stack size

Critical observations and limits of our construction

Let us consider some implementation issues:

- In order to be be able to represent any rational number, no matter with how many digits, we would need arbitrary-precision arithmetic and an infinite amount of memory
  - But this is just the Turing machine infinite-tape requirement in a different guise
  - And it comes up because we encoded stack contents as rational numbers but we didn't put a limit on stack size
- In the example we saw, the derivation of the network was carried on "by hand", and the implementation needed precomputed weight matrices and bias vectors

Critical observations and limits of our construction

Let us consider some implementation issues:

- In order to be be able to represent any rational number, no matter with how many digits, we would need arbitrary-precision arithmetic and an infinite amount of memory
  - But this is just the Turing machine infinite-tape requirement in a different guise
  - And it comes up because we encoded stack contents as rational numbers but we didn't put a limit on stack size
- In the example we saw, the derivation of the network was carried on "by hand", and the implementation needed precomputed weight matrices and bias vectors
  - One might think of automating the construction by devising a program that derives the weight matrices and bias vectors from any given double-stack PDA, and then runs a simulation

Bibliography

Hava T. Siegelmann, Eduardo D. Sontag
On the Computational Power of Neural Nets
Proceedings of the fifth Annual ACM Workshop on Computational Learning Theory (COLT 1992), 440–449
http://binds.cs.umass.edu/papers/1992_Siegelmann_COLT.pdf

Benjamin Wilson
Siegelmann & Sontag's "On the Computational Power of Neural Nets"
Sydney Machine Learning Meetup, 2018
http://drive.google.com/file/d/1HR-dXSI-dX16yibiXeeiz4pP2D8_KYDl/view

Michael A. Nielsen
Neural Networks and Deep Learning
Determination Press, 2015
http://neuralnetworksanddeeplearning.com

Lemma (Properties of q)

Let $a_1 a_2 \dots a_n \in \{0,1\}^*$ *and* $q = \sum_{i=1}^{n} \frac{2a_i + 1}{4^i} \in \mathbb{Q}$. *Then*

1. $q \in [0, 1)$
2. $q = 0 \iff \omega = \varepsilon$ *(ε is the empty string)*
3. $q \in \left[\frac{1}{4}, \frac{1}{2}\right) \iff a_1 = 0$
4. $q \in \left[\frac{3}{4}, 1\right) \iff a_1 = 1$

Theorem (stack-operation encodings)

1. $\textsc{top}(q) = \sigma(4q - 2)$
2. $\textsc{nonempty}(q) = \sigma(4q)$
3. $\textsc{push}_0(q) = \frac{q}{4} + \frac{1}{4}$
4. $\textsc{push}_1(q) = \frac{q}{4} + \frac{3}{4}$
5. $\textsc{pop}(q) = 4q - 2\sigma(4q - 2) - 1$
6. $\textsc{noop}(q) = q$

Proof.

$$
\begin{aligned}
\textsc{top}(q) = 0 &\implies q \in \left[\tfrac{1}{4}, \tfrac{1}{2}\right) \\
&\implies 4q - 2 \in [-1, 0) \\
&\implies \sigma(4q - 2) = 0 \\
\textsc{top}(q) = 1 &\implies q \in \left[\tfrac{3}{4}, 1\right) \\
&\implies 4q - 2 \in [1, 2) \\
&\implies \sigma(4q - 2) = 1 \\
\textsc{nonempty}(q) = 0 &\implies q = 0 \\
&\implies 4q = 0 \\
&\implies \sigma(4q) = 0 \\
\textsc{nonempty}(q) = 1 &\implies q \neq 0 \\
&\implies q \in \left[\tfrac{1}{4}, \tfrac{1}{2}\right) \cup \left[\tfrac{3}{4}, 1\right) \\
&\implies 4q \in [1, 2) \cup [3, 4) \\
&\implies \sigma(4q) = 1
\end{aligned}
$$

$$
\begin{aligned}
\textsc{push}_0(q) &= \frac{q}{4} + \frac{2 \cdot 0 + 1}{4} \\
&= \frac{q}{4} + \frac{1}{4} \\
\textsc{push}_1(q) &= \frac{q}{4} + \frac{2 \cdot 1 + 1}{4} \\
&= \frac{q}{4} + \frac{3}{4} \\
\textsc{pop}(q) &= 4q - (2\textsc{top}(q) + 1) \\
&= 4q - 2\sigma(4q - 2) - 1
\end{aligned}
$$

$\square$

Lemma (technical)

Let $a_1, a_2, \ldots, a_k \in \{0, 1\}$ *and* $q \in [0, 1]$. *Then*

$$a_1 a_2 \ldots a_k q = \sigma(a_1 + a_2 + \cdots + a_k - k + q)$$

Proof.

Case 1: $\exists\, i : a_i = 0$

$$\begin{aligned}
&\implies a_1 + a_2 + \cdots + a_k \leq k - 1 \\
&\implies a_1 + a_2 + \cdots + a_k - k + q \leq k - 1 - k + q = q - 1 \in [-1, 0] \\
&\implies \sigma(a_1 + a_2 + \cdots + a_k - k + q) = 0 = a_1 a_2 \ldots a_k q
\end{aligned}$$

Case 2: $a_i = 1\ \forall\, i$

$$\begin{aligned}
&\implies a_1 + a_2 + \cdots + a_k = k \\
&\implies a_1 + a_2 + \cdots + a_k - k + q = k - k + q = q \in [0, 1] \\
&\implies \sigma(a_1 + a_2 + \cdots + a_k - k + q) = q = a_1 a_2 \ldots a_k q
\end{aligned}$$

$\square$

$$
\begin{aligned}
x_1^+ &= \sigma\left(x_1^+\right) \\
&= \sigma(x_0 \cdot \neg\texttt{NONEMPTY}(q_1) + x_1 \cdot \texttt{NONEMPTY}(q_2)) \\
&= \sigma(\sigma(x_0 + \neg\texttt{NONEMPTY}(q_1) - 1) + \sigma(x_1 + \texttt{NONEMPTY}(q_2) - 1)) \\
&= \sigma(\sigma(x_0 + 1 - \sigma(4q_1) - 1) + \sigma(x_1 + \sigma(4q_2) - 1)) \\
&= \sigma(\sigma(x_0 - \sigma(4q_1)) + \sigma(x_1 + \sigma(4q_2) - 1))
\end{aligned}
$$

$$
\begin{aligned}
x_{\mathsf{h}}^+ &= \sigma\left(x_{\mathsf{h}}^+\right) \\
&= \sigma(x_1 \cdot \neg\texttt{NONEMPTY}(q_2)) \\
&= \sigma(\sigma(x_1 + \neg\texttt{NONEMPTY}(q_2) - 1)) \\
&= \sigma(\sigma(x_1 + 1 - \sigma(4q_2) - 1)) \\
&= \sigma(\sigma(x_1 - \sigma(4q_2)))
\end{aligned}
$$

$$
\begin{aligned}
q_1^+ &= \sigma(q_1^+) \\
&= \sigma(x_0 \cdot \texttt{NONEMPTY}(q_1) \cdot \texttt{TOP}(q_1) \cdot \texttt{POP}(q_1) + \\
&\qquad x_0 \cdot \texttt{NONEMPTY}(q_1) \cdot \neg\texttt{TOP}(q_1) \cdot \texttt{POP}(q_1) + \\
&\qquad x_0 \cdot \neg\texttt{NONEMPTY}(q_1) \cdot \texttt{NOOP}(q_1) + \\
&\qquad x_1 \cdot \texttt{NONEMPTY}(q_2) \cdot \texttt{PUSH}_1(q_1) + \\
&\qquad x_1 \cdot \neg\texttt{NONEMPTY}(q_2) \cdot \texttt{NOOP}(q_1)) \\
&= \sigma(\sigma(x_0 + \texttt{NONEMPTY}(q_1) + \texttt{TOP}(q_1) - 3 + \texttt{POP}(q_1)) + \\
&\qquad \sigma(x_0 + \texttt{NONEMPTY}(q_1) + \neg\texttt{TOP}(q_1) - 3 + \texttt{POP}(q_1)) + \\
&\qquad \sigma(x_0 + \neg\texttt{NONEMPTY}(q_1) - 2 + \texttt{NOOP}(q_1)) + \\
&\qquad \sigma(x_1 + \texttt{NONEMPTY}(q_2) - 2 + \texttt{PUSH}_1(q_1)) + \\
&\qquad \sigma(x_1 + \neg\texttt{NONEMPTY}(q_2) - 2 + \texttt{NOOP}(q_1))) \\
&= \sigma(\sigma(x_0 + \sigma(4q_1) + \sigma(4q_1 - 2) - 3 + 4q_1 - 2\sigma(4q_1 - 2) - 1) + \\
&\qquad \sigma(x_0 + \sigma(4q_1) + 1 - \sigma(4q_1 - 2) - 3 + 4q_1 - 2\sigma(4q_1 - 2) - 1) + \\
&\qquad \sigma(x_0 + 1 - \sigma(4q_1) - 2 + q_1) + \\
&\qquad \sigma(x_1 + \sigma(4q_2) - 2 + \tfrac{1}{4}q_1 + \tfrac{3}{4}) + \\
&\qquad \sigma(x_1 + 1 - \sigma(4q_2) - 2 + q_1)) \\
&= \sigma(\sigma(x_0 + \sigma(4q_1) - \sigma(4q_1 - 2) + 4q_1 - 4) + \\
&\qquad \sigma(x_0 + \sigma(4q_1) - 3\sigma(4q_1 - 2) + 4q_1 - 3) + \\
&\qquad \sigma(x_0 - \sigma(4q_1) + q_1 - 1) + \\
&\qquad \sigma(x_1 + \sigma(4q_2) + \tfrac{1}{4}q_1 - \tfrac{5}{4}) + \\
&\qquad \sigma(x_1 - \sigma(4q_2) + q_1 - 1))
\end{aligned}
$$

$$
\begin{aligned}
q_2^+ &= \sigma(q_2^+) \\
&= \sigma(x_0 \cdot \text{NONEMPTY}(q_1) \cdot \text{TOP}(q_1) \cdot \text{PUSH}_1(q_2) + \\
&\qquad x_0 \cdot \text{NONEMPTY}(q_1) \cdot \neg\text{TOP}(q_1) \cdot \text{NOOP}(q_2) + \\
&\qquad x_0 \cdot \neg\text{NONEMPTY}(q_1) \cdot \text{NOOP}(q_2) + \\
&\qquad x_1 \cdot \text{NONEMPTY}(q_2) \cdot \text{POP}(q_2) + \\
&\qquad x_1 \cdot \neg\text{NONEMPTY}(q_2) \cdot \text{NOOP}(q_2)) \\
&= \sigma(\sigma(x_0 + \text{NONEMPTY}(q_1) + \text{TOP}(q_1) - 3 + \text{PUSH}_1(q_2)) + \\
&\qquad \sigma(x_0 + \text{NONEMPTY}(q_1) + \neg\text{TOP}(q_1) - 3 + \text{NOOP}(q_2)) + \\
&\qquad \sigma(x_0 + \neg\text{NONEMPTY}(q_1) - 2 + \text{NOOP}(q_2)) + \\
&\qquad \sigma(x_1 + \text{NONEMPTY}(q_2) - 2 + \text{POP}(q_2)) + \\
&\qquad \sigma(x_1 + \neg\text{NONEMPTY}(q_2) - 2 + \text{NOOP}(q_2))) \\
&= \sigma(\sigma(x_0 + \sigma(4q_1) + \sigma(4q_1 - 2) - 3 + \tfrac{1}{4}q_2 + \tfrac{3}{4}) + \\
&\qquad \sigma(x_0 + \sigma(4q_1) + 1 - \sigma(4q_1 - 2) - 3 + q_2) + \\
&\qquad \sigma(x_0 + 1 - \sigma(4q_1) - 2 + q_2) + \\
&\qquad \sigma(x_1 + \sigma(4q_2) - 2 + 4q_2 - 2\sigma(4q_2 - 2) - 1) + \\
&\qquad \sigma(x_1 + 1 - \sigma(4q_2) - 2 + q_2)) \\
&= \sigma(\sigma(x_0 + \sigma(4q_1) - \sigma(4q_1 - 2) + \tfrac{1}{4}q_2 - \tfrac{9}{4}) + \\
&\qquad \sigma(x_0 + \sigma(4q_1) - \sigma(4q_1 - 2) + q_2 - 2) + \\
&\qquad \sigma(x_0 - \sigma(4q_1) + q_2 - 1) + \\
&\qquad \sigma(x_1 + \sigma(4q_2) - 2\sigma(4q_2 - 2) + 4q_2 - 3) + \\
&\qquad \sigma(x_1 - \sigma(4q_2) + q_2 - 1))
\end{aligned}
$$

$$
\begin{aligned}
x_0^+ &= \sigma\big(\underbrace{\sigma(x_0 + \sigma(4q_1) + \sigma(4q_1 - 2) - 2)}_{=:\,\sigma_1} + \\
&\qquad \underbrace{\sigma(x_0 + \sigma(4q_1) - \sigma(4q_1 - 2) - 1)}_{=:\,\sigma_2}\big) \\
x_1^+ &= \sigma\big(\underbrace{\sigma(x_0 - \sigma(4q_1))}_{=:\,\sigma_3} + \underbrace{\sigma(x_1 + \sigma(4q_2) - 1)}_{=:\,\sigma_4}\big) \\
x_{\mathsf{h}}^+ &= \sigma\big(\underbrace{\sigma(x_1 - \sigma(4q_2))}_{=:\,\sigma_5}\big)
\end{aligned}
$$

$$
\begin{aligned}
q_1^+ = \sigma\Big(& \underbrace{\sigma(x_0 + \sigma(4q_1) - \sigma(4q_1 - 2) + 4q_1 - 4)}_{=:\,\sigma_6} + \\
& \underbrace{\sigma(x_0 + \sigma(4q_1) - 3\sigma(4q_1 - 2) + 4q_1 - 3)}_{=:\,\sigma_7} + \\
& \underbrace{\sigma(x_0 - \sigma(4q_1) + q_1 - 1)}_{=:\,\sigma_8} + \\
& \underbrace{\sigma\big(x_1 + \sigma(4q_2) + \tfrac{1}{4} q_1 - \tfrac{5}{4}\big)}_{=:\,\sigma_9} + \\
& \underbrace{\sigma(x_1 - \sigma(4q_2) + q_1 - 1)}_{=:\,\sigma_{10}} \Big)
\end{aligned}
$$

$$
\begin{aligned}
q_2^+ = \sigma\Big(&\underbrace{\sigma\big(x_0 + \sigma(4q_1) - \sigma(4q_1 - 2) + \tfrac{1}{4}q_2 - \tfrac{9}{4}\big)}_{=:\,\sigma_{11}} + \\
&\underbrace{\sigma\big(x_0 + \sigma(4q_1) - \sigma(4q_1 - 2) + q_2 - 2\big)}_{=:\,\sigma_{12}} + \\
&\underbrace{\sigma\big(x_0 - \sigma(4q_1) + q_2 - 1\big)}_{=:\,\sigma_{13}} + \\
&\underbrace{\sigma\big(x_1 + \sigma(4q_2) - 2\sigma(4q_2 - 2) + 4q_2 - 3\big)}_{=:\,\sigma_{14}} + \\
&\underbrace{\sigma\big(x_1 - \sigma(4q_2) + q_2 - 1\big)}_{=:\,\sigma_{15}}\Big)
\end{aligned}
$$

W^1	x_0	x_1	x_h	q_1	q_2	b^1
x_0	1	0	0	0	0	0
x_1	0	1	0	0	0	0
q_1	0	0	0	1	0	0
q_2	0	0	0	0	1	0
$\texttt{T}(q_1)$	0	0	0	4	0	-2
$\texttt{T}(q_2)$	0	0	0	0	4	-2
$\texttt{NE}(q_1)$	0	0	0	4	0	0
$\texttt{NE}(q_2)$	0	0	0	0	4	0

Table: weights from input layer to layer 1 and biases of layer 1

W^2	x_0	x_1	q_1	q_2	$\texttt{T}(q_1)$	$\texttt{T}(q_2)$	$\texttt{NE}(q_1)$	$\texttt{NE}(q_2)$	b^2
σ_1	1	0	0	0	1	0	1	0	-2
σ_2	1	0	0	0	-1	0	1	0	-1
σ_3	1	0	0	0	0	0	-1	0	0
σ_4	0	1	0	0	0	0	0	1	-1
σ_5	0	1	0	0	0	0	0	-1	0
σ_6	1	0	4	0	-1	0	1	0	-4
σ_7	1	0	4	0	-3	0	1	0	-3
σ_8	1	0	1	0	0	0	-1	0	-1
σ_9	0	1	$\frac{1}{4}$	0	0	0	0	1	$-\frac{5}{4}$
σ_{10}	0	1	1	0	0	0	0	-1	-1
σ_{11}	1	0	0	$\frac{1}{4}$	1	0	1	0	$-\frac{9}{4}$
σ_{12}	1	0	0	1	-1	0	1	0	-2
σ_{13}	1	0	0	1	0	0	-1	0	-1
σ_{14}	0	1	0	4	0	-2	0	1	-3
σ_{15}	0	1	0	1	0	0	0	-1	-1

Table: weights from layer 1 to layer 2 and biases of layer 2

W^3	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	σ_{12}	σ_{13}	σ_{14}	σ_{15}	b^3
x_0^+	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_1^+	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
x_h^+	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
q_1^+	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0
q_2^+	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0

Table: weights from layer 2 to layer 3 and biases of layer 3