Efficiently Representing Finite-state Automata With Recurrent Neural Networks

Anej Svete Ryan Cotterell
{asvete, ryan.cotterell}@inf.ethz.ch
ETH zürich

October 2023

Abstract

Understanding neural network architectures with formal models of computation promises to spark a better understanding of the network's capabilities and limitations. A long line of work has described recurrent neural networks (RNN) in terms of their connection to the well-understood finite-state automata (FSAs), whose sequential nature provides a useful analogy to how RNNs function. Minsky's [1954] construction first showed how RNNs can simulate FSAs and provided a way of understanding RNNs as FSAs. This paper presents a comprehensive review of this construction along with two additional classical results showcasing the relationship between RNNs and FSAs: The constructions due to Dewdney [1977] and Indyk [1995]. We are not only interested in *whether* an RNN can simulate an FSA, but also in the space requirements to do so: Whereas Minsky [1954] shows that an RNN can simulate an FSA with N states using $\mathcal{O}(N)$ neurons, Dewdney [1977] improved this to $\mathcal{O}\left(N^{\frac{3}{4}}\right)$ and Indyk [1995] further to $\mathcal{O}\left(\sqrt{N}\right)$, which he also showed to be optimal. We discuss the constructions, emphasizing their commonalities, and put them into the context of more modern research, focusing on the representational capacity of neural language models.

https://github.com/rycolab/weighted-minsky

1 Introduction

The recent success of neural language models has sparked considerable interest in understanding the theoretical capabilities of neural architectures such as [Elman, 1990, Hochreiter and Schmidhuber, 1997, Cho et al., 2014] and transformers [Vaswani et al., 2017]. Classical results state that RNNs with bounded-precision and computation time are equivalent to finite-state automata [McCulloch and Pitts, 1943, Minsky, 1954, Kleene, 1956, Dewdney, 1977, Alon et al., 1991, Indyk, 1995] whereas infinite-precision RNNs with unbounded computation time can simulate Turing machines [Siegelmann and Sontag, 1992]. More recent work has investigated the computational power of RNNs and transformers in more realistic settings [e.g., Hao et al., 2018, Weiss et al., 2018, Korsky and Berwick, 2019, Merrill, 2019, Merrill et al., 2020, Hewitt et al., 2020, Hahn, 2020, Chiang and Cholak, 2022, Merrill et al., 2022, Merrill and Tsilivis, 2022, Chiang et al., 2023, *inter alia*] and as language models [Svete and Cotterell, 2023, Nowak et al., 2023].

Many of the constructions rely on common principles such as *(1)* what it means for a neural network to simulate an automaton; *(2)* how the mechanisms and the parameters of the neural network can be used to simulate an automaton; and *(3)* how the components of a computational model can be encoded in a way suitable for a neural representation. To facilitate a better understanding of such fundamental principles and to encourage further research in this area, we review three classical constructions of simple (Elman) recurrent neural networks simulating finite-state automata: The constructions by Minsky [1954], Dewdney [1977], and Indyk [1995].

Understanding the representational capacity of neural architectures can help us understand the capabilities and limitations of models using them, as described by the following representative examples.

(i) Merrill [2019], Weiss et al. [2018] show that simple RNNs with a bounded activation function cannot implement a counting mechanism, implying that they cannot recognize languages requiring any notion of counting, for example, the Dyck language of balanced parentheses.

(ii) Hahn [2020] showed that the transformer architecture cannot reliably represent the PARITY language. This implies that a language model implemented using a transformer would not be able to correctly compute a sequence of repeated negations (since the true value of the statement depends on whether the number of negations is odd or even). For the same reason, the model would also not be able to compute the result of the multiplication of a sequence of -1's.

Characterizing neural models can thus help elucidate some potential real-world shortcomings of the models and thus provide concrete limitations to their generalization ability.

In this work, we focus on the relationship between RNNs with the Heaviside activation function $H(x) \stackrel{\text{def}}{=} \mathbb{1}\{x > 0\}$ and finite-state automata, one of the simplest computational models defining formal languages. While the restriction to the Heaviside activation function might seem limiting, it provides a useful framework for exploring the representational capacity of real-world systems. More importantly for this paper, the Heaviside activation function has a long history in the literature studying the representational capacity of neural networks. Among other things, the Heaviside activation function is used by the standard model considered by the three constructions of interest. Minsky's construction [Minsky, 1954] describes a way to represent an FSA with N states using a Heaviside RNN whose hidden state is of size linear in the number $\mathcal{A}$'s states. Dewdney [1977] then improved this construction by showing that an FSA with N states can be represented by a Heaviside RNN of size $\mathcal{O}\left(|\Sigma| N^{\frac{3}{4}}\right)$. The *optimal* lower bound in the number of states was then more thoroughly studied by Alon et al. [1991]. Using the same ideas as Dewdney [1977] but a specific trick to compress the size of the processing layer of the RNN further Indyk [1995] reduced Dewdney's [1977] bound to $\mathcal{O}\left(\sqrt{N}\right)$, which turns out to be asymptotically optimal. We review and provide the proofs of some aspects of the constructions in this paper.

2 Preliminaries

An **alphabet** Σ is a finite, non-empty set. A **formal language** is a subset of Σ's Kleene closure $\Sigma^* \stackrel{\text{def}}{=} \{\varepsilon\} \cup \Sigma \cup \Sigma^2 \cup \cdots$, where ε denotes the **empty string**. Finite-state automata are a tidy and well-understood formalism for describing languages.

Definition 2.1. *A **finite-state automaton** (FSA) is a 5-tuple $(\Sigma, Q, I, F, \delta)$ where Σ is an alphabet, Q a finite set of states, $I, F \subseteq Q$ the set of initial and final states, and $\delta \subseteq Q \times \Sigma \times Q$ set of transitions.*

We assume that states are identified by integers in $\mathbb{Z}_{|Q|} \stackrel{\text{def}}{=} \{0, \ldots, |Q| - 1\}$.[1] We also adopt a more

[1] For a cleaner presentation, we also assume that vectors and matrices are zero-indexed.

suggestive notation for transitions by denoting $(q, y, q') \in \delta$ as $q \xrightarrow{y} q'$ and call transitions of the form $q \xrightarrow{y} q'$ **y-transitions**. Furthermore, we define $\texttt{Par}(q, y) \stackrel{\text{def}}{=} \left\{q' \mid \exists y \in \Sigma \colon q' \xrightarrow{y} q \in \delta\right\}$ as the set of **y-parents** of q and the **children** of the state q as the set $\left\{q' \mid \exists y \in \Sigma \colon q \xrightarrow{y} q' \in \delta\right\}$.

Definition 2.2. *An FSA $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ is **deterministic** if $|I| = 1$ and for every $(q, y) \in Q \times \Sigma$, there is at most one $q' \in Q$ such that $q \xrightarrow{y} q' \in \delta$.*

Definition 2.3. *A **path** $\boldsymbol{\pi}$ is a sequence of consecutive transitions $q_0 \xrightarrow{y_0} q_1, \cdots, q_{N-1} \xrightarrow{y_{N-1}} q_N$. Its **length** $|\boldsymbol{\pi}|$ is the number of transition in it and its **scan** $\boldsymbol{s}(\boldsymbol{\pi})$ the concatenation of the symbols on them. The path $\boldsymbol{\pi}$ is **accepting** if $q_0 \in I$ and $q_N \in F$.*

We denote with $\Pi(\mathcal{A})$ the set of all paths in $\mathcal{A}$ and with $\Pi(\mathcal{A}, \boldsymbol{y})$ the set of all paths that scan $\boldsymbol{y} \in \Sigma^*$.

Definition 2.4. *An FSA $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ **accepts** the string $\boldsymbol{y} \in \Sigma^*$ if there exists an accepting path scanning $\boldsymbol{y}$. The **language accepted by** $\mathcal{A}$, $L(\mathcal{A})$, is the set of strings accepted by $\mathcal{A}$.*

Definition 2.5. *Let Σ be an alphabet. A language $L \subseteq \Sigma^*$ is **regular** if it is accepted by some FSA $\mathcal{A}$.*

Since we will be interested in the lower space bounds required to recognize regular languages, an important notion is that of a minimal FSA.

Definition 2.6. *The FSA $\mathcal{A}$ is **minimal** for the regular language L if there is no FSA with fewer states accepting the same language.*

2.1 Recurrent Neural Networks

Over the course of this paper, we will focus on Elman RNNs [Elman, 1990] as they are the easiest to analyze and special cases of more common networks, e.g., those based on long short-term memory [LSTM; Hochreiter and Schmidhuber, 1997] and gated recurrent units [GRUs; Cho et al., 2014].

Definition 2.7. *An **Elman RNN** $\mathcal{R} = (\Sigma, \sigma, D, \mathbf{U}, \mathbf{V}, \mathbf{b}, \mathbf{h}_0)$ is an RNN with the following hidden state recurrence:*

$$\mathbf{h}_t \stackrel{\text{def}}{=} \sigma\left(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{V}\mathbf{r}(y_t) + \mathbf{b}\right), \tag{1}$$

*where $\mathbf{h}_0$ is set to some vector in $\mathbb{R}^D$. $\mathbf{r} \colon \overline{\Sigma} \to \mathbb{R}^R$ is the symbol representation function and σ is an element-wise nonlinearity. $\mathbf{b} \in \mathbb{R}^D$, $\mathbf{U} \in \mathbb{R}^{D \times D}$, and $\mathbf{V} \in \mathbb{R}^{D \times R}$. We refer to the dimensionality of the hidden state, D, as the **size** of the RNN.*

On determinism. Unlike FSAs, Elman RNNs (and most other popular RNN architectures, such as the LSTM and GRU) implement inherently deterministic transitions between internal states. This means that they can only simulate *deterministic* FSAs and any non-deterministic FSA that we would like to simulate with an RNN first has to be determinized.

Common choices for the nonlinear function σ in Eq. (1) are the sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$ and the ReLU $\sigma(x) = \max(0, x)$. However, the resulting nonlinear interactions of the parameters and the inputs make the analysis of RNN LMs challenging. One fruitful manner to make the analysis tractable is making a simplifying assumption about σ. We focus on a particularly useful simplification, namely the use of the Heaviside activation function.[2]

[2]While less common now due to its non-differentiability, the Heaviside function was the original activation function used in early work on artificial neural network due to its close analogy to the firing of brain neurons [McCulloch and Pitts, 1943, Minsky, 1954, Kleene, 1956].

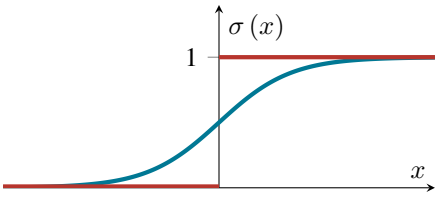


Figure 1: The sigmoid and Heaviside functions.

Definition 2.8. *The **Heaviside** function is defined as* $H(x) \stackrel{\text{def}}{=} \mathbb{1}\{x > 0\}$.

See Fig. 1 for the graph of the Heaviside function and its continuous approximation, the sigmoid. For cleaner notation, we define the set $\mathbb{B} \stackrel{\text{def}}{=} \{0, 1\}$. Using the Heaviside function, we can define the Heaviside Elman RNN, the main object of study in the rest of the paper.

Definition 2.9. *A **Heaviside Elman RNN** (HRNN) is an Elman RNN* $\mathcal{R} = (\Sigma, \sigma, D, \mathbf{U}, \mathbf{V}, \mathbf{b}, \mathbf{h}_0)$ *where* $\sigma = H$.

2.1.1 Performing the Logical AND with an HRNN

As we will see, simulating an FSA with an RNN requires the RNN to perform the logical AND operation between specific entries of binary vectors $\mathbf{x} \in \mathbb{B}^D$. The following fact shows how this can easily be performed by an HRNN with appropriately set parameters.

Fact 2.1. *Consider* m *indices* $i_1, \ldots, i_m \in \mathbb{Z}_D$ *and vectors* $\mathbf{x}, \mathbf{v} \in \mathbb{B}^D$ *such that* $v_i = \mathbb{1}\{i \in \{i_1, \ldots, i_m\}\}$, *i.e., with entries* 1 *at indices* $i_1, \ldots, i_m$. *Then,* $H\left(\mathbf{v}^\top \mathbf{x} - (m-1)\right) = 1$ *if and only if* $x_{i_k} = 1$ *for all* $k = 1, \ldots, m$. *In other words,*

$$H\left(\mathbf{v}^\top \mathbf{x} - (m-1)\right) = x_{i_1} \wedge \cdots \wedge x_{i_m}. \tag{2}$$

As a special case, $m = 2$ in Fact 2.1 corresponds to the AND operation of two elements, which is used in Minsky's construction. There, the vector $\mathbf{v}$ corresponds to the weights of a single neuron while $-(m-1)$ (-1 for $m = 2$ corresponds to its bias.

3 Minsky's Construction

This section describes the first of the three constructions of an HRNN simulating an FSA. The construction is due to Minsky [1954] and represents an FSA $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ with an HRNN of size exactly $|\Sigma||Q|$. Minsky's construction is formally described by the following theorem.

Theorem 3.1. *Let* $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ *be an FSA. Then, there exists an HRNN RNN of size* $|\Sigma||Q|$ *correctly simulating* $\mathcal{A}$.

We describe the full construction of an HRNN simulating a given FSA in the next subsection. The full construction is described to showcase the mechanism with which the HRNN can simulate the transitions of a given FSA and give intuition on why this might, in general, require a large number of parameters in the HRNN. Many principles and constraints of the simulation are also reused later in the presentation of

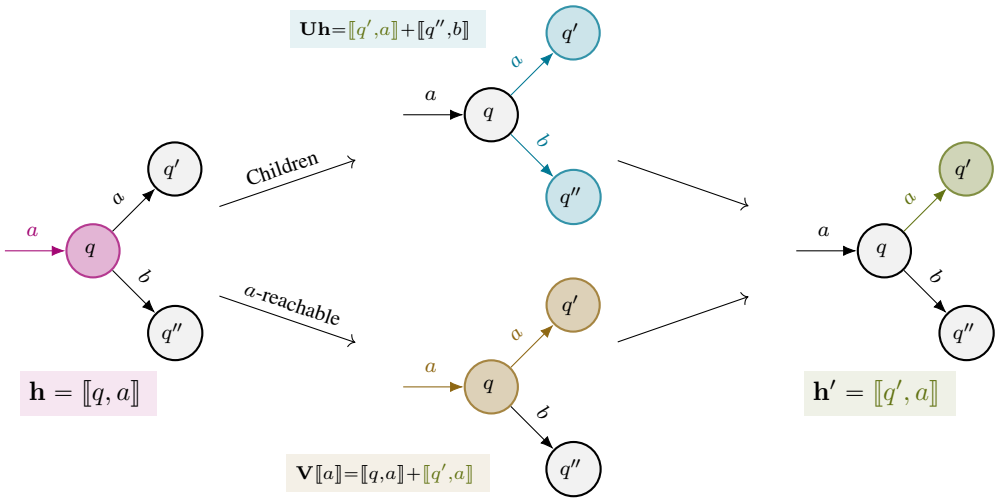


Figure 2: A high-level illustration of how the transition function of the FSA is simulated in Minsky's construction on a fragment of an FSA starting at q (encoded in $\mathbf{h}$) and reading the symbol a. The top path disjoins the representations of the children of q, whereas the bottom path disjoins the representations of states reachable by an a-transition. The Heaviside activation conjoins these two representations into $\mathbf{h}'$ (rightmost fragment).

the more efficient constructions of the HRNN simulating the FSA and the lower bounds on the size of the HRNN.

For an FSA $\mathcal{A} = (\Sigma, Q, I, F, \delta)$, we construct an HRNN $\mathcal{R}$ with $\mathcal{R} = (\Sigma, \sigma, D, \mathbf{U}, \mathbf{V}, \mathbf{b}, \mathbf{h}_0)$ defining the same distribution over Σ^*. The idea is to simulate the transition function δ with the Elman recurrence by appropriately setting $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{b}$.

Let $n \colon Q \times \Sigma \to \mathbb{Z}_{|Q||\Sigma|}$ be a permutations of $Q \times \Sigma$, $m \colon \Sigma \to \mathbb{Z}_{|\Sigma|}$ a permutation of Σ, and $\overline{m} \colon \overline{\Sigma} \to \mathbb{Z}_{|\overline{\Sigma}|}$ a permutation of $\overline{\Sigma}$. We use n, m, and $\overline{m}$ to define the one-hot encodings $[\![\cdot]\!]$ of state–symbol pairs and of the symbols, i.e., we assume that $[\![q, y]\!]_d = \mathbb{1}\{d = n(q, y)\}$ and $[\![y]\!]_d = \mathbb{1}\{d = m(y)\}$ for $q \in Q$ and $y \in \Sigma$.

HRNN's hidden states. The hidden state $\mathbf{h}_t$ of $\mathcal{R}$ will represent the one-hot encoding of the current state q_t of $\mathcal{A}$ at time t together with the symbol y_t upon reading which $\mathcal{A}$ entered q_t. Formally,

$$\mathbf{h}_t = [\![(q_t, y_t)]\!] \in \mathbb{B}^{|Q||\Sigma|}. \tag{3}$$

There is a small caveat: How do we set the incoming symbol of $\mathcal{A}$'s initial state q_ι? As we show later, the symbol y_t in $\mathbf{h}_t = [\![(q_t, y_t)]\!]$ does not affect the subsequent transitions—it is only needed to determine the target of the current transition. Therefore, we can set $\mathbf{h}_0 = [\![(q_\iota, y)]\!]$ for any $y \in \Sigma$.

Encoding the transition function. The idea of defining $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{b}$ is for the Elman recurrence to perform, upon reading y_{t+1}, element-wise conjunction between the representations of the children of q_t and the representation of the states $\mathcal{A}$ can transition into after reading in y_{t+1} from *any state*.[3] The

[3] See Fact 2.1 in §2.1.1 for a discussion of how an HRNN can implement the logical AND operation.

former is encoded in the recurrence matrix $\mathbf{U}$, which has access to the current hidden state encoding q_t while the latter is encoded in the input matrix $\mathbf{V}$, which has access to the one-hot representation of y_{t+1}. Conjoining the entries in those two representations will, due to the determinism of $\mathcal{A}$, result in a single non-zero entry: One representing the state which can be reached from q_t (1^{st} component) using the symbol y_{t+1} (2^{nd} component); see Fig. 2.

More formally, the recurrence matrix $\mathbf{U}$ lives in $\mathbb{B}^{|\Sigma||Q|\times|\Sigma||Q|}$. Each column $\mathbf{U}_{:\,,n(q,y)}$ represents the children of the state q in the sense that the column contains 1's at the indices corresponding to the state–symbol pairs (q', y') such that $\mathcal{A}$ transitions from q to q' after reading in the symbol y'. That is, for $q, q' \in Q$ and $y, y' \in \Sigma$, we define

$$U_{n(q',y'),n(q,y)} \stackrel{\text{def}}{=} \mathbb{1}\left\{ q_t \xrightarrow{y'} q' \in \delta \right\}. \tag{4}$$

Since y is free, each column is repeated $|\Sigma|$-times: Once for every $y \in \Sigma$—this is why, after entering the next state, the symbol used to enter it is not relevant for the determination of the subsequent transitions and, in the case of the initial state, any incoming symbol can be chosen to set $\mathbf{h}_0$.

The input matrix $\mathbf{V}$ lives in $\mathbb{B}^{|\Sigma||Q|\times|\Sigma|}$ and encodes the information about which states can be reached by which symbols (from *any* state). The non-zero entries in the column corresponding to $y' \in \Sigma$ correspond to the state–symbol pairs (q', y') such that q' is reachable with y' from *some* state:

$$V_{n(q',y'),m(y')} \stackrel{\text{def}}{=} \mathbb{1}\left\{ \circ \xrightarrow{y'} q' \in \delta \right\}. \tag{5}$$

Lastly, we define the bias as $\mathbf{b} \stackrel{\text{def}}{=} -\mathbf{1} \in \mathbb{R}^{|Q||\Sigma|}$, which allows the Heaviside function to perform the needed conjunction. The correctness of this process is formally proved with the following two lemmas.

Lemma 3.1. *Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be a deterministic FSA, $\boldsymbol{y} = y_1 \ldots y_T \in \Sigma^*$, and q_t the state arrived at by $\mathcal{A}$ upon reading the prefix $\boldsymbol{y}_{\leqslant t}$. Let $\mathcal{R}$ be the HRNN specified by the Minsky construction for $\mathcal{A}$, n the ordering defining the one-hot representations of state-symbol pairs by $\mathcal{R}$, and $\mathbf{h}_t$ $\mathcal{R}$'s hidden state after reading $\boldsymbol{y}_{\leqslant t}$. Then, it holds that $\mathbf{h}_0 = [\![(q_\iota, y)]\!]$ where q_ι is the initial state of $\mathcal{A}$ and $y \in \Sigma$ and $\mathbf{h}_T = [\![(q_T, y_T)]\!]$.*

Proof. Define $s(\mathbf{h} = [\![(q, y)]\!]) \stackrel{\text{def}}{=} q$. We can then restate the lemma as $s(\mathbf{h}_T) = q_T$ for all $\boldsymbol{y} \in \Sigma^*$, $|\boldsymbol{y}| = T$. Let $\boldsymbol{\pi}$ be the $\boldsymbol{y}$-labeled path in $\mathcal{A}$. We prove the lemma by induction on the string length T.

Base case: $T = 0$**.** Holds by the construction of $\mathbf{h}_0$.

Inductive step: $T > 0$**.** Let $\boldsymbol{y} \in \Sigma^*$ with $|\boldsymbol{y}| = T$ and assume that $s(\mathbf{h}_{T-1}) = q_{T-1}$.

We prove that the specifications of $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{b}$ ensure that $s(\mathbf{h}_T) = q_T$. By definition of the recurrence matrix $\mathbf{U}$ (cf. Eq. (4)), the vector $\mathbf{U}\mathbf{h}_{T-1}$ will contain a 1 at the entries $n(q', y')$ for $q' \in Q$ and $y' \in \Sigma$ such that $q_T \xrightarrow{y'} q' \in \delta$. This can equivalently be written as $\mathbf{U}\mathbf{h}_{T-1} = \bigvee_{q_T \xrightarrow{y'} q' \in \delta} [\![(q', y')]\!]$, where the disjunction is applied element-wise.

On the other hand, by definition of the input matrix $\mathbf{V}$ (cf. Eq. (5)), the vector $\mathbf{V}[\![y_T]\!]$ will contain a 1 at the entries $n(q', y_T)$ for $q' \in Q$ such that $\circ \xrightarrow{y_T} q' \in \delta$. This can also be written as $\mathbf{V}[\![y_T]\!] = \bigvee_{\circ \xrightarrow{y_T} q' \in \delta} [\![(q', y_T)]\!]$.

By Fact 2.1, $H\left(\mathbf{U}\mathbf{h}_{T-1} + \mathbf{V}[\![y_T]\!] + \mathbf{b}\right)_{n(q',y')} = H\left(\mathbf{U}\mathbf{h}_{T-1} + \mathbf{V}[\![y_T]\!] - \mathbf{1}\right)_{n(q',y')} = 1$ holds if and only if $\left(\mathbf{U}\mathbf{h}_{T-1}\right)_{n(q',y')} = 1$ and $\left(\mathbf{V}[\![y_T]\!]\right)_{n(q',y')} = 1$. This happens if

$$q_T \xrightarrow{y'} q' \in \delta \text{ and } \circ \xrightarrow{y_T} q' \in \delta \iff q_T \xrightarrow{y_T} q', \tag{6}$$

i.e., if and only if $\mathcal{A}$ transitions from q_T to q_T upon reading y_T (it transitions only to q_T due to determinism).

Since the string $\boldsymbol{y}$ was arbitrary, this finishes the proof. ■

4 Dewdney's Construction

This section describes the construction due to Dewdney [1977] in our notation. Since some of the parts are very similar to the construction due to Indyk [1995], those parts are reused in §5 and introduced more generally.

Representing states of the FSA. Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be a deterministic FSA. Recall that Minsky's construction encodes the $\mathcal{A}$'s current state as a one-hot encoding of the state-symbol pair. The construction due to Dewdney [1977], on the other hand, represents the states separately from the symbols. It encodes the states with *two-hot representations* by using the coefficients of what we call a square-root state representation. This results in representations of states of size $\mathcal{O}\left(\sqrt{|Q|}\right)$. The input symbols are incorporated into the hidden state *separately*.[4]

Definition 4.1. *Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be an FSA and $s \stackrel{\text{def}}{=} \lceil\sqrt{|Q|}\rceil$. We define the **square-root state representation** of $\mathcal{A}$'s states $q \in Q$ as*[5]

$$\phi_2(q) \stackrel{\text{def}}{=} \left(\left\lfloor \frac{q}{s} \right\rfloor, q \mod s\right). \tag{7}$$

We denote the inverse of ϕ_2 with ϕ_2^{-1} and further define for $k \in \mathbb{Z}_s$

$$\phi_2^{-1}(k, \cdot) \stackrel{\text{def}}{=} \{q \in Q \mid \varphi_0 = k \text{ where } \boldsymbol{\varphi} = \phi_2(q)\} \tag{8}$$

and $\phi_2^{-1}(\cdot, k)$ analogously.

Specifically, we will denote $\phi_2^{-1}(k, \cdot)$ and $\phi_2^{-1}(\cdot, k)$ with k in the j^{th} position (with $j \in \mathbb{Z}_2$, 0 for $\phi_2^{-1}(k, \cdot)$ and 1 for $\phi_2^{-1}(\cdot, k)$) as $\Phi_{k,j}$.

We can think of the function ϕ_2 as representing states of the FSA in a two-dimensional space $\mathbb{Z}_s \times \mathbb{Z}_s$. However, to efficiently simulate $\mathcal{A}$ with an HRNN, it is helpful to think of $\phi_2(q)$ in two different ways: as a vector $\mathbf{v} \in {\mathbb{N}_{\geqslant 0}}^{2|Q|}$, or as a matrix in $\mathbb{B}^{|Q| \times |Q|}$ in the following sense.

Definition 4.2. *Given a square-root state representation function ϕ_2, we define the **vector representation** of the state $q \in Q$ as the vector $\mathbf{v}(q) \in \mathbb{B}^{2|Q|}$ with*

$$\mathbf{v}(q)_{\varphi_0} = 1 \tag{9}$$

$$\mathbf{v}(q)_{s+\varphi_1} = 1, \tag{10}$$

[4] This again adds a factor $|\Sigma|$ to the size of the hidden state, as we discuss later.

[5] Notice that $\phi_2(q)$ represents the coefficients of the expression of $q \in \mathbb{N}$ in base s.

where $\boldsymbol{\varphi} = (\varphi_0, \varphi_1) = \phi_2(q)$*, and all other entries* 0*. Furthermore, we define the* ***matrix representation*** *of the state* $q \in Q$ *as the matrix* $\mathbf{B} \in \mathbb{B}^{|Q| \times |Q|}$ *with*

$$\mathbf{B}(q)_{\varphi_0 \varphi_1} = 1 \tag{11}$$

and all other entries 0*.*

Dewdney's construction also heavily relies on the representations of sets of states. We define those additively.

Definition 4.3. *Let* $\mathcal{Q} \subseteq Q$ *be a set of states. We define the vector representation of* $\mathcal{Q}$ *as the vector*

$$\mathbf{v}(\mathcal{Q}) \stackrel{\text{def}}{=} \bigvee_{q \in Q} \mathbf{v}(q). \tag{12}$$

Similarly, we define the matrix representation of $\mathcal{Q}$ *as the matrix*

$$\mathbf{B}(\mathcal{Q}) \stackrel{\text{def}}{=} \bigvee_{q \in Q} \mathbf{B}(q). \tag{13}$$

To help understand the above definitions, we give an example of a FSA and the representations of its states.

Example 4.1. *Consider the FSA in Fig. 3, for which* $s = \lceil \sqrt{|Q|} \rceil = \lceil \sqrt{3} \rceil = 2$*, meaning that*

$$\phi_2(0) = (0,0) \qquad \phi_2(1) = (0,1) \qquad \phi_2(2) = (1,0), \tag{14}$$

resulting in the state-to-vector mapping[6]

$$\mathbf{v}(0) = \begin{pmatrix} 1 & 0 & | & 1 & 0 \end{pmatrix} \tag{15}$$

$$\mathbf{v}(1) = \begin{pmatrix} 1 & 0 & | & 0 & 1 \end{pmatrix} \tag{16}$$

$$\mathbf{v}(2) = \begin{pmatrix} 0 & 1 & | & 1 & 0 \end{pmatrix}, \tag{17}$$

and the state-to-matrix mapping

$$\mathbf{B}(0) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \qquad \mathbf{B}(1) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \qquad \mathbf{B}(2) = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}. \tag{18}$$

The two components of the vector representations separated by "|" denote the two halves of the representation vectors, corresponding to the two components of $\phi_2(q)$*.*

High-level idea of Dewdney's construction. Given these definitions, the intuition behind Dewdney's construction of an HRNN simulating an FSA $\mathcal{A}$ is the following:

1. Represent $\mathcal{A}$'s states as vectors in $\mathbb{B}^{2s}$, or, equivalently, matrices in $\mathbb{B}^{s \times s}$.
2. For each $q \in Q$, construct the matrix representation of the set of y-parents $\mathbf{B}(\mathtt{Par}(q, y))$ for all $y \in \Sigma$.
3. To simulate $\mathcal{A}$'s transition function δ, compare the representation of the current state q_t with all constructed parent matrices $\mathbf{B}(\mathtt{Par}(q, y_t))$ given the current input symbol y_t. Activate the two-hot representation of the (unique) state q_{t+1} for which the representation of q_t was detected in q_{t+1}'s parent matrix for symbol y_t, $\mathbf{B}(\mathtt{Par}(q_{t+1}, y_t))$.

[6] Despite the notation $(\ldots \mid \ldots)$, we assume we are working with column vectors.

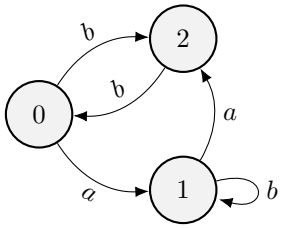


Figure 3: An example of a fragment of an FSA.

Simulating the transition function of an FSA by detecting preceding states. We elaborate on the last point above since it is the central part of the construction.[7] The idea of simulating the transition function δ is reduced to detecting *whose parent* given the current input symbol y_t is currently active—naturally, this should be the state active at $t+1$. Concretely, consider again the FSA $\mathcal{A}$ in Fig. 3. The parents of the three states, indexed by the incoming symbols are: for 0 $\{b:2\}$, for 1 $\{a:1, b:0\}$, and for 2 $\{a:1, b:0\}$. Suppose that at some time t, $\mathcal{A}$ is in state 0 and is reading in the symbol b. Then, since the state 0 is the b-parent of the state 2, we know that at time $t+1$, $\mathcal{A}$ will be in state 2. This principle can be applied more generally: To determine the state of an FSA at time $t+1$, we simply have to somehow detect whose *parent* is active at time t *given the current input symbol* at time t.

The crux of Dewdney's construction is then the following:[8] How do we, using only the Elman update rule, determine whose y_t-parent is active at time t? This can be done by *detecting* which parent matrix $\mathbf{B}\left(\mathtt{Par}\left(q, y_t\right)\right)$ the representation of the current state q_t is included in in the sense that if $\phi_2\left(q_t\right) = \boldsymbol{\varphi}$, it holds that $\mathbf{B}\left(\mathtt{Par}\left(q, y_t\right)\right)_{\boldsymbol{\varphi}_0 \boldsymbol{\varphi}_1} = 1$. To be able to formally talk about the detection of a representation in a set of parents, we define several notions of **matrix detection**.

Informally, we say that a matrix is easily detectable if the presence of its non-zero elements can be detected using a single neuron in the hidden layer of a HRNN.

Definition 4.4. *Let $\mathbf{B} \in \mathbb{B}^{D \times D}$ be a binary matrix. We say that $\mathbf{B}$ is **easily detectable** if there exist $\mathbf{w} \in \mathbb{Q}^{2D}$ and $b \in \mathbb{Q}$ (neuron coefficients) such that*

$$\sigma\left(\langle \mathbf{e}_{ij}, \mathbf{w} \rangle + b\right) = 1 \iff B_{ij} = 1, \tag{19}$$

where $\mathbf{e}_{ij} = \begin{pmatrix} \mathbf{e}_i & | & \mathbf{e}_j \end{pmatrix}$ refers to the $2D$-dimensional vector with 1's at positions i and $D+j$. In words, this means that the neuron defined by $\mathbf{w}, b$ fires on the input $\mathbf{e}_{ij}$ if and only if $B_{ij} = 1$.

We define detectable matrices as the matrices which can be detected using a conjunction of two neurons.

Definition 4.5. *Let $\mathbf{B} \in \mathbb{B}^{D \times D}$ be a binary matrix. We say that $\mathbf{B}$ is **detectable** if there exist $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{Q}^{2D}$ and $b_1, b_2 \in \mathbb{Q}$ such that*

$$\sigma\left(\langle \mathbf{e}_{ij}, \mathbf{w}_1 \rangle + b_1\right) = 1 \wedge \sigma\left(\langle \mathbf{e}_{ij}, \mathbf{w}_2 \rangle + b_2\right) = 1 \iff B_{ij} = 1. \tag{20}$$

Furthermore, we say that a matrix is (easily) **permutation-detectable** if there exist permutation matrices $\mathbf{P}$ and $\mathbf{Q}$ such that $\mathbf{PBQ}$ is (easily) detectable.

[7] Later, we will see that Indyk [1995] uses the exact same idea for simulating δ.

[8] Again, the same applies to Indyk [1995].

Intuitively, this means that one can effectively replace an easily detectable matrix $\mathbf{B}$ with *a single neuron*: Instead of specifying the matrix explicitly, one can simply detect if an entry B_{ij} of $\mathbf{B}$ is 1 by passing $\mathbf{e}_{ij}$ through the neuron and seeing if it fires. This reduces the space complexity from D^2 to $2D$. Similarly, one can replace a detectable matrix with *two* neurons. As shown in Fact 2.1, the required conjunction of the two resulting neurons can then easily be performed by a third (small) neuron, meaning that a detectable matrix is effectively represented by a two-layer MLP.

An example of easily detectable matrices are the so-called **northwestern** matrices.

Definition 4.6. *A matrix $\mathbf{B} \in \mathbb{B}^{D \times D}$ is **northwestern** if there exists a vector $\boldsymbol{\alpha}$ with $|\boldsymbol{\alpha}| = D$ and $D \geqslant \alpha_1 \geqslant \ldots \geqslant \alpha_D \geqslant 0$ such that*

$$B_{ij} = 1 \iff j \leqslant \alpha_i. \tag{21}$$

Intuitively, northwestern matrices contain all their ones contiguously in their upper left (northwest) corner. An example of a northwestern matrix for $\boldsymbol{\alpha} = \begin{pmatrix} 2 & 1 & 1 \end{pmatrix}$ is

$$\mathbf{B} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}. \tag{22}$$

Lemma 4.1. *Northwestern matrices are easily detectable.*

Proof. Define

$$\mathbf{w} \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{\alpha} & | & D & \ldots & 1 \end{pmatrix}$$

and $b = -D$. It is easy to see that for any $\mathbf{e}_{ij}$ where $\mathbf{B}_{ij} = 1$, it holds that

$$\begin{aligned} \langle \mathbf{e}_{ij}, \mathbf{w} \rangle &= \alpha_i + (D - j + 1) \geqslant j + D - j + 1 = D + 1 \\ \implies H\left(\langle \mathbf{e}_{ij}, \mathbf{w} \rangle + b\right) &= H\left(\langle \mathbf{e}_{ij}, \mathbf{w} \rangle - D\right) = 1. \end{aligned}$$

On the other hand, for $B_{ij} = 0$, we have

$$\begin{aligned} \langle \mathbf{e}_{ij}, \mathbf{w} \rangle &= \alpha_i + (D - j + 1) < j + D - j + 1 = D \\ \implies H\left(\langle \mathbf{e}_{ij}, \mathbf{w} \rangle + b\right) &= H\left(\langle \mathbf{e}_{ij}, \mathbf{w} \rangle - D\right) = 0. \end{aligned}$$

■

A more general useful class of detectable matrices are line matrices [Dewdney, 1977].

Definition 4.7. *A binary matrix $\mathbf{B} \in \mathbb{B}^{D \times D}$ is a **line matrix** if any of the following conditions hold:*

1. *All $\mathbf{B}$'s ones lie either in the same row ($\mathbf{B}$ is a **row matrix**) or in the same column ($\mathbf{B}$ is a **column matrix**).*
2. *$\mathbf{B}$ is a* transversal*, i.e., a matrix in which there is at most one 1 in any column and row.*

Lemma 4.2. *Row and column matrices are easily permutation-detectable.*

Proof. Let $i, N \in \mathbb{Z}_D$ and $\mathbf{B}$ be a row matrix with $\mathbf{B}_{ij_n} = 1$ for $n \in \mathbb{Z}_N$, i.e., a row matrix with all its ones in the i^{th} row. Define $\mathbf{P} \in \mathbb{B}^{D\times D}$ as $P_{1i} = 1$ and 0 elsewhere and $\mathbf{Q} \in \mathbb{B}^{D\times D}$ with $\mathbf{Q}_{j_n n} = 1$ and 0 elsewhere. Then, $\mathbf{PBQ}$ contains all its 1 in its northwestern corner (contiguously in the first row) and is thus easily detectable. Let $\mathbf{w} \stackrel{\text{def}}{=} \begin{pmatrix}\boldsymbol{\alpha} & | & D & \dots & 1\end{pmatrix}$, $b = D$ be the neuron weights from Lemma 4.1. Define $\mathbf{w}' \stackrel{\text{def}}{=} \begin{pmatrix}\mathbf{P}^\top\boldsymbol{\alpha} & | & \mathbf{Q}(D & \dots & 1)\end{pmatrix}$, $b' = D$. It is easy to see that this "rearranges" the components of the neuron recognizing the northwestern matrix $\mathbf{PBQ}$ to make them recognize the original matrix, meaning that the neuron defined by $\mathbf{w}'$ and b' recognizes the line matrix. The proof for a column matrix is analogous. ■

Lemma 4.3. *Transversals are permutation-detectable.*

Proof. The core idea of this proof is that every transversal can be permuted into a diagonal matrix, which can be written as a Hadamard product of a lower-triangular and an upper-triangular matrix.

Let $\mathbf{B}$ be a transversal. Pre-multiplying $\mathbf{B}$ with its transpose $\mathbf{P} \stackrel{\text{def}}{=} \mathbf{B}^\top$ results in a diagonal matrix. It is easy to see that $\mathbf{PB}$ can be written as a Hadamard product $\mathbf{H}_1 \otimes \mathbf{H}_2$ of a lower-triangular matrix $\mathbf{H}_1$ and an upper-triangular matrix $\mathbf{H}_2$. Both are easily permutation detectable. A conjunction of the neurons detecting $\mathbf{H}_1$ and $\mathbf{H}_2$ (again, performed by another neuron) detects the original matrix $\mathbf{B}$. In the following, we will refer to $\mathbf{H}_1$ and $\mathbf{H}_2$ as the *factors* of the transversal. ■

Crucially, any binary matrix $\mathbf{B} \in \mathbb{B}^{D\times D}$ can be decomposed into a set of line matrices $\mathcal{B}$ whose disjunction is $\mathbf{B}$: $\bigvee_{\mathbf{M}\in\mathcal{B}} \mathbf{M} = \mathbf{B}$. It is easy to see that $\mathbf{B}_{ij} = 1$ if and only if there exists $\mathbf{M} \in \mathcal{B}$ such that $\mathbf{M}_{ij} = 1$. This means that non-zero entries of *any* $\mathbf{B} \in \mathbb{B}^{D\times D}$ decomposed into the set of line matrices $\mathcal{B}$ can be detected using an MLP in two steps:

1. Detect the non-zero entries of the individual line matrices from the decomposition $\mathcal{B}$ (which are, as shown above, detectable).
2. Take a disjunction of the detections of the individual line matrices to result in the activation of the original matrix.

The disjunction can again be performed by applying another 2-layer MLP to the activations of the line matrices. An important consideration in both Dewdney's as well as Indyk's construction later will be *how large* $\mathcal{B}$ has to be.

Using matrix decomposition and detection for simulating the transition function. We now describe how Dewdney's construction uses matrix detection based on the decomposition of matrices into line matrices to simulate an FSA using an HRNN. From a high level, the update steps of the HRNN will, just like in Minsky's construction, simulate the transition function of the simulated FSA. However, in contrast to the Minsky construction, in which each transition step in the FSA was implemented by a *single* application of the Elman update rule, here, a single transition in the FSA will be implemented using *multiple* applications of the Elman update rule, the end result of which is the activation of the two-hot representation of the appropriate next state. Nonetheless, there are, abstractly, two sub-steps of the update step, analogous to the Minsky construction (cf. Fig. 2):

1. Detect the activations of all possible next states, considering any possible input symbol (performed by the term $\mathbf{U}\mathbf{h}_t$ in Minsky's construction).
2. Filter the activations of the next states by choosing only the one transitioned into by a y_t-transition (performed by conjoining with the term $\mathbf{V}[\![y_t]\!]$ in Minsky's construction).

The novelty of Dewdney's construction comes in the first sub-step: How can the Elman update step be used to activate the two-hot representation of q_t's children? As alluded to, this relies on the pre-computed parent matrices $\texttt{Par}\,(q, y)$ (cf. Definition 4.2). The parent matrices of individual states are compressed (disjoined) into component-activating matrices, the representation matrices of the parents of specific *sets* of states (cf. Definition 4.3), defined through the function ϕ_2 in the following sense.

Definition 4.8. *A **component-activating matrix** is the representation matrix* $\mathbf{B}_{j,y,k} \stackrel{\text{def}}{=} \mathbf{B}\,(\texttt{Par}\,(\Phi_{k,j}, y))$ *for some* $k \in \mathbb{Z}_r$ *and* $j \in \mathbb{Z}_2$.

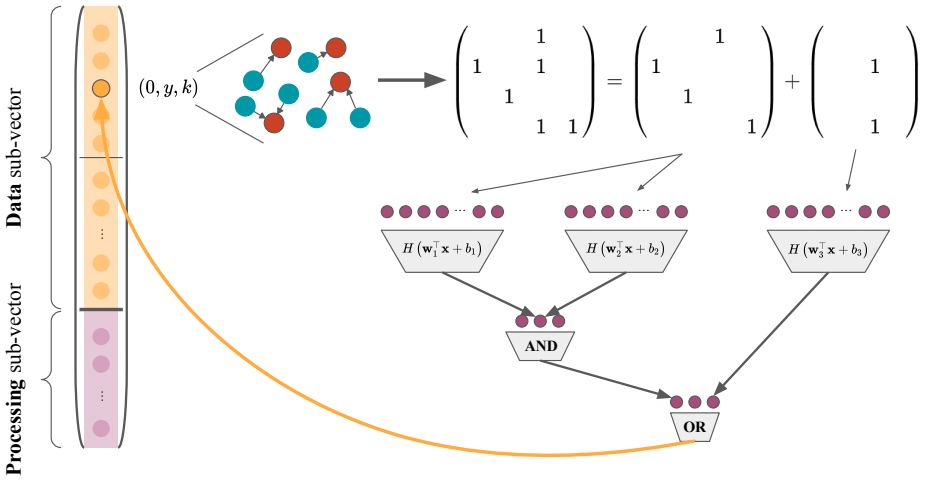
Intuitively, the component-activating matrix $\mathbf{B}_{j,y,k}$ is the result of the disjunction of the matrix representations of all y-parents q of all states q' whose j^{th} component of the vector $\phi_2\,(q')$ equals k. This results in $2|\Sigma|s$ matrices. They can be pre-computed and naturally depend on the transition function δ. The name component-activating matrix is inspired by the fact that each of the matrices "controls" the activation of one of the $2|\Sigma|s$ neurons in a specific sub-vector of the HRNN hidden state. That is, each component-activating matrix controls a particular dimension, indexed by the tuple (j, y, k) for $j \in \mathbb{B}, y \in \Sigma, k \in \mathbb{Z}_s$, in the **data sub-vector** of the HRNN hidden state. As we will see shortly, they contain all the information required for simulating $\mathcal{A}$ with a HRNN.

To define the transition function of the HRNN simulating $\mathcal{A}$, all $2|\Sigma|s$ component-activating matrices are decomposed into permutation-detectable line matrices (cf. Definition 4.7) whose activations are combined (disjoined) into the activations of individual component-activating matrices. Analogously to above, we will denote the sets of line matrices decomposing the component-activating matrices as $\mathcal{B}_{j,y,k}$, i.e., $\mathbf{B}_{j,y,k} = \bigvee_{\mathbf{M} \in \mathcal{B}_{j,y,k}} \mathbf{M}$. The dimensions of the hidden state corresponding to the activations of the line matrices before they are combined into the activations of the component-activating matrices form the **processing sub-vector** of the HRNN hidden state since they are required in the pre-processing steps of the update step to determine the activation of the actual hidden state. This is schematically drawn in Fig. 4a.

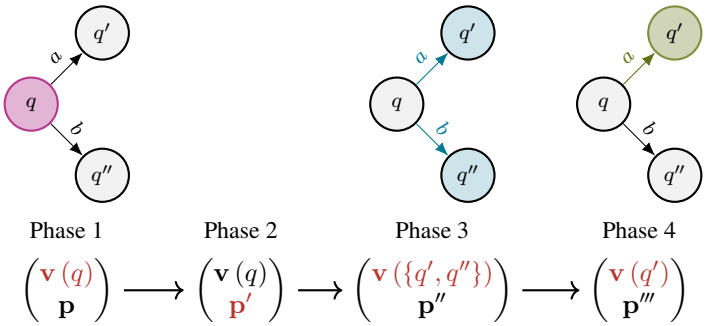
For any component-activating matrix $\mathbf{B}$ decomposed into the set of line matrices $\mathcal{B}$, we know by Lemmas 4.2 and 4.3 that all $\mathbf{M} \in \mathcal{B}$ are detectable by a single-layer MLP. By adding an additional layer to the MLP, we can disjoin the detections of $\mathbf{M} \in \mathcal{B}$ into the detection of $\mathbf{B}$. More abstractly, this MLP, therefore, detects the activation of *one* of the $2|Q|s$ cells of the data sub-vector of the HRNN hidden state—all of them together then form the two-hot encoding of all possible next states of the FSA (before taking into account the input symbol). Designing $2|Q|s$ such single-values MLPs, therefore, results in an MLP activating the two-hot representations of all possible next states of the simulated FSA. Conjoining these activations with the input symbol, analogously to how this is done in the Minsky construction, results in the activation of the two-hot representation of only the actual next state of the simulated FSA. This is illustrated in Fig. 4b.

High-level overview of simulating a transition. In summary, after decomposing all the component-activating matrices into the sets $\mathcal{B}_{j,y,k}$, the detection of all candidate next states (before considering the input symbol) in the update step of HRNN is composed of the following sub-steps.

1. Compute the activations of the two *factors* of all the transversals in $\mathcal{B}_{j,y,k}$ for all j, y, k (Lemma 4.3).
2. Conjoin the activations of the two factors into the activations of the transversals (Lemma 4.3).
3. Compute the activations of the column and row matrices in $\mathcal{B}_{j,y,k}$ for all j, y, k (Lemma 4.2).
4. Disjoin of the activations of all the line matrices (transversals, row, and column matrices) in $\mathcal{B}_{j,y,k}$ for all x, y, k to compute the activations of all $2|\Sigma|s$ component-activatimg matrices.



(a) High-level overview of Dewdney's construction. The highlighted orange neuron in the representation of the state from the data sub-vector corresponds to the activation of one of the components of the red states (which have in common that their 0^{th} component of $\phi_2(q)$ is the same). The matrix corresponding to the disjunction of the representations of their *y-parents* (blue states) is decomposed into two line matrices—a transversal and a column matrix. The non-zero elements of the former can be detected by a conjunction of two neurons while the non-zero elements of the latter can be detected directly by a single neuron. Those activations are then disjoined to result in the activation in the orange neuron. The purple neurons in the processing sub-vector are composed of the neurons in the networks implementing the detection of line matrices and their conjunctions and disjunctions (also shown in purple).



(b) A high-level illustration of how the transition function of the FSA is implemented in Dewdney's construction on an example of an FSA fragment, where the simulated automaton is initially in the state q and reads the symbol a, transitioning to q'. The components whose changes are relevant at a given step are highlighted. Starting in the state q, which is stored in the data sub-vector $\mathbf{v}(q)$, in the first sub-step, the processing bits of the appropriate line matrices are activated ($\mathbf{p}'$). Next, the activated line matrices are used to activate the representations of all of q's children in the data sub-vector ($\mathbf{v}(\{q', q''\})$). Lastly, these representations are conjoined with the states reachable by the symbol a, resulting in the representation of the state q in the data sub-vector ($\mathbf{v}(q)$).

This results in the activation of the two-hot representations of all possible next states (i.e., all children of q_t). In the last sub-step of the HRNN update step, these are conjoined with the representation of the current input symbol. This step is very similar to the analogous stage in Minsky's construction, with the difference that here, the non-zero entries of the vector $\mathbf{V}\mathbf{h}_t$ must cover the two-hot representations of the states with an incoming y_t-transition. This conjunction then ensures that among all the children of q_t, only the one reached by taking the y_t-transition will be encoded in $\mathbf{h}_{t+1}$. The construction just described can be summarized by the following lemma.

Lemma 4.4. *Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be a deterministic FSA. Then, Dewdney's construction results in a HRNN correctly simulating $\mathcal{A}$'s transition function, i.e, $s(\mathbf{h}_t) = q_t$ for all t.*

Proof. The proof follows the reasoning on the activation of appropriate matrices according to the transition function of the FSA outlined above. To formally prove the lemma, we would have to follow a similar set of steps to how the correctness of Minsky's construction (Lemma 3.1) was proved. We omit this for conciseness. ■

This shows that Dewdeny's construction correctly encodes the FSA in a HRNN. However, its space efficiency remains to be determined. As mentioned above, working with two-hot representations of the states means that the data sub-vector is of size $\mathcal{O}\left(|\Sigma|\sqrt{|Q|}\right)$. However, the construction also requires a number of processing dimensions in the processing sub-vector. To understand the full complexity of the construction, we have to determine the maximal number of processing bits in the HRNN. The first step to the answer is contained in the following lemma, which describes the number of line matrices required to cover an arbitrary binary matrix. It lies in the core of the efficiency of Dewdney's construction.

Lemma 4.5. *Let $\mathbf{B} \in \mathbb{B}^{D \times D}$ with N^2 elements equalling 1. Then, there exists a decomposition $\mathcal{B}$ of $\mathbf{B}$ into at most $2N$ line matrices such that $\bigvee_{\mathbf{M} \in \mathcal{B}} \mathbf{M} = \mathbf{B}$.*

Proof. Based on Dewdney [1977].[9] Define the sequence of transversals $\mathbf{T}_1, \mathbf{T}_2, \ldots$ where $\mathbf{T}_i$ is the transversal containing the maximum number of ones in the matrix $\mathbf{B}_i \stackrel{\text{def}}{=} \mathbf{B} - \bigvee_{j=1}^{i-1} \mathbf{B}_j$. The transversal containing the maximal number of ones can be found using the maximum matching algorithm. Continue this sequence until there are no more ones in $\mathbf{B}_i$. The number of ones in the matrices $\mathbf{B}_i$, $\|\mathbf{B}_i\|_1$, forms a (weakly) decreasing sequence.

If there are at most $2N$ transversals in the sequence, the lemma holds. Otherwise, we compare the functions $f(i) \stackrel{\text{def}}{=} \|\mathbf{T}_i\|_1$ and $g(i) \stackrel{\text{def}}{=} 2N - i$.

- If $f(i) > g(i)$ for all $i = 1, \ldots, N$, then $\sum_{i=1}^{N} f(i) = \sum_{i=1}^{N} \|\mathbf{T}_i\|_1 > \sum_{i=1}^{N} 2N - i = 2N^2 - \frac{1}{2}N(N+1) \geqslant N^2$. However, the transversals in the decomponsition cannot contain more ones than the original matrix.

- We conclude that for some $i \leqslant N$, $f(i) \leqslant g(i)$. Let i_0 be the first such index in $1, \ldots, N$ and $\mathcal{L}_1 \stackrel{\text{def}}{=} \{\mathbf{T}_1, \ldots, \mathbf{T}_k\}$. Since the maximum number of independent ones (in the sense that at most one appears in a single row/column) in $\mathbf{B}_{i_0-1}$ is $\|\mathbf{T}_{i_0}\|_1 \leqslant 2N - i_0$ (those are chosen by the maximum transversal $\mathbf{T}_{i_0}$). By König's theorem [Szárnyas, 2020], there is a set of at most $2N - i_0$ column or row matrices $\mathcal{L}_2 \stackrel{\text{def}}{=} \{\mathbf{L}_1, \ldots \mathbf{L}_k\}$ with $k \leqslant 2N - i_0$ which cover $\mathbf{B}_{i_0-1}$.[10] Therefore, $\mathcal{L} \stackrel{\text{def}}{=} \mathcal{L}_1 \cup \mathcal{L}_2$ constitutes a valid cover of $\mathbf{B}$ with $\leqslant N + 2N - i_0 = \mathcal{O}(N)$ matrices.

[9] The proof in Dewdney [1977] contains a mistake that affects the constant, but not the big-O complexity of the space complexity. It is fixed in this proof.

[10] Intuitively, since all ones are contained within $\leqslant 2N - i_0$ rows or columns, they can be simply covered by matrices containing those.

■

We will denote the number of matrices in the line decomposition of a matrix $\mathbf{B}$ constructed by the greedy procedure from Lemma 4.5 as $L(\mathbf{B})$. Connecting this lemma to Dewdney's construction, this shows that the number of neurons required to detect the activation of a *single set* $\mathtt{Par}(k, y)$ grows asymptotically as the square root of the number of ones in the representation matrix $\mathbf{B}(\mathtt{Par}(k, y))$—this is how many line matrices the matrix will decompose into. The size of each neuron is $2|\Sigma|s$.

This allows us to show how many neurons the entire HRNN simulating $\mathcal{A}$ has. Since we know that the data sub-vector will always have exactly $2|\Sigma|s$ cells, we characterize the number of processing cells in the following lemma.

Lemma 4.6. *Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be a deterministic FSA. Then, Dewdney's construction results in a HRNN with a hidden state of size $\mathcal{O}\left(|\Sigma||Q|^{\frac{3}{4}}\right)$.*

Proof. The number of cells in the entire processing sub-vector is simply the sum of the processing neurons of all the data components. In the worst case, a single component-activating matrix $\mathbf{B}$ requires $2L(\mathbf{B}) + 1$ neurons (2 for each transversal in the decomposition of $\mathbf{B}$ and an additional one for their disjunction). Therefore, enumerating the set of matrices $\{\mathbf{B}_{j,y,k} \mid j \in \mathbb{Z}_2, y \in \Sigma, k \in \mathbb{Z}_s\}$ with $\mathbf{B}_n$ for $n = 1, \ldots, 2|\Sigma|s$, the number of neurons required by all component-activating matrices is bounded as follows.

$$\sum_{n=1}^{2|\Sigma|s} 2L(\mathbf{B}_n) + 1 \leqslant \sum_{n=1}^{2|\Sigma|s} 2\left(2\lceil\sqrt{\|\mathbf{B}_n\|_1}\rceil\right) + 1 \stackrel{\text{def}}{=} \sum_{n=1}^{2|\Sigma|s} 4m_n + 1 \tag{23}$$

Since the matrices $\mathbf{B}_n$ contain one non-zero entry for each state-symbol pair, it holds that

$$\sum_{n=1}^{2|\Sigma|s} \|\mathbf{B}_n\|_1 \leqslant \sum_{n=1}^{2|\Sigma|s} m_n^2 = |\Sigma||Q| \tag{24}$$

Pretending that m_n can take real values, the value of Eq. (23) is maximized under the constraint from Eq. (24) when all m_n are equal with $m_n = \sqrt{2s}$. This means that

$$\sum_{n=1}^{2|\Sigma|s} 4m_n + 1 \leqslant \sum_{n=1}^{2|\Sigma|s} 4\sqrt{2s} + 1 = 8|\Sigma|s\sqrt{2s} + 1 = \mathcal{O}\left(|\Sigma||Q|^{\frac{3}{4}}\right), \tag{25}$$

finishing the proof. ■

All results stated in this section can be summarized in the following theorem.

Theorem 4.1. *Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be a deterministic FSA. Then, there exists a HRNN of size $\mathcal{O}\left(|\Sigma||Q|^{\frac{3}{4}}\right)$ correctly simulating $\mathcal{A}$.*

Proof. The existence of the RNN follows from the discussion on detectable matrices and Lemma 4.4 while the space bound is guaranteed by Lemma 4.6. ■

5 Indyk's Construction

§4 describes a construction of an HRNN of size $\mathcal{O}\left(|\Sigma||Q|^{\frac{3}{4}}\right)$ simulating an FSA. While this improves the space efficiency compared to Minsky's construction, it is not asymptotically optimal. Indyk [1995] proved that a HRNN simulating an FSA $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ over a binary alphabet $\Sigma = \mathbb{B}$ requires at least $\Omega\left(\sqrt{|Q|}\right)$ hidden dimensions. He also provided a construction that achieves this lower bound. This construction is conceptually very similar to Dewdney's in that it works by activating neurons corresponding to some form of compressed parent matrices (component-activating matrices) and then selecting the transition which matches the input symbol. Again, it additively covers these matrices with components that are easy to detect, similar to how Dewdney's construction uses line matrices. However, Indyk's construction defines component-activating matrices based on different sets of states and covers them with a different decomposition—these are the two crucial differences allowing the construction to achieve the optimal lower bound.

We first define the component-activating matrices and their role in updating the hidden state of the HRNN. In Indyk's construction, the component-activating matrices are based on *four-hot* rather than two-hot encodings of states.

Definition 5.1. *Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be an FSA, $r \stackrel{\text{def}}{=} \lceil |Q|^{\frac{1}{4}} \rceil$, and ρ a permutation of $Q = [|Q|]$.*[11] *We define the **four-hot representation** of $q \in Q$ as*

$$\phi_4(q) = (\ell_1, \ell_2, \ell_3, \ell_4) \tag{26}$$

where

$$\ell_j = \frac{\rho(q)}{r^{j-1}} \mod r. \tag{27}$$

We denote the inverse of ϕ_4 with ϕ_4^{-1} and further define for $k \in \mathbb{Z}_r$

$$\phi_4^{-1}(k, \cdot, \cdot, \cdot) \stackrel{\text{def}}{=} \{q \in Q \mid \phi_4(q)_1 = k\} \tag{28}$$

and $\phi_4^{-1}(\cdot, k, \cdot, \cdot)$, $\phi_4^{-1}(\cdot, \cdot, k, \cdot)$, and $\phi_4^{-1}(\cdot, \cdot, \cdot, k)$ analogously.

We will denote $\phi_4^{-1}(\ldots, k, \ldots)$ with k in j^{th} position (with $j \in \mathbb{Z}_4$) as $\Phi_{k,j}$. Despite using the four-hot representations, Indyk's construction still requires the two-hot representations based on ϕ_2 as before. In this case, however, they again depend on the chosen permutation ρ. This allows us to define the component-activating matrices as follows.

Definition 5.2. *A **component-activating matrix** in Indyk's construction is the representation matrix $\mathbf{B}(\mathsf{Par}(\Phi_{k,j}, y))$ for some $k \in \mathbb{Z}_r$, $j \in \mathbb{Z}_4$, and $y \in \Sigma$.*

For efficient detection, the component-activating matrices are covered by so-called non-decreasing matrices.

Definition 5.3. *We say that $\mathbf{B} \in \mathbb{B}^{D \times D}$ is **non-decreasing** if there exists a* non-decreasing *(partial) function $f : \mathbb{Z}_D \to \mathbb{Z}_D$ (from columns to rows) such that*

$$\mathbf{B}_{ij} = 1 \iff f(j) = i \tag{29}$$

***and**, if f is defined for some $j \in \mathbb{Z}_D$, it is also defined for all $j' \geqslant j$.*

[11] The exact form of ρ will be important later. For now, one can think of ρ as the identity function.

Example 5.1. *An example of a non-decreasing matrix is*

$$\mathbf{B} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{30}$$

The function f defining the non-decreasing matrix $\mathbf{B}$ is $f = \begin{pmatrix} 0 & 1 & 2 & 3 \\ \varnothing & 0 & 1 & 1 \end{pmatrix}$, where $\varnothing$ denotes that the function is not defined.

Again, clearly, any matrix $\mathbf{B} \in \mathbb{B}^{D \times D}$ can be (non-uniquely) decomposed into at most D non-decreasing matrices. Moreover, non-decreasing matrices are detectable.

Lemma 5.1. *Non-decreasing matrices are detectable.*

Proof. Let $\mathbf{B} \in \mathbb{B}^{D \times D}$ be a non-decreasing matrix defined by the partial function f. Divide the domain of f into the set of intervals in which the function is constant, with $I(j)$ denoting the interval of $j \in \mathbb{Z}_{r^2}$ for j such that $f(j)$ is defined. Then, it is easy to see that $B_{ij} = 1 \iff i = f(j)$, meaning that by defining the parameters $\mathbf{w}$ and b as

$$w_{f(j)} \stackrel{\text{def}}{=} r^2 - I(j) \tag{31}$$

$$w_{r^2+j} \stackrel{\text{def}}{=} I(j) \tag{32}$$

$$b \stackrel{\text{def}}{=} -r^2 \tag{33}$$

and other elements as 0, we get that

$$B_{ij} = 1 \iff i = f(j) \iff w_i + w_j + b = 0. \tag{34}$$

Compared to earlier, where component-activating matrices were detected by testing an *inequality*, detecting a non-decreasing matrix requires testing an *equality*. Since all terms in the equality are integers, testing the equality can be performed with the Heaviside activation function by conjoining two neurons; one testing the inequality $w_i + w_j + b - 1 < 0$ and another one testing the inequality $w_i + w_j + b + 1 > 0$. Both can individually be performed by a single neuron and then conjoined by an additional one. ■

With this, the high-level idea of Indyk's construction is outlined in Fig. 5. After constructing the component-activating matrices based on ϕ_4 and decomposing them into non-decreasing matrices, the rest of Indyk's construction is very similar to Dewdney's construction, although the full update step of the HRNN requires some additional processing. To test the equality needed to detect non-decreasing matrices in the decomposition, Eq. (34), the four-hot representations are first converted into two-hot ones. This can be done by a simple conjunction of the first two and the last two components of the four-hot representation. Then, the activations of the non-decreasing matrices can be computed and disjoined into the representations of the component-activating matrices. These form the $4|\Sigma|r$ components of the data sub-vector of the HRNN hidden state. They contain the activations of all possible next states, i.e., the children of the current state of $\mathcal{A}$. These are then conjoined with the representation of the current input symbol in the same way as in Dewdney's construction but adapted to the four-hot representations of the states. The process is thus very similar to the phases of Dewdeney's construction illustrated in Fig. 4b.

Indyk's construction can be summarized by the following lemma.

Lemma 5.2. *Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be a deterministic FSA. Then, Indyk's construction results in a HRNN correctly simulating $\mathcal{A}$'s transition function, i.e, $s(\mathbf{h}_t) = q_t$ for all t.*

Proof. Again, the proof follows the reasoning outlined above, and, to formally prove it is correct, we would have to follow a similar set of steps to how the correctness of Minsky's construction (Lemma 3.1) was proved. We omit this for conciseness. ■

The only remaining thing to show is that Indyk's construction achieves the theoretically optimal lower bound on the size of the HRNN simulating a deterministic FSA. All previous steps of the construction were valid no matter the chosen permutation ρ. The permutation, however, matters for space efficiency: Intuitively, it determines how efficiently one can decompose the resulting component-activating matrices (which depend on the permutation) into non-decreasing matrices in the sense of how many non-decreasing matrices are required to cover it. Indyk, therefore, proved that there always exists, with non-zero probability, a permutation in which the decomposition across all states is efficient enough to achieve the minimum number of neurons required. This is formalized by the following lemma.

Lemma 5.3. *Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be a deterministic FSA. There exists a permutation of Q such that Indyk's construction results in a HRNN of size $\mathcal{O}\left(|\Sigma|\sqrt{|Q|}\right)$.*

Proof. The proof can be found in Indyk [1995, Lemma 6]. ■

This concludes our presentation of Indyk's construction. All results stated in this section can be summarized by the following theorem.

Theorem 5.1. *Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be a deterministic FSA. There exists a HRNN of size $\mathcal{O}\left(|\Sigma|\sqrt{|Q|}\right)$ correctly simulating $\mathcal{A}$.*

Proof. Again, the proof follows from the fact that Indyk's construction correctly simulates the FSA (cf. Lemma 5.2) while the space bound is guaranteed by Lemma 5.3. ■

6 Conclusion

We presented three classical constructions of an RNN with the Heaviside activation function simulating a finite-state automaton. Given an FSA with N states, Minsky's [1954] construction defines an RNN with neurons, Dewdney's [1977] construction an RNN with $\mathcal{O}\left(N^{\frac{3}{4}}\right)$, and Indyk's [1995] construction an RNN with $\mathcal{O}\left(\sqrt{N}\right)$ neurons. The latter is *optimal* in the general case—there exist FSAs which cannot be simulate correctly by an RNN with fewer neurons. The question of how these constructions translate to the case of *weighted* FSAs was addressed by Svete and Cotterell [2023], who show that Minsky's construction is in some ways optimal in the weighted case, since an RNN simulating a weighted FSA might require $\Omega\left(N\right)$ neurons. We hope the review sparks further research into relationship between formal models of computation and modern neural architectures, helping us understand the complex architectures in terms of well-understood formalisms.

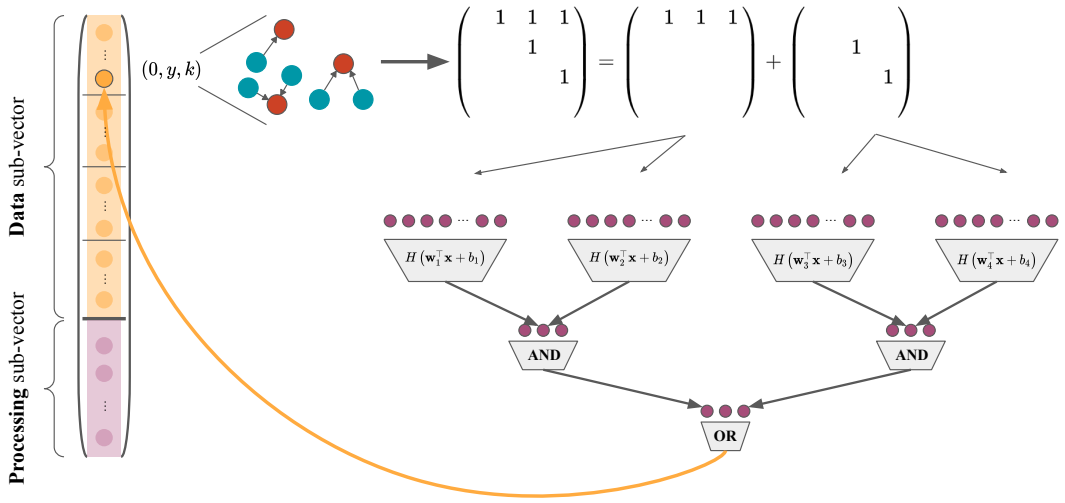


Figure 5: High-level overview of Indyk's construction. The highlighted orange neuron in the representation of the state from the data sub-vector corresponds to the activation of one of the components of the red states (which have in common that their 0^{th} component of $\phi_4(q)$ is the same). The matrix corresponding to the disjunction of the representations of their *y-parents* (blue states) is decomposed into two non-decreasing matrices. The non-zero elements of both can be detected by a conjunction of two neurons; here, $f_1 = \begin{pmatrix} 0 & 1 & 2 & 3 \\ \varnothing & 0 & 0 & 0 \end{pmatrix}$ and $f_2 = \begin{pmatrix} 0 & 1 & 2 & 3 \\ \varnothing & \varnothing & 1 & 2 \end{pmatrix}$, meaning that $\mathbf{w}_1 = \begin{pmatrix} 3 & 0 & 0 & 0 & | & 0 & 1 & 1 & 1 \end{pmatrix}$, $\mathbf{w}_2 = \begin{pmatrix} 0 & 3 & 2 & 0 & | & 0 & 0 & 1 & 2 \end{pmatrix}$, and $b_1 = b_2 = 4$. Those activations are then disjoined to result in the activation in the orange neuron. The purple neurons in the processing sub-vector are composed of the neurons in the networks implementing the detection of line matrices and their conjunctions and disjunctions (also shown in purple). Note that even if the second matrix were not non-decreasing in itself (i.e., the columns of the two ones would be flipped), one could still transform it into a non-decreasing matrix by permuting the columns and permuting the corresponding neurons.

References

Noga Alon, A. K. Dewdney, and Teunis J. Ott. Efficient simulation of finite automata by neural nets. *Journal of the ACM*, 38:495–514, 1991. URL https://api.semanticscholar.org/CorpusID:14769260.

David Chiang and Peter Cholak. Overcoming a theoretical limitation of self-attention. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7654–7664, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.527. URL https://aclanthology.org/2022.acl-long.527.

David Chiang, Peter Cholak, and Anand Pillay. Tighter bounds on the expressivity of transformer encoders. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 5544–5562. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/chiang23a.html.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-4012. URL https://aclanthology.org/W14-4012.

A. K. Dewdney. Threshold matrices and the state assignment problem for neural nets. In *Proceedings of the 8th SouthEastern Conference on Combinatorics, Graph Theory and Computing*, pages 227–245, Baton Rouge, La, USA, 1977. URL https://ci.nii.ac.jp/ncid/BA16515546?l=en.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. doi: https://doi.org/10.1207/s15516709cog1402_1. URL https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1.

Michael Hahn. Theoretical Limitations of Self-Attention in Neural Sequence Models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 01 2020. ISSN 2307-387X. doi: 10.1162/tacl_a_00306. URL https://doi.org/10.1162/tacl_a_00306.

Yiding Hao, William Merrill, Dana Angluin, Robert Frank, Noah Amsel, Andrew Benz, and Simon Mendelsohn. Context-free transductions with neural stacks. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 306–315, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5433. URL https://aclanthology.org/W18-5433.

John Hewitt, Michael Hahn, Surya Ganguli, Percy Liang, and Christopher D. Manning. RNNs can generate bounded hierarchical languages with optimal memory. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1978–2010, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.156. URL https://aclanthology.org/2020.emnlp-main.156.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 11 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

P. Indyk. Optimal simulation of automata by neural nets. In Ernst W. Mayr and Claude Puech, editors, *STACS 95*, pages 337–348, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg. ISBN 978-3-540-49175-0. URL https://dl.acm.org/doi/10.1145/103516.103523.

S. C. Kleene. Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies. (AM-34), Volume 34*, pages 3–42. Princeton University Press, Princeton, 1956. ISBN 9781400882618. doi: doi:10.1515/9781400882618-002. URL https://doi.org/10.1515/9781400882618-002.

Samuel A. Korsky and Robert C. Berwick. On the computational power of RNNs. *CoRR*, abs/1906.06349, 2019. URL http://arxiv.org/abs/1906.06349.

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943. ISSN 1522-9602. doi: 10.1007/BF02478259. URL https://doi.org/10.1007/BF02478259.

William Merrill. Sequential neural networks as automata. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 1–13, Florence, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-3901. URL https://www.aclweb.org/anthology/W19-3901.

William Merrill and Nikolaos Tsilivis. Extracting finite automata from RNNs using state merging. *arXiv preprint arXiv:2201.12451*, 2022. URL https://arxiv.org/abs/2201.12451.

William Merrill, Gail Weiss, Yoav Goldberg, Roy Schwartz, Noah A. Smith, and Eran Yahav. A formal hierarchy of RNN architectures. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 443–459, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.43. URL https://aclanthology.org/2020.acl-main.43.

William Merrill, Ashish Sabharwal, and Noah A. Smith. Saturated transformers are constant-depth threshold circuits. *Transactions of the Association for Computational Linguistics*, 10:843–856, 2022. doi: 10.1162/tacl_a_00493. URL https://aclanthology.org/2022.tacl-1.49.

Marvin Lee Minsky. *Neural Nets and the Brain Model Problem*. PhD thesis, Princeton University, 1954. URL https://www.proquest.com/docview/301998727?parentSessionId=tt6FtxLC54LAeBn4iwlPCmf7YomIxumldfOoWkNvHtM%3D.

Franz Nowak, Anej Svete, Li Du, and Ryan Cotterell. On the representational capacity of recurrent neural language models. *arXiv preprint arXiv:2310.12942*, 2023. URL https://arxiv.org/abs/2310.12942.

Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 440–449, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 089791497X. doi: 10.1145/130385.130432. URL https://doi.org/10.1145/130385.130432.

Anej Svete and Ryan Cotterell. Recurrent neural language models as probabilistic finite-state automata. *arXiv preprint arXiv:2310.05161*, 2023. URL https://arxiv.org/abs/2310.05161.

Gábor Szárnyas. Graphs and matrices: A translation of "graphok és matrixok" by Dénes König (1931). *arXiv preprint arXiv:2009.03780*, 2020. URL https://arxiv.org/abs/2009.03780.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Gail Weiss, Yoav Goldberg, and Eran Yahav. On the practical computational power of finite precision RNNs for language recognition. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 740–745, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2117. URL https://aclanthology.org/P18-2117.