On the Turing Completeness of Recurrent Neural Networks

Francesco Ballerini

Tesi di Laurea Triennale in Ingegneria Informatica

Relatore: Andrew David Bagdanov

Università degli Studi di Firenze
Scuola di Ingegneria
a.a. 2018–2019

In the end, we self-perceiving,
self-inventing, locked-in mirages are little
miracles of self-reference.

I Am a Strange Loop
Douglas Hofstadter

Abstract

The purpose of this thesis is to discuss the Turing completeness of recurrent neural networks, which we will model as multilayer feed-forward neural networks feeding their output back into their own input until certain conditions on the output layer are satisfied. In particular, we will prove that, given any binary single-tape Turing machine (actually, any pushdown automaton with two binary stacks) $\mathcal{M}$, it is possible to build a recurrent neural network $\mathcal{N}$ with rational weights, biases, and activations which simulates $\mathcal{M}$: every output generation of $\mathcal{N}$ will correspond to a step in the execution of $\mathcal{M}$. This proof, heavily based on [1], will be constructive, yet, due to its generality, very laborious to apply to any given machine; for this reason, we will then show how to deploy its main ideas in a specific example, also providing an implementation in order to verify, at least in that particular case, the effectiveness of our construction.

Chapter 1

Double-stack pushdown automata

As a departure point, we pick single-tape Turing machines with binary alphabets. As is well-known, by storing separately the parts of the tape to the left and to the right of the head, we may equivalently study pushdown automata with two binary stacks.

1.1 Preliminaries

We choose to represent the content of a stack as a fraction whose denominator is a power of four. More precisely, a binary string

$$\omega := a_1 a_2 \dots a_n \in \{0,1\}^*,$$

which corresponds, from left to right, to the symbols in the stack from top to bottom, is encoded as

$$\begin{aligned}\delta[\omega] &:= \sum_{i=1}^{n} \frac{2a_i + 1}{4^i} \in \mathbb{Q} \qquad (1.1)\\ &= \sum_{i=1}^{n} \frac{b_i}{4^i}\\ &= b_1 \cdot 4^{-1} + b_2 \cdot 4^{-2} + \dots + b_n \cdot 4^{-n}\\ &= [0.b_1 b_2 \dots b_n]_4,\end{aligned}$$

where $b_i := 2a_i + 1$, so $b_i = 1$ if $a_i = 0$ and $b_i = 3$ if $a_i = 1$. That is, we think of the i^{th} symbol from the top of a stack as corresponding to the i^{th} number to the right of the decimal point in a finite expansion in base four; a 0 stored in the stack is associated with a 1 in the expansion, whereas a 1 in the stack is associated with a 3 in the expansion. An empty stack is represented by zero, since, in this case, the sum in equation 1.1 has zero terms.

Note that, for any $\omega \in \{0,1\}^*$,

$$\delta[\omega] \in [0,1);$$

in particular

$$\delta[\omega] = 0 \iff \omega = \varepsilon, \qquad (1.2)$$

where ε denotes the empty string, and for any $\omega = a_1 a_2 \dots a_n \in \{0,1\}^+$

$$\delta[\omega] \in \left[\frac{1}{4}, \frac{1}{2}\right) \iff a_1 = 0 \tag{1.3}$$

$$\delta[\omega] \in \left[\frac{3}{4}, 1\right) \iff a_1 = 1. \tag{1.4}$$

Consider the usual stack tests and operations, translated into functions taking as input the rational encoding of the stack itself:

$$\begin{aligned} \texttt{TOP} &: \mathbb{Q} \to \{0,1\} \\ \texttt{NONEMPTY} &: \mathbb{Q} \to \{0,1\} \\ \texttt{PUSH}_0 &: \mathbb{Q} \to \mathbb{Q} \\ \texttt{PUSH}_1 &: \mathbb{Q} \to \mathbb{Q} \\ \texttt{POP} &: \mathbb{Q} \to \mathbb{Q} \end{aligned}$$

($\texttt{TOP}$ returns the symbol on top of the stack, $\texttt{NONEMPTY}$ returns 0 if the stack is empty and 1 otherwise, $\texttt{PUSH}_0$ and $\texttt{PUSH}_1$ return the rational encoding of the stack after the addition of a 0 and a 1, respectively, at the top of it, and $\texttt{POP}$ returns the rational encoding of the stack after its top symbol has been removed). Let σ be the *sigmoid function* defined as

$$\sigma(x) := \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \leq x \leq 1 \\ 1 & \text{if } x > 1. \end{cases} \tag{1.5}$$

Then, the following theorem holds:

Theorem 1.1. *Let $\omega \in \{0,1\}^*$ and $q := \delta[\omega]$. Then,*

$$\texttt{TOP}(q) = \sigma(4q - 2) \tag{1.6}$$

$$\texttt{NONEMPTY}(q) = \sigma(4q) \tag{1.7}$$

$$\texttt{PUSH}_0(q) = \left(\frac{1}{4}, 0, \frac{1}{4}\right) \cdot (q, \texttt{TOP}(q), 1) \tag{1.8}$$

$$\texttt{PUSH}_1(q) = \left(\frac{1}{4}, 0, \frac{3}{4}\right) \cdot (q, \texttt{TOP}(q), 1) \tag{1.9}$$

$$\texttt{POP}(q) = (4, -2, -1) \cdot (q, \texttt{TOP}(q), 1), \tag{1.10}$$

where $\cdot$ is the dot product in $\mathbb{Q}^3$.

Proof.

$$\begin{aligned}
\texttt{TOP}(q) = 0 &\implies q \in \left[\frac{1}{4}, \frac{1}{2}\right) && \text{(by equation 1.3)}\\
&\implies 4q - 2 \in [-1, 0) \\
&\implies \sigma(4q-2) = 0 && \text{(by equation 1.5)}\\
\texttt{TOP}(q) = 1 &\implies q \in \left[\frac{3}{4}, 1\right) && \text{(by equation 1.4)}\\
&\implies 4q - 2 \in [1, 2) \\
&\implies \sigma(4q-2) = 1 && \text{(by equation 1.5)}\\
\texttt{NONEMPTY}(q) = 0 &\implies q = 0 && \text{(by equation 1.2)}\\
&\implies 4q = 0 \\
&\implies \sigma(4q) = 0 && \text{(by equation 1.5)}\\
\texttt{NONEMPTY}(q) = 1 &\implies q \neq 0 \\
&\implies q \in \left[\frac{1}{4}, \frac{1}{2}\right) \cup \left[\frac{3}{4}, 1\right) && \text{(by equations 1.3 and 1.4)}\\
&\implies 4q \in [1, 2) \cup [3, 4) \\
&\implies \sigma(4q) = 1, && \text{(by equation 1.5)}
\end{aligned}$$

hence

$$\begin{aligned}
\texttt{TOP}(q) &= \sigma(4q-2)\\
\texttt{NONEMPTY}(q) &= \sigma(4q).
\end{aligned}$$

Furthermore,

$$\begin{aligned}
\texttt{PUSH}_0(q) &= \frac{q}{4} + \frac{2 \cdot 0 + 1}{4} && \text{(by equation 1.1)}\\
&= \left(\frac{1}{4}, 0, \frac{1}{4}\right) \cdot (q, \texttt{TOP}(q), 1)\\
\texttt{PUSH}_1(q) &= \frac{q}{4} + \frac{2 \cdot 1 + 1}{4} && \text{(by equation 1.1)}\\
&= \left(\frac{1}{4}, 0, \frac{3}{4}\right) \cdot (q, \texttt{TOP}(q), 1)\\
\texttt{POP}(q) &= 4q - (2\texttt{TOP}(q) + 1) && \text{(by equation 1.1)}\\
&= (4, -2, -1) \cdot (q, \texttt{TOP}(q), 1).
\end{aligned}$$

□

Remark 1.1.1. Note that a binary representation, rather than a base-four one, would not allow such simple stack operations.

1.2 A formalization

We start by giving a more formal structure to some of the concepts introduced in section 1.1:

Definition 1.1. Let Q denote the set defined as follows:

$$Q := \left\{ q \in \mathbb{Q} \;\middle|\; q = \sum_{i=1}^{n} \frac{b_i}{4^i} \text{ for } 0 \le n < \infty \text{ and } b_i \in \{1,3\} \right\},$$

where, when $k = 0$, the sum must be interpreted as $q = 0$, hence $0 \in Q$. Moreover, define function

$$\delta : \{0,1\}^* \to Q$$

as

$$\delta[a_1, a_2, \ldots, a_n] := \sum_{i=1}^{n} \frac{2a_i + 1}{4^i};$$

in particular, $\delta[\varepsilon] = 0$. Note that δ is surjective, i.e.

$$Q = \{ \delta[\omega] \mid \omega \in \{0,1\}^* \};$$

therefore, from now on, we will be talking interchangeably about elements of Q and rational encodings of binary strings.

Definition 1.2. Define functions $\zeta : Q \to \{0,1\}$ and $\tau : Q \to \{0,1\}$ as

$$\begin{aligned} \zeta[q] &:= \begin{cases} 0 & \text{if } q \le \frac{1}{2} \\ 1 & \text{if } q > \frac{1}{2}. \end{cases} \\ \tau[q] &:= \begin{cases} 0 & \text{if } q = 0 \\ 1 & \text{if } q \neq 0. \end{cases} \end{aligned} \tag{1.11}$$

Let $\omega = a_1 a_2 \ldots a_n \in \{0,1\}^+$ be a stack content and let $q = \delta[\omega]$ be its rational encoding. Since

$$\begin{aligned} \zeta[q] = 0 &\iff q \le \frac{1}{2} \\ &\iff a_1 = 0 && \text{(by equation 1.3)} \\ \zeta[q] = 1 &\iff q > \frac{1}{2} \\ &\iff a_1 = 1, && \text{(by equation 1.4)} \end{aligned}$$

we think of ζ as the *top of stack function*; on the other hand, since we encode an empty stack with the number zero, we interpret τ as the *nonempty stack function.* Note that it can never happen that $\zeta[q] = 1$ while $\tau[q] = 0$, so $(\zeta[q], \tau[q]) \in \{(0,0), (0,1), (1,1)\}$ for all $q \in Q$.

We are now ready for the main definitions:

Definition 1.3. A *double-stack pushdown automaton* (double-stack PDA) is a 6-tuple

$$(S, s_0, s_{\mathrm{h}}, \vartheta_0, \vartheta_1, \vartheta_2),$$

where S is a finite set, $s_0, s_{\mathrm{h}} \in S$ are called *initial state* and *halting state*, respectively, and $\vartheta_0, \vartheta_1, \vartheta_2$ are functions defined as

$$\begin{aligned} &\vartheta_0 : S \times \{0,1\}^4 \to S \\ &\vartheta_i : S \times \{0,1\}^4 \to \left\{ (1,0,0), \left(\frac{1}{4}, 0, \frac{1}{4}\right), \left(\frac{1}{4}, 0, \frac{3}{4}\right), (4, -2, -1) \right\} \end{aligned}$$

for $i = 1, 2$. ϑ_0 computes the next state, whereas ϑ_1 and ϑ_2 compute the next operation for the first and second stack, respectively. The triples

$$(1, 0, 0), \left(\frac{1}{4}, 0, \frac{1}{4}\right), \left(\frac{1}{4}, 0, \frac{3}{4}\right), (4, -2, -1)$$

correspond to the stack operations `NOOPERATION`, `PUSH`$_0$, `PUSH`$_1$, and `POP`, respectively (see equations 1.8, 1.9, and 1.10). It is assumed that, for all $s \in S$ and $q_1, q_2 \in Q$,

$$\vartheta_1(s, \zeta[q_1], \zeta[q_2], 0, \tau[q_2]) \neq (4, -2, -1)$$
$$\vartheta_2(s, \zeta[q_1], \zeta[q_2], \tau[q_1], 0) \neq (4, -2, -1),$$

i.e. one does not attempt to pop an empty stack.

Definition 1.4. Let $\mathcal{M} = (S, s_0, s_{\mathrm{h}}, \vartheta_0, \vartheta_1, \vartheta_2)$ be a double-stack PDA. Then,

$$f : S \times Q \times Q \to S \times Q \times Q$$

defined by

$$\begin{aligned} f(s, q_1, q_2) := (&\vartheta_0(s, \zeta[q_1], \zeta[q_2], \tau[q_1], \tau[q_2]),\\ &\vartheta_1(s, \zeta[q_1], \zeta[q_2], \tau[q_1], \tau[q_2]) \cdot (q_1, \zeta[q_1], 1),\\ &\vartheta_2(s, \zeta[q_1], \zeta[q_2], \tau[q_1], \tau[q_2]) \cdot (q_2, \zeta[q_2], 1)) \end{aligned}$$

is called *transition function* of $\mathcal{M}$.

Definition 1.5. Let $\mathcal{M} = (S, s_0, s_{\mathrm{h}}, \vartheta_0, \vartheta_1, \vartheta_2)$ be a double-stack PDA and let $\omega \in \{0, 1\}^*$. We say $\mathcal{M}$ *halts on input* ω if $\mathcal{M}$ reaches the halting state starting with ω on the first stack and nothing on the second, that is,

$$f^k(s_0, \delta[\omega], 0) = (s_{\mathrm{h}}, \delta[\omega_1], \delta[\omega_2])$$

for some $k \in \mathbb{N}_{>0}$ and $\omega_1, \omega_2 \in \{0, 1\}^*$, where f^k is the k^{th} iteration of f.

Definition 1.6. Let $\mathcal{M} = (S, s_0, s_{\mathrm{h}}, \vartheta_0, \vartheta_1, \vartheta_2)$ be a double-stack PDA and let

$$\varphi : \{0, 1\}^* \to \{0, 1\}^*$$

be a partially defined function. We say $\mathcal{M}$ *computes* φ if $\mathcal{M}$ halts on input $\omega \in \{0, 1\}^*$ if and only if $\varphi(\omega)$ is defined, and, in that case, $\varphi(\omega)$ is the content of the first stack when $\mathcal{M}$ halts; that is, $\varphi(\omega)$ is defined if and only if

$$f^k(s_0, \delta[\omega], 0) = (s_{\mathrm{h}}, \delta[\varphi(\omega)], \delta[\omega'])$$

for some $k \in \mathbb{N}_{>0}$ and $\omega' \in \{0, 1\}^*$.

Chapter 2

Neural networks

For the purpose of our discussion, we will model a *neural network* as a *dynamical system* whose state at each instant t is a vector $x(t) \in \mathbb{Q}^n$ with $n \in \mathbb{N}_{>0}$, where the j^{th} coordinate keeps track of the *activation* of the j^{th} *neuron*.

Definition 2.1. A *rational discrete-time dynamical system* (from now on, just *dynamical system*) is specified by an *evolution function*

$$g : \mathbb{Q}^n \to \mathbb{Q}^n,$$

where $n \in \mathbb{N}_{>0}$ is the *dimension* of the system. Given an *initial state*

$$x(0) =: x_0 \in \mathbb{Q}^n,$$

one obtains the *state at time* $t \geq 1$ by recursively solving the equation

$$x(t) := g(x(t-1)),$$

which, from now on, we will write as

$$x^+ = g(x).$$

Definition 2.2. A *rational neural network* (from now on, just *neural network*) of dimension n is a dynamical system with evolution function of form

$$g(x) := \sigma(Wx + b)$$

for some matrix $W \in \mathbb{Q}^{n \times n}$ and (column) vector $b \in \mathbb{Q}^n$. Note that we are adopting the (abuse of) notation

$$\sigma(q_1, q_2, \ldots, q_n) := (\sigma(q_1), \sigma(q_2), \ldots, \sigma(q_n))$$

for any $(q_1, q_2, \ldots, q_n) \in \mathbb{Q}^n$, where the σ in the second member is the one defined by equation 1.5.

A neural network is usually depicted as a graph whose vertices, called *neurons*, are divided into groups, called *layers*, so that each neuron in one layer is connected to every neuron in the following layer. In particular, with reference to figure 2.1 (where we adopt the notation of [2]):

- a_j^ℓ is the *activation* of the j^{th} neuron in the ℓ^{th} layer;
- b_j^ℓ is the *bias* of the j^{th} neuron in the ℓ^{th} layer (note that the first layer, called *input layer*, has no biases);

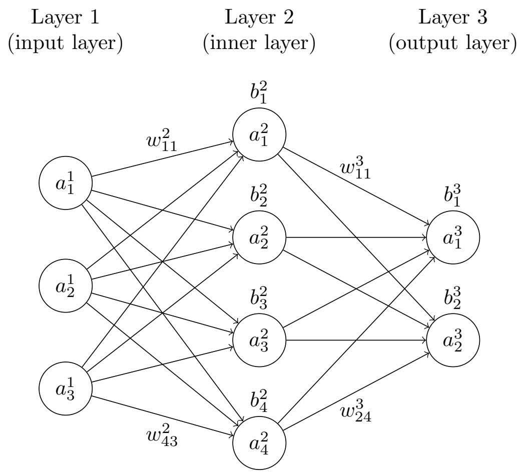

Figure 2.1: a neural network

- w_{jk}^ℓ is the *weight* of the connection from the k^{th} neuron in the $(\ell-1)^{\text{th}}$ layer to the j^{th} neuron in the ℓ^{th} layer.

The activation of each neuron is the output of a sigmoid applied to a linear combination of the bias of that same neuron and the activations of the previous layer multiplied by their respective weights; for example

$$a_3^2 := \sigma\big(w_{31}^2 a_1^1 + w_{32}^2 a_2^1 + w_{33}^2 a_3^1 + b_3^2\big)\,,$$

hence, in general,

$$a_j^\ell := \sigma\left(\sum_{k=1}^{m} w_{jk}^\ell\, a_k^{\ell-1} + b_j^\ell\right), \tag{2.1}$$

where m is the number of neurons in the $(\ell-1)^{\text{th}}$ layer. Therefore, by defining matrix W^ℓ and column vectors a^ℓ, b^ℓ as

$$\begin{aligned}\big(W^\ell\big)_{jk} &:= w_{jk}^\ell\\ \big(a^\ell\big)_j &:= a_j^\ell\\ \big(b^\ell\big)_j &:= b_j^\ell,\end{aligned}$$

respectively, we can rewrite equation 2.1 in the following matrix form:

$$a^\ell = \sigma\big(W^\ell a^{\ell-1} + b^\ell\big)\,.$$

Chapter 3

Simulation of double-stack PDAs through neural networks

As an intermediate step in our construction, we will show how to simulate a double-stack PDA with a generic dynamical system.

Definition 3.1. Let $\mathcal{M} = (S, s_0, s_\mathrm{h}, \vartheta_0, \vartheta_1, \vartheta_2)$ be a double-stack PDA which computes $\varphi : \{0,1\}^* \to \{0,1\}^*$ and let $\mathcal{D}$ be a dynamical system with evolution function $g : \mathbb{Q}^n \to \mathbb{Q}^n$. We say $\mathcal{D}$ *simulates* $\mathcal{M}$ if, given an encoding of the initial state $e[s_0] \in \mathbb{Q}^{n-2}$, an encoding of the halting state $e[s_\mathrm{h}] \in \mathbb{Q}^{n-2}$, and $\omega \in \{0,1\}^*$, $\varphi(\omega)$ is defined if and only if

$$g^k(e[s_0], \delta[\omega], 0) = (e[s_\mathrm{h}], \delta[\varphi(\omega)], \delta[\omega'])$$

for some $k \in \mathbb{N}_{>0}$ and $\omega' \in \{0,1\}^*$.

Theorem 3.1. *Let $\mathcal{M} = (S, s_0, s_\mathrm{h}, \vartheta_0, \vartheta_1, \vartheta_2)$ be a double-stack PDA with $S = \{0, 1, \ldots, s\}$ computing $\varphi : \{0,1\}^* \to \{0,1\}^*$. Then, there exists a dynamical system with evolution function $g : \mathbb{Q}^{s+2} \to \mathbb{Q}^{s+2}$ simulating $\mathcal{M}$.*

Proof. Define functions

$$\alpha_{ij} : \{0,1\}^4 \to \{0,1\}$$

for $i = 0, 1, \ldots, s$, $j = 1, 2, \ldots, s$ and

$$\beta^\ell_{ik} : \{0,1\}^4 \to \{0,1\}$$

for $k = 1, 2$, $\ell = 1, 2, 3, 4$ as follows:

$$\alpha_{ij}(a, b, d, e) = 1 \iff \vartheta_0(i, a, b, d, e) = j,$$

that is, iff, being $\mathcal{M}$ in state i and having a as top of stack one, b as top of stack two, d as the result of the nonemptiness test on stack one, and e as the result of the nonemptiness test on stack two, $\mathcal{M}$ goes to state j;

$$\beta^1_{ik}(a, b, d, e) = 1 \iff \vartheta_k(i, a, b, d, e) = (1, 0, 0),$$

that is, iff, being $\mathcal{M}$ in state i and being a, b, d, e the readings from the stacks, the content of stack k does not change (i.e. the operation `NOOPERATION` occurs on stack k);

$$\beta^2_{ik}(a, b, d, e) = 1 \iff \vartheta_k(i, a, b, d, e) = \left(\frac{1}{4}, 0, \frac{1}{4}\right),$$

that is, iff, being $\mathcal{M}$ in state i and being a, b, d, e the readings from the stacks, the operation $\texttt{PUSH}_0$ occurs on stack k;

$$\beta^3_{ik}(a, b, d, e) = 1 \iff \vartheta_k(i, a, b, d, e) = \left(\frac{1}{4}, 0, \frac{3}{4}\right),$$

that is, iff, being $\mathcal{M}$ in state i and being a, b, d, e the readings from the stacks, the operation $\texttt{PUSH}_1$ occurs on stack k;

$$\beta^4_{ik}(a, b, d, e) = 1 \iff \vartheta_k(i, a, b, d, e) = (4, -2, -1),$$

that is, iff, being $\mathcal{M}$ in state i and being a, b, d, e the readings from the stacks, the operation $\texttt{POP}$ occurs on stack k.

Consider vectors of form

$$(x_1, x_2, \dots, x_s, q_1, q_2) \in \mathbb{Q}^{s+2},$$

where $x_1, x_2, \dots, x_s \in \{0, 1\}$ and $q_1, q_2 \in Q$. The first s components are used to encode the state $\mathcal{M}$ is in, with $i \in S \setminus \{0\}$ corresponding to the i^{th} canonical vector

$$e_i := (0, \dots, 0, 1, 0, \dots, 0) \in \mathbb{Q}^s,$$

where the 1 is in the i^{th} position, and $0 \in S$ corresponding to

$$e_0 := (0, 0, \dots, 0) \in \mathbb{Q}^s;$$

q_1 and q_2, on the other hand, encode the contents of stack one and two, respectively.

Let $g : \mathbb{Q}^{s+2} \to \mathbb{Q}^{s+2}$ be defined by

$$g(x_1, x_2, \dots, x_s, q_1, q_2) := (x_1^+, x_2^+, \dots, x_s^+, q_1^+, q_2^+), \tag{3.1}$$

where, denoting $x_0 := 1 - \sum_{j=1}^{s} x_j$,

$$x_j^+ := \sum_{i=0}^{s} \alpha_{ij}(\zeta[q_1], \zeta[q_2], \tau[q_1], \tau[q_2]) x_i \tag{3.2}$$

for $j = 1, 2, \dots, s$ and

$$\begin{aligned} q_k^+ := &\left(\sum_{i=0}^{s} \beta^1_{ik}(\zeta[q_1], \zeta[q_2], \tau[q_1], \tau[q_2]) x_i\right) q_k + \\ &\left(\sum_{i=0}^{s} \beta^2_{ik}(\zeta[q_1], \zeta[q_2], \tau[q_1], \tau[q_2]) x_i\right) \left(\frac{1}{4} q_k + \frac{1}{4}\right) + \\ &\left(\sum_{i=0}^{s} \beta^3_{ik}(\zeta[q_1], \zeta[q_2], \tau[q_1], \tau[q_2]) x_i\right) \left(\frac{1}{4} q_k + \frac{3}{4}\right) + \\ &\left(\sum_{i=0}^{s} \beta^4_{ik}(\zeta[q_1], \zeta[q_2], \tau[q_1], \tau[q_2]) x_i\right) (4q_k - 2\zeta[q_k] - 1) \end{aligned} \tag{3.3}$$

for $k = 1, 2$.

Let f be the transition function of $\mathcal{M}$ and let $\pi : S \times Q \times Q \to \mathbb{Q}^{s+2}$ be defined by

$$\pi[i, q_1, q_2] := (e_i, q_1, q_2). \tag{3.4}$$

It follows immediately from equations 3.2 and 3.3 that, for any $(i, q_1, q_2) \in S \times Q \times Q$,

$$g(\pi[i, q_1, q_2]) = \pi[f(i, q_1, q_2)],$$

which, applied repeatedly, becomes

$$g^k(\pi[i, q_1, q_2]) = \pi[f^k(i, q_1, q_2)] \tag{3.5}$$

for all $k \in \mathbb{N}_{>0}$.

Assume, without loss of generality, that $s_0 = 0$ and $s_\mathrm{h} = 1$, and let $\omega \in \{0,1\}^*$. Then, $\varphi(\omega)$ is defined if and only if

$$\begin{aligned}
g^\ell(e_0, \delta[\omega], 0) &= g^\ell(\pi[0, \delta[\omega], 0]) && \text{(by equation 3.4)}\\
&= \pi[f^\ell(0, \delta[\omega], 0)] && \text{(by equation 3.5)}\\
&= \pi[1, \delta[\varphi(\omega)], \delta[\omega']] && \text{(by definition 1.6)}\\
&= (e_1, \delta[\varphi(\omega)], \delta[\omega']) && \text{(by equation 3.4)}
\end{aligned}$$

for some $\ell \in \mathbb{N}_{>0}$ and $\omega' \in \{0,1\}^*$, i.e. g is the evolution function of a dynamical system which simulates $\mathcal{M}$. □

Now that we have seen how to simulate a double-stack PDA with a generic dynamical system with evolution function $g : \mathbb{Q}^{s+2} \to \mathbb{Q}^{s+2}$, we will show how to implement g with a neural network. We first need the following technical lemma:

Lemma 3.2. *Let $\alpha : \{0,1\}^4 \to \{0,1\}$, $a, b, d, e, x \in \{0,1\}$, and $q \in [0,1)$. Then, there exist vectors*

$$v_1, v_2, \ldots, v_{16} \in \mathbb{Z}^6$$

and scalars

$$c_1, c_2, \ldots, c_{16} \in \{0,1\}$$

such that

$$\alpha(a,b,d,e)x = \sum_{m=1}^{16} c_m \sigma(v_m \cdot \eta) \tag{3.6}$$

and

$$\alpha(a,b,d,e)xq = \sigma\left(q - 1 + \sum_{m=1}^{16} c_m \sigma(v_m \cdot \eta)\right),$$

where $\eta := (1, a, b, d, e, x)$ and $\cdot$ is the dot product in $\mathbb{Z}^6$.

Proof. The fact that α goes from $\{0,1\}^4$ to $\{0,1\}$ allows us to write it as a polynomial (see table 3.1):

$$\begin{aligned}
\alpha(a,b,d,e) = c_1 &+ c_2 e + c_3 d + c_4 de +\\
&c_5 b + c_6 be + c_7 bd + c_8 bde +\\
&c_9 a + c_{10} ae + c_{11} ad + c_{12} ade +\\
&c_{13} ab + c_{14} abe + c_{15} abd + c_{16} abde
\end{aligned}$$

a	b	d	e	Term in the polynomial
0	0	0	0	c_1
0	0	0	1	c_2e
0	0	1	0	c_3d
0	0	1	1	c_4de
0	1	0	0	c_5b
0	1	0	1	c_6be
0	1	1	0	c_7bd
0	1	1	1	c_8bde
1	0	0	0	c_9a
1	0	0	1	$c_{10}ae$
1	0	1	0	$c_{11}ad$
1	0	1	1	$c_{12}ade$
1	1	0	0	$c_{13}ab$
1	1	0	1	$c_{14}abe$
1	1	1	0	$c_{15}abd$
1	1	1	1	$c_{16}abde$

Table 3.1: $\alpha : \{0,1\}^4 \to \{0,1\}$ as a polynomial

for some $c_1, c_2, \ldots, c_{16} \in \{0,1\}$, and thus

$$\begin{aligned}\alpha(a,b,d,e)x = {} & c_1x + c_2ex + c_3dx + c_4dex + \\ & c_5bx + c_6bex + c_7bdx + c_8bdex + \\ & c_9ax + c_{10}aex + c_{11}adx + c_{12}adex + \\ & c_{13}abx + c_{14}abex + c_{15}abdx + c_{16}abdex.\end{aligned} \tag{3.7}$$

Note that, for any $a_1, a_2, \ldots, a_k \in \{0,1\}$ with $k \in \mathbb{N}_{>0}$ and $p \in [0,1]$, one has

$$a_1a_2\ldots a_kp = \sigma(a_1 + a_2 + \cdots + a_k - k + p). \tag{3.8}$$

In fact, if there exists $i \in \{1,2,\ldots,k\}$ such that $a_i = 0$, then $a_1 + a_2 + \cdots + a_k \leq k - 1$, and thus $a_1 + a_2 + \cdots + a_k - k + p \leq k - 1 - k + p = p - 1 \in [-1,0]$, hence $\sigma(a_1 + a_2 + \cdots + a_k - k + p) = 0 = a_1a_2\ldots a_kp$; on the other hand, if $a_i = 1$ for all $i \in \{1,2,\ldots,k\}$, then $a_1 + a_2 + \cdots + a_k = k$, and thus $a_1 + a_2 + \cdots + a_k - k + p = k - k + p = p \in [0,1]$, hence $\sigma(a_1 + a_2 + \cdots + a_k - k + p) = p = a_1a_2\ldots a_kp$.

Therefore, we can rewrite the terms of equation 3.7 as (consider $p = 1$)

$$\begin{aligned}c_1x &= c_1\sigma(x) \\ &= c_1\sigma((0,0,0,0,0,1)\cdot(1,a,b,d,e,x)) \\ c_2ex &= c_2\sigma(e + x - 1) \\ &= c_2\sigma((-1,0,0,0,1,1)\cdot(1,a,b,d,e,x)) \\ c_3dx &= c_3\sigma(d + x - 1)\end{aligned}$$

$$\begin{aligned}
&= c_3\sigma((-1,0,0,1,0,1)\cdot(1,a,b,d,e,x))\\
c_4dex &= c_4\sigma(d+e+x-2)\\
&= c_4\sigma((-2,0,0,1,1,1)\cdot(1,a,b,d,e,x))\\
c_5bx &= c_5\sigma(b+x-1)\\
&= c_5\sigma((-1,0,1,0,0,1)\cdot(1,a,b,d,e,x))\\
c_6bex &= c_6\sigma(b+e+x-2)\\
&= c_6\sigma((-2,0,1,0,1,1)\cdot(1,a,b,d,e,x))\\
c_7bdx &= c_7\sigma(b+d+x-2)\\
&= c_7\sigma((-2,0,1,1,0,1)\cdot(1,a,b,d,e,x))\\
c_8bdex &= c_8\sigma(b+d+e+x-3)\\
&= c_8\sigma((-3,0,1,1,1,1)\cdot(1,a,b,d,e,x))\\
c_9ax &= c_9\sigma(a+x-1)\\
&= c_9\sigma((-1,1,0,0,0,1)\cdot(1,a,b,d,e,x))\\
c_{10}aex &= c_{10}\sigma(a+e+x-2)\\
&= c_{10}\sigma((-2,1,0,0,1,1)\cdot(1,a,b,d,e,x))\\
c_{11}adx &= c_{11}\sigma(a+d+x-2)\\
&= c_{11}\sigma((-2,1,0,1,0,1)\cdot(1,a,b,d,e,x))\\
c_{12}adex &= c_{12}\sigma(a+d+e+x-3)\\
&= c_{12}\sigma((-3,1,0,1,1,1)\cdot(1,a,b,d,e,x))\\
c_{13}abx &= c_{13}\sigma(a+b+x-2)\\
&= c_{13}\sigma((-2,1,1,0,0,1)\cdot(1,a,b,d,e,x))\\
c_{14}abex &= c_{14}\sigma(a+b+e+x-3)\\
&= c_{14}\sigma((-3,1,1,0,1,1)\cdot(1,a,b,d,e,x))\\
c_{15}abdx &= c_{15}\sigma(a+b+d+x-3)\\
&= c_{15}\sigma((-3,1,1,1,0,1)\cdot(1,a,b,d,e,x))\\
c_{16}abdex &= c_{16}\sigma(a+b+d+e+x-4)\\
&= c_{16}\sigma((-4,1,1,1,1,1)\cdot(1,a,b,d,e,x)),
\end{aligned}$$

hence

$$\alpha(a,b,d,e)x = \sum_{m=1}^{16} c_m\sigma(v_m\cdot\eta)$$

for some $c_1, c_2, \ldots, c_{16} \in \{0,1\}$ and $v_1, v_2, \ldots, v_{16} \in \mathbb{Z}^6$.

Note also that, for any $c \in \{0,1\}$ and $r \in [0,1)$, it holds that

$$cr = \sigma(r-1+c). \tag{3.9}$$

In fact, if $c=0$, then $r-1+c = r-1 \in [-1,0)$, hence $\sigma(r-1+c) = 0 = cr$; on the other hand, if $c=1$, then $r-1+c = r \in [0,1)$, hence $\sigma(r-1+c) = r = cr$.

Therefore, by equations 3.6 and 3.9,

$$\alpha(a,b,d,e)xq = \sigma\left(q-1+\sum_{m=1}^{16} c_m\sigma(v_m\cdot\eta)\right)$$

for some $c_1, c_2, \ldots, c_{16} \in \{0,1\}$ and $v_1, v_2, \ldots, v_{16} \in \mathbb{Z}^6$. □

a	b	d	e	Term in the polynomial	v_m
0	0	0	0	c_1	$(0,0,0,0,0,1)$
0	0	0	1	c_2e	$(-1,0,0,0,1,1)$
0	0	1	0	c_3d	$(-1,0,0,1,0,1)$
0	0	1	1	c_4de	$(-2,0,0,1,1,1)$
0	1	0	0	0	
0	1	0	1	c_5be	$(-2,0,1,0,1,1)$
0	1	1	0	0	
0	1	1	1	c_6bde	$(-3,0,1,1,1,1)$
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	c_7ad	$(-2,1,0,1,0,1)$
1	0	1	1	c_8ade	$(-3,1,0,1,1,1)$
1	1	0	0	0	
1	1	0	1	0	
1	1	1	0	0	
1	1	1	1	c_9abde	$(-4,1,1,1,1,1)$

Table 3.2: α_{ij}s and β^{ℓ}_{ik}s as polynomials

Remark 3.2.1. Note that, if the α from lemma 3.2 stands for the α_{ij}s or the β^{ℓ}_{ik}s from theorem 3.1, then $a = \zeta[q_1], b = \zeta[q_2], d = \tau[q_1], e = \tau[q_2]$, and since, as we already pointed out in definition 1.2, it can never happen that $(\zeta[q_1] = 1 \wedge \tau[q_1] = 0) \vee (\zeta[q_2] = 1 \wedge \tau[q_2] = 0)$, i.e. $(a = 1 \wedge d = 0) \vee (b = 1 \wedge e = 0)$, we just need nine out of the sixteen $c_1, c_2, \ldots, c_{16}$ to write α as a polynomial (see table 3.2):

$$\begin{aligned}\alpha(a,b,d,e) = c_1 + c_2e + c_3d + c_4de + c_5be +{}& \\ c_6bde + c_7ad + c_8ade + c_9abde,&\end{aligned}$$

hence

$$\alpha(a,b,d,e)x = \sum_{m=1}^{9} c_m\sigma(v_m \cdot \eta) \tag{3.10}$$

and

$$\alpha(a,b,d,e)xq = \sigma\left(q - 1 + \sum_{m=1}^{9} c_m\sigma(v_m \cdot \eta)\right).$$

Corollary 3.1.1. *Let $\mathcal{M} = (S, s_0, s_{\mathrm{h}}, \vartheta_0, \vartheta_1, \vartheta_2)$ be a double-stack PDA with $S = \{0,1,\ldots,s\}$ computing $\varphi : \{0,1\}^* \to \{0,1\}^*$. Then, there exists a neural network with evolution function $g : \mathbb{Q}^{s+2} \to \mathbb{Q}^{s+2}$ simulating $\mathcal{M}$.*

Proof. Apply equation 3.10 to equation 3.2, where $\alpha = \alpha_{ij}$ and $x = x_i$. Furthermore, write $\sigma(4q_k - 2)$ whenever $\zeta[q_k]$ appears (see equation 1.6) and $\sigma(4q_k)$ whenever $\tau[q_k]$ appears (see equation 1.7). Finally, note that $x_j^+ = \sigma\bigl(x_j^+\bigr)$, since $x_j^+ \in \{0,1\}$. This way, equation 3.2 becomes

$$x_j^+ = \sigma\left(\sum_{i=0}^{s}\sum_{m=1}^{9} c_{ijm}\sigma_{im}\right), \tag{3.11}$$

where

$$\sigma_{im} := \sigma(v_m \cdot (1, \sigma(4q_1-2), \sigma(4q_2-2), \sigma(4q_1), \sigma(4q_2), x_i)) \tag{3.12}$$

(the v_ms are shown in table 3.2).

On the other hand, apply equation 3.9 to equation 3.3, where $c = \sum_{i=0}^{s} \beta_{ik}^{\ell}(\dots)x_i$ and $r = q_k$, $r = \frac{1}{4}q_k + \frac{1}{4}$, $r = \frac{1}{4}q_k + \frac{3}{4}$, or $r = 4q_k - 2\zeta[q_k] - 1$. Again, write $\sigma(4q_k-2)$ whenever $\zeta[q_k]$ appears and $\sigma(4q_k)$ whenever $\tau[q_k]$ appears. Finally, note that $q_k^+ = \sigma\bigl(q_k^+\bigr)$, since $q_k \in [0,1)$. This way, equation 3.3 becomes

$$\begin{aligned} q_k^+ = \sigma\Biggl(&\sigma\left(q_k - 1 + \sum_{i=0}^{s}\beta_{ik}^{1}(\sigma(4q_1-2), \sigma(4q_2-2), \sigma(4q_1), \sigma(4q_2))x_i\right) + \\ &\sigma\left(\frac{1}{4}q_k + \frac{1}{4} - 1 + \sum_{i=0}^{s}\beta_{ik}^{2}(\sigma(4q_1-2), \sigma(4q_2-2), \sigma(4q_1), \sigma(4q_2))x_i\right) + \\ &\sigma\left(\frac{1}{4}q_k + \frac{3}{4} - 1 + \sum_{i=0}^{s}\beta_{ik}^{3}(\sigma(4q_1-2), \sigma(4q_2-2), \sigma(4q_1), \sigma(4q_2))x_i\right) + \\ &\sigma\left(4q_k - 2\sigma(4q_k-2) - 1 - 1\ + \sum_{i=0}^{s}\beta_{ik}^{4}(\sigma(4q_1-2), \sigma(4q_2-2), \sigma(4q_1), \sigma(4q_2))x_i\right)\Biggr). \end{aligned}$$

Now, if we apply equation 3.10 with $x = x_i$ and $\alpha = \beta_{ik}^1$, $\alpha = \beta_{ik}^2$, $\alpha = \beta_{ik}^3$, or $\alpha = \beta_{ik}^4$, we get

$$q_k^+ = \sigma\bigl(q_{k1}^+ + q_{k2}^+ + q_{k3}^+ + q_{k4}^+\bigr), \tag{3.13}$$

where

$$\begin{aligned} q_{k1}^+ &:= \sigma\left(q_k - 1 + \sum_{i=0}^{s}\sum_{m=1}^{9} c_{ikm}^{1}\sigma_{im}\right) \\ q_{k2}^+ &:= \sigma\left(\frac{1}{4}q_k - \frac{3}{4} + \sum_{i=0}^{s}\sum_{m=1}^{9} c_{ikm}^{2}\sigma_{im}\right) \\ q_{k3}^+ &:= \sigma\left(\frac{1}{4}q_k - \frac{1}{4} + \sum_{i=0}^{s}\sum_{m=1}^{9} c_{ikm}^{3}\sigma_{im}\right) \\ q_{k4}^+ &:= \sigma\left(4q_k - 2\sigma(4q_k-2) - 2 + \sum_{i=0}^{s}\sum_{m=1}^{9} c_{ikm}^{4}\sigma_{im}\right). \end{aligned} \tag{3.14}$$

Write g as a composition

$$g = g_4 \circ g_3 \circ g_2 \circ g_1$$

of four functions of form $\sigma(Wx+b)$:

$$\begin{aligned} g_1 &: \mathbb{Q}^{s+2} \to \mathbb{Q}^{s+7} \\ g_2 &: \mathbb{Q}^{s+7} \to \mathbb{Q}^{9s+13} \\ g_3 &: \mathbb{Q}^{9s+13} \to \mathbb{Q}^{s+8} \\ g_4 &: \mathbb{Q}^{s+8} \to \mathbb{Q}^{s+2}. \end{aligned}$$

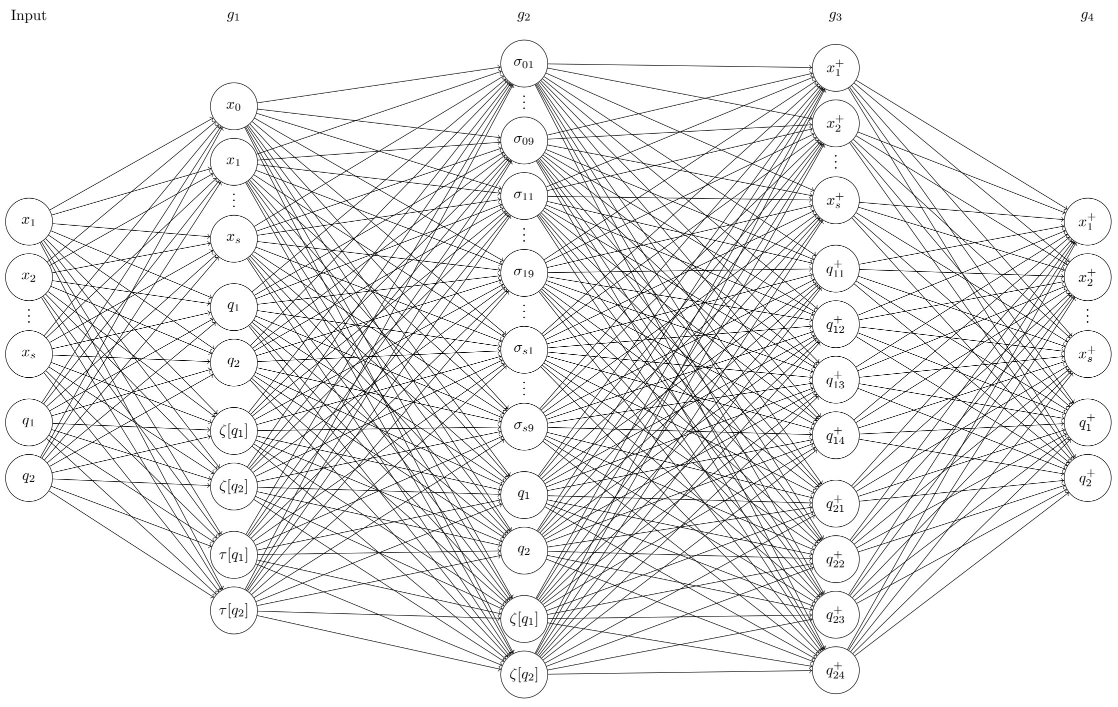

Figure 3.1: a recurrent neural network simulating a generic double-stack PDA

Each of those functions computes a piece of equations 3.11 and 3.13 and corresponds to a layer of the neural network $\mathcal{N}$ shown in figure 3.1:

- g_1:
  - Input: the same as g (see equation 3.1), i.e. $x_1, x_2, \dots, x_s, q_1, q_2$;
  - Computes: $x_0 = 1 - \sum_{j=1}^{s} x_j$ (as defined in theorem 3.1), $\zeta[q_k] = \sigma(4q_k - 2)$, and $\tau[q_k] = \sigma(4q_k)$, for $k = 1, 2$;
  - Output: $x_0, x_1, \dots, x_s, q_1, q_2, \zeta[q_k], \tau[q_k]$, for $k = 1, 2$;
- g_2:
  - Input: $x_0, x_1, \dots, x_s, q_1, q_2, \zeta[q_k], \tau[q_k]$, for $k = 1, 2$;
  - Computes: the σ_{im}s of equation 3.12, for $i = 0, 1 \dots, s$ and $m = 1, 2, \dots, 9$;
  - Output: $\sigma_{im}, q_k, \zeta[q_k]$, for $i = 0, 1 \dots, s$, $m = 1, 2, \dots, 9$, and $k = 1, 2$;
- g_3:
  - Input: $\sigma_{im}, q_k, \zeta[q_k]$, for $i = 0, 1 \dots, s$, $m = 1, 2, \dots, 9$, and $k = 1, 2$;
  - Computes: the x_j^+s of equation 3.11 and the $q_{k\ell}^+$s of equations 3.14, for $j = 1, 2, \dots, s$, $k = 1, 2$, and $\ell = 1, 2, 3, 4$;
  - Output: $x_j^+, q_{k\ell}^+$, for $j = 1, 2, \dots, s$, $k = 1, 2$, and $\ell = 1, 2, 3, 4$;
- g_4:

- Input: $x_j^+, q_{k\ell}^+$, for $j = 1, 2, \ldots, s$, $k = 1, 2$, and $\ell = 1, 2, 3, 4$;
- Computes: the q_k^+s of equation 3.13, for $k = 1, 2$;
- Output: x_j^+, q_k^+, for $j = 1, 2, \ldots, s$ and $k = 1, 2$.

The output of g_4 becomes then the input of g_1 for the next iteration of g; $\mathcal{N}$ is therefore referred to as a *recurrent* neural network. The weight matrices and bias vectors of $\mathcal{N}$ are shown in tables 3.3, 3.4, 3.5, and 3.6.

We have already proved in theorem 3.1 that g is the evolution function of a dynamical system $\mathcal{D}$ which simulates $\mathcal{M}$, assuming, without loss of generality, that $s_0 = 0$ and $s_\mathrm{h} = 1$ (encoded in $\mathcal{D}$ by e_0 and e_1, respectively). On the other hand, we have just built a neural network $\mathcal{N}$ implementing g. Thus, given $\omega \in \{0, 1\}^*$, $\varphi(\omega)$ is defined if and only if an input layer $x_1 = x_2 = \cdots = x_s = 0$ (i.e. e_0), $q_1 = \delta[\omega], q_2 = 0$ eventually causes $\mathcal{N}$ to output in the g_4 layer $x_1^+ = 1, x_2^+ = x_3^+ = \cdots = x_s^+ = 0$ (i.e. e_1), $q_1^+ = \delta[\varphi(\omega)]$ (and some $q_2^+ \in Q$). $\square$

W^1	x_1	x_2	$\cdots$	x_s	q_1	q_2	b^1
x_0	-1	-1	$\cdots$	-1	0	0	1
x_1	1	0	$\cdots$	0	0	0	0
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_s	0	0	$\cdots$	1	0	0	0
q_1	0	0	$\cdots$	0	1	0	0
q_2	0	0	$\cdots$	0	0	1	0
$\zeta[q_1]$	0	0	$\cdots$	0	4	0	-2
$\zeta[q_2]$	0	0	$\cdots$	0	0	4	-2
$\tau[q_1]$	0	0	$\cdots$	0	4	0	0
$\tau[q_2]$	0	0	$\cdots$	0	0	4	0

Table 3.3: weights from the input layer to the g_1 layer and biases of the g_1 layer

W^2	x_0	x_1	$\cdots$	x_s	q_1	q_2	$\zeta[q_1]$	$\zeta[q_2]$	$\tau[q_1]$	$\tau[q_2]$	b^2
σ_{01}	1	0	$\cdots$	0	0	0	0	0	0	0	0
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
σ_{09}	1	0	$\cdots$	0	0	0	1	1	1	1	-4
σ_{11}	0	1	$\cdots$	0	0	0	0	0	0	0	0
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
σ_{19}	0	1	$\cdots$	0	0	0	1	1	1	1	-4
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
σ_{s1}	0	0	$\cdots$	1	0	0	0	0	0	0	0
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
σ_{s9}	0	0	$\cdots$	1	0	0	1	1	1	1	-4
q_1	0	0	$\cdots$	0	1	0	0	0	0	0	0
q_2	0	0	$\cdots$	0	0	1	0	0	0	0	0
$\zeta[q_1]$	0	0	$\cdots$	0	0	0	1	0	0	0	0
$\zeta[q_2]$	0	0	$\cdots$	0	0	0	0	1	0	0	0

Table 3.4: weights from the g_1 layer to the g_2 layer and biases of the g_2 layer

W^3	σ_{01}	$\cdots$	σ_{09}	σ_{11}	$\cdots$	σ_{19}	$\cdots$	σ_{s1}	$\cdots$	σ_{s9}	q_1	q_2	$\zeta[q_1]$	$\zeta[q_2]$	b^3
x_1^+	c_{011}	$\cdots$	c_{019}	c_{111}	$\cdots$	c_{119}	$\cdots$	c_{s11}	$\cdots$	c_{s19}	0	0	0	0	0
x_2^+	c_{021}	$\cdots$	c_{029}	c_{121}	$\cdots$	c_{129}	$\cdots$	c_{s21}	$\cdots$	c_{s29}	0	0	0	0	0
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_s^+	c_{0s1}	$\cdots$	c_{0s9}	c_{1s1}	$\cdots$	c_{1s9}	$\cdots$	c_{ss1}	$\cdots$	c_{ss9}	0	0	0	0	0
q_{11}^+	c_{011}^1	$\cdots$	c_{019}^1	c_{111}^1	$\cdots$	c_{119}^1	$\cdots$	c_{s11}^1	$\cdots$	c_{s19}^1	1	0	0	0	-1
q_{12}^+	c_{011}^2	$\cdots$	c_{019}^2	c_{111}^2	$\cdots$	c_{119}^2	$\cdots$	c_{s11}^2	$\cdots$	c_{s19}^2	$\frac{1}{4}$	0	0	0	$-\frac{3}{4}$
q_{13}^+	c_{011}^3	$\cdots$	c_{019}^3	c_{111}^3	$\cdots$	c_{119}^3	$\cdots$	c_{s11}^3	$\cdots$	c_{s19}^3	$\frac{1}{4}$	0	0	0	$-\frac{1}{4}$
q_{14}^+	c_{011}^4	$\cdots$	c_{019}^4	c_{111}^4	$\cdots$	c_{119}^4	$\cdots$	c_{s11}^4	$\cdots$	c_{s19}^4	4	0	-2	0	-2
q_{21}^+	c_{021}^1	$\cdots$	c_{029}^1	c_{121}^1	$\cdots$	c_{129}^1	$\cdots$	c_{s21}^1	$\cdots$	c_{s29}^1	0	1	0	0	-1
q_{22}^+	c_{021}^2	$\cdots$	c_{029}^2	c_{121}^2	$\cdots$	c_{129}^2	$\cdots$	c_{s21}^2	$\cdots$	c_{s29}^2	0	$\frac{1}{4}$	0	0	$-\frac{3}{4}$
q_{23}^+	c_{021}^3	$\cdots$	c_{029}^3	c_{121}^3	$\cdots$	c_{129}^3	$\cdots$	c_{s21}^3	$\cdots$	c_{s29}^3	0	$\frac{1}{4}$	0	0	$-\frac{1}{4}$
q_{24}^+	c_{021}^4	$\cdots$	c_{029}^4	c_{121}^4	$\cdots$	c_{129}^4	$\cdots$	c_{s21}^4	$\cdots$	c_{s29}^4	0	4	0	-2	-2

Table 3.5: weights from the g_2 layer to the g_3 layer and biases of the g_3 layer

W^4	x_1^+	x_2^+	$\cdots$	x_s^+	q_{11}^+	q_{12}^+	q_{13}^+	q_{14}^+	q_{21}^+	q_{22}^+	q_{23}^+	q_{24}^+	b^4
x_1^+	1	0	$\cdots$	0	0	0	0	0	0	0	0	0	0
x_2^+	0	1	$\cdots$	0	0	0	0	0	0	0	0	0	0
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_s^+	0	0	$\cdots$	1	0	0	0	0	0	0	0	0	0
q_1^+	0	0	$\cdots$	0	1	1	1	1	0	0	0	0	0
q_2^+	0	0	$\cdots$	0	0	0	0	0	1	1	1	1	0

Table 3.6: weights from the g_3 layer to the g_4 layer and biases of the g_4 layer

Chapter 4

An example

4.1 Formal construction

We are now going to apply the formal construction of chapter 3 to a specific example to get a more compact and concrete overview of how it actually operates. In the following discussion, mostly based on [3], we will not precisely follow such construction step by step, though, nor will we make use of every single piece of notation we introduced, as the low complexity of the example will allow several simplifications, which will come natural as our calculation unfolds.

Consider a double-stack PDA $\mathcal{M}$ which computes unary addition starting with the first stack containing the two unary numbers to add separated by a zero (see figure 4.1 and table 4.1). $\mathcal{M}$ is uniquely described at time $t \geq 0$ by the triple (s, q_1, q_2), where s is the state $\mathcal{M}$ is in and q_1, q_2 are the encodings of its first and second stack, respectively. We can equivalently express such triple as

$$(x_0, x_1, x_{\mathrm{h}}, q_1, q_2) \in \mathbb{Q}^5,$$

where, for each $i \in \{0, 1, \mathrm{h}\}$,

$$x_i := \begin{cases} 1 & \text{if } \mathcal{M} \text{ is in state } s_i \\ 0 & \text{otherwise,} \end{cases}$$

and we let

$$\left(x_0^+, x_1^+, x_{\mathrm{h}}^+, q_1^+, q_2^+\right) \in \mathbb{Q}^5$$

denote that same five-tuple at time $t+1$, where

$$\begin{aligned}
x_0^+ &= \sigma\left(x_0^+\right) && (\text{since } x_0^+ \in \{0,1\}) \\
&= \sigma(x_0 \cdot \texttt{NONEMPTY}(q_1) \cdot \texttt{TOP}(q_1) + \\
&\qquad x_0 \cdot \texttt{NONEMPTY}(q_1) \cdot \neg\texttt{TOP}(q_1)) \\
&= \sigma(\sigma(x_0 + \texttt{NONEMPTY}(q_1) + \texttt{TOP}(q_1) - 2) + \\
&\qquad \sigma(x_0 + \texttt{NONEMPTY}(q_1) + \neg\texttt{TOP}(q_1) - 2)) && (\text{by equation 3.8}) \\
&= \sigma(\sigma(x_0 + \sigma(4q_1) + \sigma(4q_1 - 2) - 2) + \\
&\qquad \sigma(x_0 + \sigma(4q_1) + 1 - \sigma(q_1 - 2) - 2)) && (\text{by equations 1.6 and 1.7}) \\
&= \sigma(\underbrace{\sigma(x_0 + \sigma(4q_1) + \sigma(4q_1 - 2) - 2)}_{=:\,\sigma_1} + \\
&\qquad \underbrace{\sigma(x_0 + \sigma(4q_1) - \sigma(4q_1 - 2) - 1)}_{=:\,\sigma_2})
\end{aligned}$$

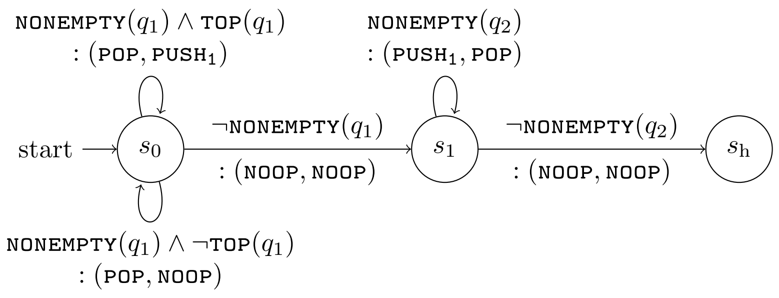
$\texttt{NONEMPTY}(q_1) \wedge \texttt{TOP}(q_1)$
$: (\texttt{POP}, \texttt{PUSH}_1)$
$\texttt{NONEMPTY}(q_2)$
$: (\texttt{PUSH}_1, \texttt{POP})$
start
s_0
$\neg\texttt{NONEMPTY}(q_1)$
$: (\texttt{NOOP}, \texttt{NOOP})$
s_1
$\neg\texttt{NONEMPTY}(q_2)$
$: (\texttt{NOOP}, \texttt{NOOP})$
s_h
$\texttt{NONEMPTY}(q_1) \wedge \neg\texttt{TOP}(q_1)$
$: (\texttt{POP}, \texttt{NOOP})$

Figure 4.1: a double-stack PDA computing unary addition

Time	State	Stack 1	Stack 2
0	s_0	101	ε
1	s_0	01	1
2	s_0	1	1
3	s_0	ε	11
4	s_1	ε	11
5	s_1	1	1
6	s_1	11	ε
7	s_h	11	ε

Table 4.1: execution of the double-stack PDA in figure 4.1 with input 101

$$\begin{aligned}
x_1^+ &= \sigma\big(x_1^+\big) && \text{(since } x_1^+ \in \{0,1\}\text{)}\\
&= \sigma(x_0 \cdot \neg\texttt{NONEMPTY}(q_1) + \\
&\qquad x_1 \cdot \texttt{NONEMPTY}(q_2))\\
&= \sigma(\sigma(x_0 + \neg\texttt{NONEMPTY}(q_1) - 1) + \\
&\qquad \sigma(x_1 + \texttt{NONEMPTY}(q_2) - 1)) && \text{(by equation 3.8)}\\
&= \sigma(\sigma(x_0 + 1 - \sigma(4q_1) - 1) + \\
&\qquad \sigma(x_1 + \sigma(4q_2) - 1)) && \text{(by equation 1.7)}\\
&= \sigma(\underbrace{\sigma(x_0 - \sigma(4q_1))}_{=:\,\sigma_3} + \\
&\qquad \underbrace{\sigma(x_1 + \sigma(4q_2) - 1)}_{=:\,\sigma_4})
\end{aligned}$$

$$\begin{aligned}
x_\text{h}^+ &= \sigma\big(x_\text{h}^+\big) && \text{(since } x_\text{h}^+ \in \{0,1\}\text{)}\\
&= \sigma(x_1 \cdot \neg\texttt{NONEMPTY}(q_2))\\
&= \sigma(\sigma(x_1 + \neg\texttt{NONEMPTY}(q_2) - 1)) && \text{(by equation 3.8)}\\
&= \sigma(\sigma(x_1 + 1 - \sigma(4q_2) - 1)) && \text{(by equation 1.7)}
\end{aligned}$$

$$= \sigma(\underbrace{\sigma(x_1 - \sigma(4q_2))}_{=:\,\sigma_5})$$

$$\begin{aligned}
q_1^+ &= \sigma\big(q_1^+\big) && \text{(since } q_1^+ \in [0,1)\text{)} \\
&= \sigma(x_0 \cdot \texttt{NONEMPTY}(q_1) \cdot \texttt{TOP}(q_1) \cdot \texttt{POP}(q_1) + \\
&\qquad x_0 \cdot \texttt{NONEMPTY}(q_1) \cdot \neg\texttt{TOP}(q_1) \cdot \texttt{POP}(q_1) + \\
&\qquad x_0 \cdot \neg\texttt{NONEMPTY}(q_1) \cdot \texttt{NOOPERATION}(q_1) + \\
&\qquad x_1 \cdot \texttt{NONEMPTY}(q_2) \cdot \texttt{PUSH}_\texttt{1}(q_1) + \\
&\qquad x_1 \cdot \neg\texttt{NONEMPTY}(q_2) \cdot \texttt{NOOPERATION}(q_1)) \\
&= \sigma(\sigma(x_0 + \texttt{NONEMPTY}(q_1) + \texttt{TOP}(q_1) - 3 + \texttt{POP}(q_1)) + \\
&\qquad \sigma(x_0 + \texttt{NONEMPTY}(q_1) + \neg\texttt{TOP}(q_1) - 3 + \texttt{POP}(q_1)) + \\
&\qquad \sigma(x_0 + \neg\texttt{NONEMPTY}(q_1) - 2 + \texttt{NOOPERATION}(q_1)) + \\
&\qquad \sigma(x_1 + \texttt{NONEMPTY}(q_2) - 2 + \texttt{PUSH}_\texttt{1}(q_1)) + \\
&\qquad \sigma(x_1 + \neg\texttt{NONEMPTY}(q_2) - 2 + \texttt{NOOPERATION}(q_1))) && \text{(by equation 3.8)} \\
&= \sigma\big(\sigma(x_0 + \sigma(4q_1) + \sigma(4q_1 - 2) - 3 + 4q_1 - 2\sigma(4q_1 - 2) - 1) + \\
&\qquad \sigma(x_0 + \sigma(4q_1) + 1 - \sigma(4q_1 - 2) - 3 + 4q_1 - 2\sigma(4q_1 - 2) - 1) + \\
&\qquad \sigma(x_0 + 1 - \sigma(4q_1) - 2 + q_1) + \\
&\qquad \sigma\big(x_1 + \sigma(4q_2) - 2 + \tfrac{1}{4}q_1 + \tfrac{3}{4}\big) + \\
&\qquad \sigma(x_1 + 1 - \sigma(4q_2) - 2 + q_1)\big) && \text{(by equations 1.6, 1.7, 1.9, and 1.10)} \\
&= \sigma\big(\underbrace{\sigma(x_0 + \sigma(4q_1) - \sigma(4q_1 - 2) + 4q_1 - 4)}_{=:\,\sigma_6} + \\
&\qquad \underbrace{\sigma(x_0 + \sigma(4q_1) - 3\sigma(4q_1 - 2) + 4q_1 - 3)}_{=:\,\sigma_7} + \\
&\qquad \underbrace{\sigma(x_0 - \sigma(4q_1) + q_1 - 1)}_{=:\,\sigma_8} + \\
&\qquad \underbrace{\sigma\big(x_1 + \sigma(4q_2) + \tfrac{1}{4}q_1 - \tfrac{5}{4}\big)}_{=:\,\sigma_9} + \\
&\qquad \underbrace{\sigma(x_1 - \sigma(4q_2) + q_1 - 1)}_{=:\,\sigma_{10}}\big)
\end{aligned}$$

$$\begin{aligned}
q_2^+ &= \sigma\big(q_2^+\big) && \text{(since } q_2^+ \in [0,1)\text{)} \\
&= \sigma(x_0 \cdot \texttt{NONEMPTY}(q_1) \cdot \texttt{TOP}(q_1) \cdot \texttt{PUSH}_\texttt{1}(q_2) + \\
&\qquad x_0 \cdot \texttt{NONEMPTY}(q_1) \cdot \neg\texttt{TOP}(q_1) \cdot \texttt{NOOPERATION}(q_2) + \\
&\qquad x_0 \cdot \neg\texttt{NONEMPTY}(q_1) \cdot \texttt{NOOPERATION}(q_2) + \\
&\qquad x_1 \cdot \texttt{NONEMPTY}(q_2) \cdot \texttt{POP}(q_2) + \\
&\qquad x_1 \cdot \neg\texttt{NONEMPTY}(q_2) \cdot \texttt{NOOPERATION}(q_2)) \\
&= \sigma(\sigma(x_0 + \texttt{NONEMPTY}(q_1) + \texttt{TOP}(q_1) - 3 + \texttt{PUSH}_\texttt{1}(q_2)) + \\
&\qquad \sigma(x_0 + \texttt{NONEMPTY}(q_1) + \neg\texttt{TOP}(q_1) - 3 + \texttt{NOOPERATION}(q_2)) + \\
&\qquad \sigma(x_0 + \neg\texttt{NONEMPTY}(q_1) - 2 + \texttt{NOOPERATION}(q_2)) +
\end{aligned}$$

$$
\begin{aligned}
&\sigma(x_1 + \texttt{NONEMPTY}(q_2) - 2 + \texttt{POP}(q_2)) + \\
&\sigma(x_1 + \neg\texttt{NONEMPTY}(q_2) - 2 + \texttt{NOOPERATION}(q_2))) \qquad \text{(by equation 3.8)} \\
= \sigma\big(&\sigma\big(x_0 + \sigma(4q_1) + \sigma(4q_1 - 2) - 3 + \tfrac{1}{4}q_2 + \tfrac{3}{4}\big) + \\
&\sigma(x_0 + \sigma(4q_1) + 1 - \sigma(4q_1 - 2) - 3 + q_2) + \\
&\sigma(x_0 + 1 - \sigma(4q_1) - 2 + q_2) + \\
&\sigma(x_1 + \sigma(4q_2) - 2 + 4q_2 - 2\sigma(4q_2 - 2) - 1) + \\
&\sigma(x_1 + 1 - \sigma(4q_2) - 2 + q_2)\big) \qquad \text{(by equations 1.6, 1.7, 1.9, and 1.10)} \\
= \sigma\big(&\underbrace{\sigma\big(x_0 + \sigma(4q_1) - \sigma(4q_1 - 2) + \tfrac{1}{4}q_2 - \tfrac{9}{4}\big)}_{=:\,\sigma_{11}} + \\
&\underbrace{\sigma(x_0 + \sigma(4q_1) - \sigma(4q_1 - 2) + q_2 - 2)}_{=:\,\sigma_{12}} + \\
&\underbrace{\sigma(x_0 - \sigma(4q_1) + q_2 - 1)}_{=:\,\sigma_{13}} + \\
&\underbrace{\sigma(x_1 + \sigma(4q_2) - 2\sigma(4q_2 - 2) + 4q_2 - 3)}_{=:\,\sigma_{14}} + \\
&\underbrace{\sigma(x_1 - \sigma(4q_2) + q_2 - 1)}_{=:\,\sigma_{15}}\big).
\end{aligned}
$$

The neural network $\mathcal{N}$ simulating $\mathcal{M}$ is shown in figure 4.2. As we already pointed out in corollary 3.1.1, the output of $\mathcal{N}$ becomes its own input until $\mathcal{M}$ reaches the halting state, i.e. until the third neuron of the output layer of $\mathcal{N}$ has activation equal to one. When that happens, the activation of the fourth neuron of the output layer will be the encoded output of $\mathcal{M}$, that is, the rational number corresponding to the sum of the two unary numbers in input to $\mathcal{M}$ itself. The weight matrices and bias vectors of $\mathcal{N}$ are shown in tables 4.2, 4.3, and 4.4, where we adopt the more compact notation $\texttt{TOP}(q) =: \zeta[q], \texttt{NONEMPTY}(q) =: \tau[q]$ introduced in definition 1.2.

4.2 Implementation

The results of section 4.1 can be deployed to obtain an ad hoc implementation of the example we are discussing, allowing us to verify the correctness of our simulation in a simple but representative specific scenario. The code we are going to present is written in Python 3.7.

First of all, we need a class modeling a stack whose content is encoded as a rational number (see equation 1.1) and whose operations are defined as in equations 1.8, 1.9, 1.10, and 1.11:

```
"""
stack.py
~~~~~~~~
"""

class Stack:

    def __init__(self, stack):
    """stack must be a list of binary values representing,
    from left to right, the stack content from top to bottom."""
        self.encoding = 0
```

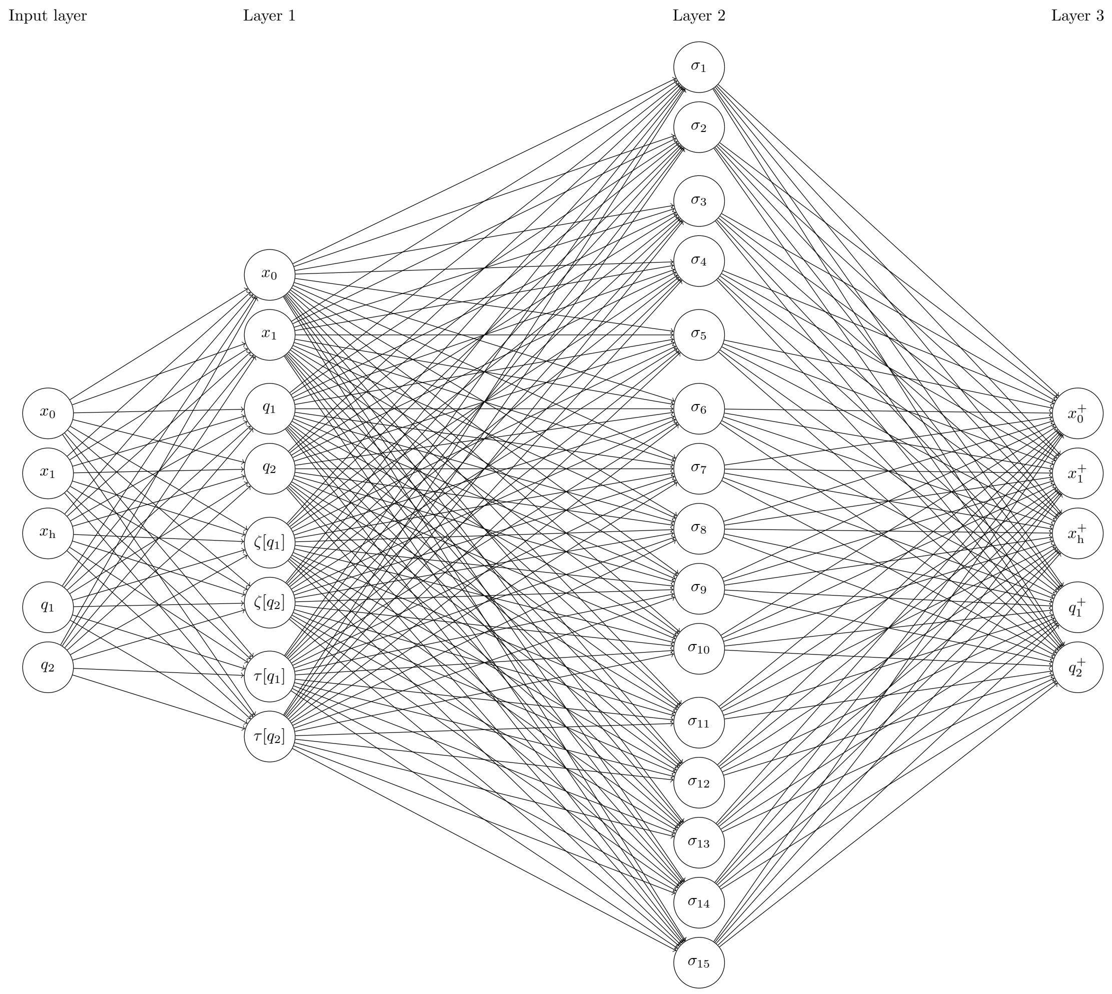

Figure 4.2: a recurrent neural network simulating the double-stack PDA in figure 4.1

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
$\zeta[q_1]$	0	0	0	4	0	-2
$\zeta[q_2]$	0	0	0	0	4	-2
$\tau[q_1]$	0	0	0	4	0	0
$\tau[q_2]$	0	0	0	0	4	0

Table 4.2: weights from the input layer to layer 1 and biases of layer 1

W^2	x_0	x_1	q_1	q_2	$\zeta[q_1]$	$\zeta[q_2]$	$\tau[q_1]$	$\tau[q_2]$	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 4.3: 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 4.4: weights from layer 2 to layer 3 and biases of layer 3

```
        for i in range(len(stack)):
            self.encoding += (2 * stack[i] + 1) / 4 ** (i + 1)

    def top(self):
        return self.encoding > 0.5

    def nonempty(self):
        return self.encoding != 0

    def push(self, symbol):
        if symbol != 0 and symbol != 1:
            raise ValueError('Stack symbols must be binary')
        self.encoding = self.encoding / 4 + (2 * symbol + 1) / 4

    def pop(self):
        if not self.nonempty():
            raise RuntimeError('Cannot pop an empty stack')
        self.encoding = 4 * self.encoding - 2 * self.top() - 1
```

Then, we write a class defining the double-stack PDA in figure 4.1:

```
"""
machine.py
~~~~~~~~~~
"""

from stack import Stack

class DoubleStackPDA:

    def __init__(self, stack1):
        """stack1 must be a list containing the two
        unary numbers to add separated by a zero."""
        self.stack1 = Stack(stack1)
        self.stack2 = Stack([])
        self.states = ['s0', 's1', 'sh']
        self.state = self.states[0]

    def update(self):
        if self.state is self.states[0] and \
                self.stack1.nonempty() and \
                self.stack1.top():
            self.stack1.pop()
            self.stack2.push(1)
        elif self.state is self.states[0] and \
                self.stack1.nonempty() and \
                not self.stack1.top():
            self.stack1.pop()
        elif self.state is self.states[0] and \
                not self.stack1.nonempty():
```

```
            self.state = self.states[1]
        elif self.state is self.states[1] and \
                self.stack2.nonempty():
            self.stack1.push(1)
            self.stack2.pop()
        elif self.state is self.states[1] and \
                not self.stack2.nonempty():
            self.state = self.states[2]
        else:
            raise RuntimeError('Undefined transition')

    def execute(self):
        while self.state is not self.states[2]:
            self.update()
        print('Stack 1 encoding = {}'.format(self.stack1.encoding))
        print('Stack 2 encoding = {}'.format(self.stack2.encoding))
```

Furthermore, by using tables 4.2, 4.3, and 4.4, we can implement the neural network in figure 4.2 (the following code is loosely inspired by [2]):

```
"""
network.py
~~~~~~~~~~
"""

import numpy
from stack import Stack

class Network:

    def __init__(self):
        self.weights = [
            numpy.array([[1, 0, 0, 0, 0],
                         [0, 1, 0, 0, 0],
                         [0, 0, 0, 1, 0],
                         [0, 0, 0, 0, 1],
                         [0, 0, 0, 4, 0],
                         [0, 0, 0, 0, 4],
                         [0, 0, 0, 4, 0],
                         [0, 0, 0, 0, 4]]),
            numpy.array([[1, 0, 0, 0, 1, 0, 1, 0],
                         [1, 0, 0, 0, -1, 0, 1, 0],
                         [1, 0, 0, 0, 0, 0, -1, 0],
                         [0, 1, 0, 0, 0, 0, 0, 1],
                         [0, 1, 0, 0, 0, 0, 0, -1],
                         [1, 0, 4, 0, -1, 0, 1, 0],
                         [1, 0, 4, 0, -3, 0, 1, 0],
                         [1, 0, 1, 0, 0, 0, -1, 0],
                         [0, 1, 1/4, 0, 0, 0, 0, 1],
                         [0, 1, 1, 0, 0, 0, 0, -1],
```

```
                         [1, 0, 0, 1/4, 1, 0, 1, 0],
                         [1, 0, 0, 1, -1, 0, 1, 0],
                         [1, 0, 0, 1, 0, 0, -1, 0],
                         [0, 1, 0, 4, 0, -2, 0, 1],
                         [0, 1, 0, 1, 0, 0, 0, -1]]),
            numpy.array([[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                         [0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]])
        ]
        self.biases = [
            numpy.array([0, 0, 0, 0, -2, -2, 0, 0]),
            numpy.array([-2, -1, 0, -1, 0, -4, -3,
                         -1, -5/4, -1, -9/4, -2, -1, -3, -1]),
            numpy.zeros(5)
        ]

    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]
        )
        stack1_encoding, stack2_encoding = feedforward(a)
        print('Stack 1 encoding = {}'.format(stack1_encoding))
        print('Stack 2 encoding = {}'.format(stack2_encoding))

def sigmoid(x):
    """Applies the sigmoid function elementwise."""
    return numpy.piecewise(
        x,
        [x < 0, numpy.logical_and(0 <= x, x <= 1), x > 1],
        [0, lambda x: x, 1]
    )
```

Finally, we write a test to see if the simulation actually works:

```
"""
test.py
~~~~~~~
"""

from machine import DoubleStackPDA
```

```
from network import Network

stack = [1, 0, 1]

print('\n======== Double-stack PDA ========\n')
m = DoubleStackPDA(stack)
m.execute()

print('\n======== Neural network ========\n')
n = Network()
n.execute(stack)
```

and, if we run such test, we get

```
======== Double-stack PDA ========

Stack 1 encoding = 0.9375
Stack 2 encoding = 0.0

======== Neural network ========

Stack 1 encoding = 0.9375
Stack 2 encoding = 0.0
```

which shows that the neural network correctly simulates the double-stack PDA.

Bibliography

[1] Hava T. Siegelmann, Eduardo D. Sontag. "On the Computational Power of Neural Nets". In *Proceedings of the fifth Annual ACM Workshop on Computational Learning Theory* (COLT 1992), pp. 440–449. DOI: `10.1145/130385.130432`. URL: `http://binds.cs.umass.edu/papers/1992_Siegelmann_COLT.pdf`

[2] Michael A. Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015. URL: `http://neuralnetworksanddeeplearning.com`

[3] Benjamin Wilson. *Siegelmann & Sontag's "On the Computational Power of Neural Nets".* From a talk given at the Sydney Machine Learning Meetup, 2018. URL: `http://drive.google.com/file/d/1HR-dXSI-dX16yibiXeeiz4pP2D8_KYDl/view`