


Lower Bounds on the Expressivity of Recurrent Neural Language Models

Anej Svete* Franz Nowak* Anisha Mohamed Sahabdeen Ryan Cotterell
{asvete, fnowak, amohame, ryan.cotterell}@ethz.ch

ETH zürich

Abstract

The recent successes and spread of large neural language models (LMs) call for a thorough understanding of their computational ability. Describing LMs’ computational abilities through their representational capacity is a lively area of research. However, investigation into the representational capacity of neural LMs has predominantly focused on their ability to *recognize* formal languages. For example, recurrent neural networks (RNN) with Heaviside activation functions are tightly linked to regular languages, i.e., languages defined by finite-state automata (FSAs). Such results, however, fall short of describing the capabilities of RNN *language models* (LMs), which are definitionally *distributions* over strings. We take a fresh look at the representational capacity of RNN LMs by connecting them to *probabilistic* FSAs and demonstrate that RNN LMs with linearly bounded precision can express arbitrary regular LMs.

 <https://github.com/rycolab/nondeterministic-rnns>

1 Introduction

Neural language models (LMs) excel at many NLP tasks, e.g., machine translation, conversational AI, text classification, and natural language inference (Cho et al., 2014b; Albalak et al., 2022; Sun et al., 2023; Schick and Schütze, 2021). Their strong empirical performance suggests the need for theory to explain what LMs can and cannot do in a formal sense. Such theory should serve to better understand as well as alleviate the practical limitations of neural LMs. However, the most common paradigm to perform such an analysis of neural LMs revolves around proving which formal languages common LM architectures can express (Ackerman and Cybenko, 2020; Icard, 2020; Merrill et al., 2020; Pérez et al., 2021, *inter alia*), i.e., to characterize the representational capacity of neural LMs with tools from the theory of computation.

Formally, a language model is a probability distribution over Σ^* , the set of all strings from

some alphabet Σ .¹ We say that two language models are weakly equivalent if they express the same distribution over strings. PFSA (both deterministic and non-deterministic) and recurrent neural LMs, e.g., those derived from Elman recurrent neural networks (RNNs; Elman, 1990), however, are instances of *families* of language models, i.e., sets of distributions over strings that can be represented under the specific formalism. For instance, if one changes the parameters of a PFSA, one arrives at a different language model. Therefore, going beyond the formal comparison of individual LMs, we may also desire to make formal statements about families of models. One natural way to construct an (upper) bound on the representational capacity of an entire family of language models \mathcal{F}_1 , e.g., those expressible by RNNs, is to find another family of language models \mathcal{F}_2 and show that, for every language model $q \in \mathcal{F}_2$, there exists a weakly equivalent $p \in \mathcal{F}_1$. Our goal in this paper is to show that, for every non-deterministic PFSA, there exists a weakly equivalent RNN LM.

The approach to studying language models presented in the previous paragraph differs fundamentally from most work that uses formal language theory to study the representational capacity of language models. Indeed, most papers try to relate language models to language *acceptors* (Siegelmann and Sontag, 1992; Merrill et al., 2020; Pérez et al., 2021; Merrill et al., 2022, *inter alia*). E.g., finite-state automata have been linked to RNNs extensively before (Alon et al., 1991; Indyk, 1995; Merrill, 2019; Merrill and Tsilivis, 2022). However, acceptors are fundamentally different objects than LMs, as they define *sets* of strings rather than distributions over them. Thus, analyzing acceptors, does not address our desire to understand recurrent neural LMs as probability distributions over Σ^* .

In previous work, Svete and Cotterell (2023) demonstrate a particularly close relationship between recurrent neural language models and *deterministic* PFSA. However, their theory does not extend to the family of non-deterministic

*Equal contribution.

¹An alphabet is a finite, non-empty set of symbols.

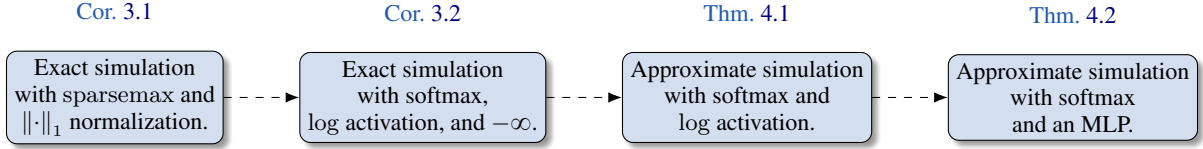


Figure 1: A roadmap of the results in this paper describing the representational capacity of Elman LMs. We start with an intuitive result showing how a sparsemax RNN with an $\|\cdot\|_1$ -normalized hidden state can exactly simulate a regular LM. We then bring these modeling choices closer to practical implementations, leading us to the final result, which shows how sparsemax Elman LMs with an MLP-transformed hidden state can approximate arbitrary regular LMs.

PFSAs because it is a basic fact of probabilistic formal language theory that not all PFSAs can be determinized (Allauzen and Mohri, 2003). To date, no corresponding result exists for the non-deterministic case. To fill this void, we give a construction that shows real-time ReLU-activated Elman RNNs (Elman LMs) with linearly bounded precision can exactly simulate or arbitrary approximate *non*-deterministic PFSAs depending on the components employed in the network. Concretely, we show that (1) Elman LMs with the sparsemax function (Martins and Astudillo, 2016) can represent any regular LM exactly; and, (2) Elman LMs with the softmax projection function and a nonlinear output function can approximate a regular LM arbitrarily well. The roadmap of the paper is presented in Fig. 1.

2 Preliminaries

We begin by introducing some preliminaries.

2.1 Language Modeling

Most modern neural LMs define the probability $p(\mathbf{y})$ of a string $\mathbf{y} \in \Sigma^*$ as a product of conditional probability distributions p , i.e.,

$$p(\mathbf{y}) \stackrel{\text{def}}{=} p(\text{EOS} \mid \mathbf{y}) \prod_{t=1}^{|\mathbf{y}|} p(y_t \mid \mathbf{y}_{<t}), \quad (1)$$

where $\text{EOS} \notin \Sigma$ is a special end-of-sequence symbol. A language model expressed as in Eq. (1) is called **autoregressive**. Let $\bar{\Sigma} \stackrel{\text{def}}{=} \Sigma \cup \{\text{EOS}\}$. Then, each $p(\bar{y}_t \mid \mathbf{y}_{<t})$ is a distribution over $\bar{\Sigma}$. Additionally, we may also consider ε -augmented language models where each p is a distribution over $\bar{\Sigma} \cup \{\varepsilon\}$ where $\varepsilon \notin \Sigma$ is a special symbol not in the alphabet that represents the empty string. This allows the language model to perform computations *longer* than the length of the string it generates. An autoregressive language model is called **real-time** if each p is *only* a distribution over $\bar{\Sigma}$, i.e., not over $\bar{\Sigma} \cup \{\varepsilon\}$. Throughout this paper, we construct and

investigate the expressivity of real-time autoregressive language models based on RNNs.

At a high level, we are interested in encoding real-time LMs as RNN LMs. To do so, we need a notion of equivalence between language models. In this paper, we will work with weak equivalence.

Definition 2.1. *Two LMs p and q over Σ^* are **weakly equivalent** if $p(\mathbf{y}) = q(\mathbf{y})$ for all $\mathbf{y} \in \Sigma^*$.*

2.2 Regular Language Models

Probabilistic finite-state automata are a well-understood real-time computational model.

Definition 2.2. *A **probabilistic finite-state automaton** (PFSA) is a 5-tuple $(\Sigma, Q, \tau, \lambda, \rho)$ where Σ is an alphabet, Q is a finite set of states, $\tau \subseteq Q \times \Sigma \times \mathbb{Q}_{\geq 0} \times Q$ is a finite set of weighted transitions where we write transitions $(q, y, w, q') \in \tau$ as $q \xrightarrow{y/w} q'$,² and $\lambda, \rho: Q \rightarrow \mathbb{Q}_{\geq 0}$ are functions that assign each state its initial and final weight, respectively. Moreover, for all states $q \in Q$, τ , λ and ρ satisfy $\sum_{q \in Q} \lambda(q) = 1$, and $\sum_{q \xrightarrow{y/w} q' \in \tau} w + \rho(q) = 1$.*

We next define some basic concepts. A PFSA $\mathcal{A} = (\Sigma, Q, \tau, \lambda, \rho)$ is **deterministic** if $|\{q \mid \lambda(q) > 0\}| = 1$ and, for every $q \in Q, y \in \Sigma$, there is at most one $q' \in Q$ such that $q \xrightarrow{y/w} q' \in \tau$ with $w > 0$.³ Any state q where $\lambda(q) > 0$ is called an **initial state**, and if $\rho(q) > 0$, it is called a **final state**. A **path p** of length N is a sequence of subsequent transitions in \mathcal{A} , denoted as

$$q_1 \xrightarrow{y_1/w_1} q_2 \xrightarrow{y_2/w_2} q_3 \cdots q_N \xrightarrow{y_N/w_N} q_{N+1}. \quad (2)$$

The **yield** of a path is $\mathbf{s}(p) \stackrel{\text{def}}{=} y_1 \dots y_N$. The **prefix weight** \tilde{w} of a path p is the product of

²We further assume a (q, y, q') triple appears in at most one element of τ .

³In this paper, we do *not* distinguish between a transition for a given symbol with weight $w = 0$ and the absence of a transition on that symbol. That is, we assume there always exists a transition $q \xrightarrow{y/w} q' \in \tau$ for any $q, q' \in Q$ and $y \in \Sigma$, albeit possibly with $w = 0$. Such a choice turns out to be useful in our technical exposition; see Alg. 1.

the transition and initial weights, whereas the **weight** of a path additionally has the final weight multiplied in. In symbols, this means

$$\tilde{w}(\mathbf{p}) \stackrel{\text{def}}{=} \prod_{n=0}^N w_n, \quad (3) \quad w(\mathbf{p}) \stackrel{\text{def}}{=} \prod_{n=0}^{N+1} w_n, \quad (4)$$

with $w_0 \stackrel{\text{def}}{=} \lambda(q_1)$ and $w_{N+1} \stackrel{\text{def}}{=} \rho(q_{N+1})$. We write $P(\mathcal{A})$ for the set of all paths in \mathcal{A} and we write $P(\mathcal{A}, \mathbf{y})$ for the set of all paths in \mathcal{A} with yield \mathbf{y} . The sum of weights of all paths that yield a certain string $\mathbf{y} \in \Sigma^*$ is called the **stringsum**, given in the notation below

$$\mathcal{A}(\mathbf{y}) \stackrel{\text{def}}{=} \sum_{\mathbf{p} \in P(\mathcal{A}, \mathbf{y})} w(\mathbf{p}). \quad (5)$$

The stringsum gives the probability of the string \mathbf{y} . We say a state $q \in Q$ is **accessible** if there exists a path with non-zero weight from an initial state to q . A state $q \in Q$ is **co-accessible** if there exists a path with non-zero weight from q to a final state. An automaton in which all states are accessible and co-accessible is called **trim**.

PFSAs as Autoregressive Language Models.

We now show how a PFSA \mathcal{A} induces a LM $p_{\mathcal{A}}$ over strings $\mathbf{y} \in \Sigma^*$. By Def. 2.2, the weights of all available transitions of a PFSA in state q , together with the final weight, define a probability distribution over the next action, i.e., taking a transition or halting. We can translate this into a distribution over $\bar{\Sigma}$ where EOS corresponds to halting. In the following, we use the notation $\bar{y}_t \in \bar{\Sigma}$ and $y_t \in \Sigma$ to distinguish between symbols that could be EOS and those that cannot, respectively. Specifically, we can define the probability over $\bar{\Sigma}$ as follows

$$p_{\mathcal{A}}(\bar{y}_t | q, \mathbf{y}_{<t}) = \begin{cases} \sum_{q' \xrightarrow{\bar{y}_t/w} q'} w & \text{if } \bar{y}_t \in \Sigma \\ \rho(q) & \text{if } \bar{y}_t = \text{EOS}. \end{cases} \quad (6)$$

Moreover, in a PFSA, the probability of \bar{y}_t is conditionally independent of $\mathbf{y}_{<t}$ given the state q , i.e.,

$$p_{\mathcal{A}}(\bar{y}_t | q, \mathbf{y}_{<t}) = p_{\mathcal{A}}(\bar{y}_t | q). \quad (7)$$

Finally, using the law of total probability, we can define an autoregressive language model as

$$p_{\mathcal{A}}(\bar{y}_t | \mathbf{y}_{<t}) \stackrel{\text{def}}{=} \sum_{q \in Q} p_{\mathcal{A}}(\bar{y}_t | q) p_{\mathcal{A}}(q | \mathbf{y}_{<t}) \\ = \sum_{q \in Q} p_{\mathcal{A}}(\bar{y}_t | q) \frac{p_{\mathcal{A}}(q, \mathbf{y}_{<t})}{p_{\mathcal{A}}(\mathbf{y}_{<t})} \quad (8a)$$

$$= \sum_{q \in Q} p_{\mathcal{A}}(\bar{y}_t | q) \frac{p_{\mathcal{A}}(q, \mathbf{y}_{<t})}{\sum_{q' \in Q} p_{\mathcal{A}}(q', \mathbf{y}_{<t})}, \quad (8b)$$

where $p_{\mathcal{A}}(q, \mathbf{y}_{<t})$ can be written as

$$p_{\mathcal{A}}(q, \mathbf{y}_{<t}) = \left(\overset{\rightarrow}{\lambda} \prod_{s=1}^t \mathbf{T}^{(y_s)} \right)_q, \quad (9)$$

where $\overset{\rightarrow}{\lambda}$ and $\mathbf{T}^{(y_s)}$ refer to the vectorized initial function and symbol-specific transition matrices of the PFSA \mathcal{A} , respectively; see App. A for a full specification. Eq. (8a) shows that a PFSA induces a language model $p_{\mathcal{A}}$ as in Eq. (1) through the conditional probabilities defined above.

Definition 2.3. An LM p is a **regular language model** if there exists a PFSA \mathcal{A} whose induced language model $p_{\mathcal{A}}$ is weakly equivalent to $p_{\mathcal{A}}$.

See Fig. 2 for examples of PFSAs that induce regular language models over $\Sigma = \{a, b\}$.

PFSAs and FSAs. Although PFSAs share many properties with unweighted (boolean-weighted) finite-state automata, one important difference relates to determinization. In the unweighted case, the class of deterministic and non-deterministic FSAs are equivalent, i.e., any non-deterministic FSA has an equivalent deterministic FSA that accepts the same language. This result, however, does not hold for PFSAs: There exist PFSAs that admit no deterministic equivalent (Mohri, 1997; Buchsbaum et al., 2000), meaning that non-deterministic PFSAs are strictly more expressive than deterministic ones. For example, the PFSA in Fig. 2b is non-deterministic and does not admit a deterministic equivalent, since $p(ab^n)$ cannot be expressed as a single term for arbitrary $n \in \mathbb{N}_{\geq 0}$.

2.3 Recurrent Neural Language Models

Recurrent neural LMs are LMs whose conditional probabilities are given by a recurrent neural network. Throughout 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

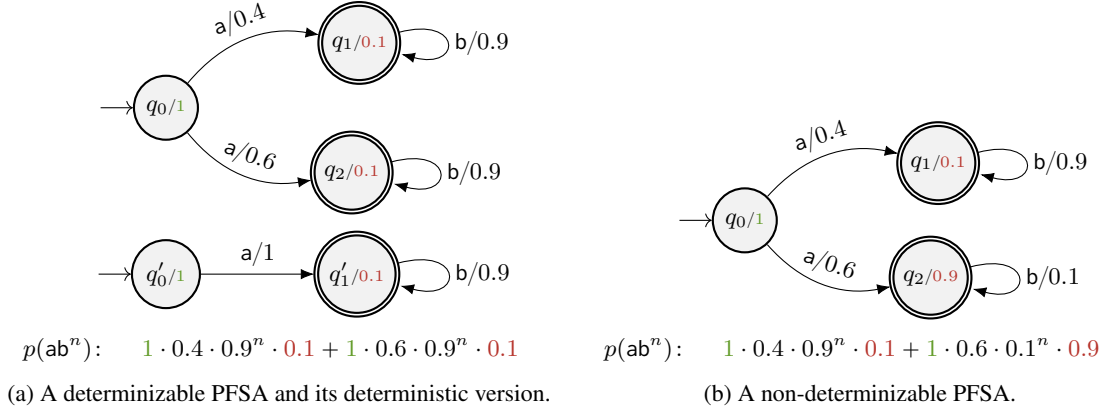


Figure 2: Examples of PFSA inducing probability distributions over $\{a, b\}^*$. Numbers in green signify initial weights, and numbers in red signify final weights.

(LSTM; Hochreiter and Schmidhuber, 1997) and gated recurrent units (GRUs; Cho et al., 2014a).

Definition 2.4. An *Elman RNN* (ERNN) $\mathcal{R} = (\mathbb{Q}^D, \Sigma, \sigma, \mathbf{U}, \mathbf{V}, \mathbf{b}, \boldsymbol{\eta})$ is an RNN with the following hidden state recurrence:

$$\mathbf{h}_0 = \boldsymbol{\eta} \quad (t = 0) \quad (10a)$$

$$\mathbf{h}_t = \boldsymbol{\alpha}(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{V}\mathbf{r}(y_t) + \mathbf{b}) \quad (t > 0), \quad (10b)$$

where $\mathbf{h}_t \in \mathbb{Q}^D$ is the hidden state vector⁴ at time step t , $\boldsymbol{\eta} \in \mathbb{Q}^D$ is an initialization parameter, $y_t \in \Sigma$ is the input symbol at time step t , $\mathbf{r}: \Sigma \rightarrow \mathbb{Q}^R$ is a symbol representation function, $\mathbf{U} \in \mathbb{Q}^{D \times D}$ and $\mathbf{V} \in \mathbb{Q}^{D \times R}$ are parameter matrices, $\mathbf{b} \in \mathbb{Q}^D$ is a bias vector, and $\boldsymbol{\alpha}: \mathbb{Q}^D \rightarrow \mathbb{Q}^D$ is an element-wise non-linear activation function.

Because \mathbf{h}_t hides the string that was consumed by the Elman RNN, we also use the evocative notation $\mathbf{h}(\mathbf{y})$ to denote the result of the application of Eq. (10b) over the string $\mathbf{y} = y_1 \cdots y_t$. Common examples of $\boldsymbol{\alpha}$ include the element-wise application of ReLU, i.e., $\text{ReLU}(x) \stackrel{\text{def}}{=} \max(0, x)$, the Heaviside function $H(x) \stackrel{\text{def}}{=} \mathbb{1}\{x > 0\}$ and the sigmoid function $\sigma(x) \stackrel{\text{def}}{=} \frac{1}{1 + \exp(-x)}$. The ReLU function is the most common choice of activation in modern deep learning and the focus of our analysis in this paper (Goodfellow et al., 2016). To define an autoregressive language model, an Elman RNN constructs a distribution over the next symbol in Σ by transforming the hidden state using some function $\mathbf{F}: \mathbb{R}^D \rightarrow \mathbb{R}^{|\Sigma|}$.

Definition 2.5. Let \mathcal{R} be an ERNN and $\mathbf{F}: \mathbb{R}^D \rightarrow \mathbb{R}^{|\Sigma|}$ a differentiable function. An *Elman LM* is an LM whose conditional distributions are defined by projecting $\mathbf{F}(\mathbf{h}_{t-1})$ onto the probability simplex $\Delta^{|\Sigma|-1}$ using a projection function

⁴Throughout this paper all vectors are column vectors.

$$\pi: \mathbb{R}^{|\Sigma|} \rightarrow \Delta^{|\Sigma|-1}.$$

$$p_{\mathcal{R}}(\bar{y}_t | \mathbf{y}_{<t}) \stackrel{\text{def}}{=} \pi(\mathbf{F}(\mathbf{h}_{t-1}))_{\bar{y}_t}. \quad (11)$$

We call \mathbf{F} an *output function*.

The vector $\mathbf{F}(\mathbf{h}_{t-1}) \in \mathbb{R}^{|\Sigma|}$ thus represents a vector of $|\Sigma|$ values that are later projected onto $\Delta^{|\Sigma|-1}$ to define $p_{\mathcal{R}}(\bar{y}_t | \mathbf{y}_{<t})$. The most common choice for the projection function π is the **softmax** defined for $\mathbf{x} \in \mathbb{R}^N$, $N \in \mathbb{N}$, and $n \in [N]$ as

$$\sigma(\mathbf{x})_n \stackrel{\text{def}}{=} \frac{\exp(\lambda x_n)}{\sum_{n'=1}^N \exp(\lambda x_{n'})}, \quad (12)$$

where $\lambda \in \mathbb{R}$ is called the **inverse temperature** parameter. Softmax may be viewed as the solution to the following convex optimization problem:

$$\boldsymbol{\sigma}(\mathbf{x}) = \underset{\mathbf{z} \in \Delta^{N-1}}{\text{argmax}} \mathbf{z}^\top \mathbf{x} + H(\mathbf{z}), \quad (13)$$

where $H(\mathbf{z}) \stackrel{\text{def}}{=} -\sum_{n=1}^N z_n \log z_n$ is the Shannon entropy.⁵ An important limitation of the softmax is that it implies the LM has full support, i.e., an Elman LM with a softmax projection assigns positive probability to all strings in Σ^* . To construct Elman LMs without full support, we also consider the **sparsemax** (Martins and Astudillo, 2016) as our projection function:

$$\text{sparsemax}(\mathbf{x}) \stackrel{\text{def}}{=} \underset{\mathbf{z} \in \Delta^{N-1}}{\text{argmin}} \|\mathbf{z} - \mathbf{x}\|_2^2. \quad (14)$$

Note that this, too, is a convex optimization problem. In contrast to the softmax function, sparsemax is the **identity** on Δ^{N-1} , i.e., we have $\text{sparsemax}(\mathbf{x}) = \mathbf{x}$ for $\mathbf{x} \in \Delta^{N-1}$.

⁵See Boyd and Vandenberghe (2004, Ex. 25) for a derivation.

Common output functions. We now outline three special cases of \mathbf{F} which will be useful in our exposition.

- **Affine output Elman LMs.** A common way to implement \mathbf{F} is as an affine transformation. In that case, \mathbf{F} takes the form of $\mathbf{F}(\mathbf{h}) \stackrel{\text{def}}{=} \mathbf{E}\mathbf{h} + \mathbf{d}$ for some $\mathbf{E} \in \mathbb{R}^{|\Sigma| \times D}$, $\mathbf{d} \in \mathbb{R}^{|\Sigma|}$. We call \mathbf{E} the **output matrix**.
- **Non-linear output Elman LMs.** In this case, \mathbf{F} is an arbitrary non-linear function.
- **Deep output Elman LMs.** \mathbf{F} is often implemented as a multi-layer perceptron in practice (Pascanu et al., 2014). In that case, we call \mathbf{F} a deep output Elman LM.

To concretely discuss the deep output Elman LMs, we now give the definition of a multi-layer perceptron, a common non-linear model we will deploy as the output function of the Elman RNN. Our definition is precise since we will later require an application of a universal approximation theorem.

Definition 2.6. A *multi-layer perceptron (MLP)* $\mathbf{F}: \mathbb{R}^N \rightarrow \mathbb{R}^M$ is a function defined as the composition of elementary functions $\mathbf{f}_1, \dots, \mathbf{f}_L$

$$\mathbf{F}(\mathbf{x}) \stackrel{\text{def}}{=} \mathbf{f}_L \circ \mathbf{f}_{L-1} \circ \dots \circ \mathbf{f}_1(\mathbf{x}), \quad (15)$$

where each function \mathbf{f}_ℓ for $\ell \in [L]$ is defined as

$$\mathbf{f}_\ell(\mathbf{x}) \stackrel{\text{def}}{=} \beta(\mathbf{W}_\ell \mathbf{x} + \mathbf{b}_\ell) \quad \ell \in [L-1] \quad (16a)$$

$$\mathbf{f}_L(\mathbf{x}) \stackrel{\text{def}}{=} \mathbf{W}_L \mathbf{x} + \mathbf{b}_L, \quad (16b)$$

where $\mathbf{W}_\ell \in \mathbb{R}^{N_\ell \times M_\ell}$ is a weight matrix with dimensions N_ℓ and M_ℓ specific to layer ℓ , $\mathbf{b}_\ell \in \mathbb{R}^{M_\ell}$ is a bias vector, and β is an element-wise non-linear activation function. The function \mathbf{f}_1 is called the **input layer**, the function \mathbf{f}_L is called the **output layer**, and the function \mathbf{f}_ℓ for $\ell = 2, \dots, L-1$ are called **hidden layers**.⁶

Besides their common application in neural LMs, MLPs with various activation functions also possess well-understood approximation abilities (Cybenko, 1989; Funahashi, 1989; Hornik et al., 1989; Pinkus, 1999). This makes their application to our goal—approximating language models—natural. In our constructions, we concretely focus on the result by Pinkus (1999) due to its generality. We

⁶Note that we refer to MLPs by the number of hidden layers, e.g., a one-layer-MLP is an MLP with one *hidden* layer.

restate it below in our own words, slightly adapting it to our use case:⁷

Theorem 2.1 (Pinkus (1999, Theorem 3.1)). *For any continuous, non-polynomial element-wise activation function β , any continuous function $\mathbf{G}: \mathbb{R}^N \rightarrow \mathbb{R}^M$, any compact subset $\mathcal{K} \subset \mathbb{R}^N$, and any $\epsilon > 0$, there exists a single-layer MLP $\mathbf{F}: \mathbb{R}^N \rightarrow \mathbb{R}^M$ with activation function β such that $\max_{\mathbf{x} \in \mathcal{K}} \|\mathbf{G}(\mathbf{x}) - \mathbf{F}(\mathbf{x})\|_\infty < \epsilon$.*

Bounded Precision. We assume the hidden states \mathbf{h}_t to be rational vectors. An important consideration in processing strings $\mathbf{y} \in \Sigma^*$ is then the number of bits required to represent the entries in \mathbf{h}_t and how the number of bits scales with the length of the string, $|\mathbf{y}|$. This motivates the definition of precision.

Definition 2.7. The *precision* $\psi(\mathbf{y})$ of an Elman RNN is the number of bits required to represent the entries of $\mathbf{h}(\mathbf{y})$:

$$\psi(\mathbf{y}) \stackrel{\text{def}}{=} \max_{d \in [D]} \min_{\substack{p, q \in \mathbb{N}, \\ \frac{p}{q} = \mathbf{h}(\mathbf{y})_d}} \lceil \log_2 p \rceil + \lceil \log_2 q \rceil. \quad (17)$$

We say that an Elman RNN is of

- **constant precision** if $\psi(\mathbf{y}) = \mathcal{O}(1)$, i.e., if $\psi(\mathbf{y}) \leq C$ for all $\mathbf{y} \in \Sigma^*$ and some $C \in \mathbb{R}$,
- **logarithmically bounded precision** if $\psi(\mathbf{y}) = \mathcal{O}(\log |\mathbf{y}|)$, i.e., if there exist $T_0 \in \mathbb{N}$ and $C \in \mathbb{R}$ such that for all $\mathbf{y} \in \Sigma^*$ with $|\mathbf{y}| \geq T_0$, $\psi(\mathbf{y}) \leq C \log_2 |\mathbf{y}|$,
- **linearly bounded precision** if $\psi(\mathbf{y}) = \mathcal{O}(|\mathbf{y}|)$, i.e., if there exist $T_0 \in \mathbb{N}$ and $C \in \mathbb{R}$ such that for all $\mathbf{y} \in \Sigma^*$ with $|\mathbf{y}| \geq T_0$, $\psi(\mathbf{y}) \leq C|\mathbf{y}|$, and
- **unbounded precision** if $\psi(\mathbf{y})$ cannot be bounded by a function of $|\mathbf{y}|$.

The constructions in the existing literature range from constant to unbounded precision. The RNNs considered by Svete and Cotterell (2023), encoding of deterministic PFSAs, for example, are of constant precision. Weiss et al. (2018); Merrill (2019); Merrill et al. (2020), etc., consider models

⁷In his original theorem, Pinkus (1999) only proves the approximation for real-valued functions \mathbf{G} , i.e., $M = 1$. Our restatement trivially extends the original theorem to a range of \mathbb{R}^M noting that one can use one large single-layer MLP consisting of M separate single-layer MLPs in parallel to approximate M individual functions. Furthermore, the original theorem also proves the converse direction that for polynomial β no such MLP exists, which we drop for brevity.

with logarithmically bounded precision. In contrast, Siegelmann and Sontag (1992) and Nowak et al. (2023) require unbounded precision to be able to represent possibly infinite running times required for the Turing completeness of RNNs.

3 Exact Simulation

This section presents our results on the exact simulation (that is, weak equivalence) of regular LMs with both sparsemax as well as softmax-normalized Elman LMs. The results rely on Thm. 3.1, which provides the basis for all our subsequent results by showing that Elman RNNs with linear precision can compute the state–string distributions $p_{\mathcal{A}}(q, \mathbf{y})$ (cf. Eq. (9)) of any PFSA \mathcal{A} .

A note on notation. Throughout the paper, we implicitly index vectors and matrices directly with symbols for conciseness. Concretely, we assume an implicit ordering of the symbols and a corresponding ordering of the vector and matrix entries. This ordering is effected through a bijection $n: \Sigma \rightarrow [|\Sigma|]$ (or any other set in place of Σ). We then define, for $\mathbf{x} \in \mathbb{R}^{|\Sigma|}$, $\mathbf{x}_y \stackrel{\text{def}}{=} \mathbf{x}_{n(y)}$ and, for $\mathbf{X} \in \mathbb{R}^{|\Sigma| \times |\Sigma|}$, $\mathbf{X}_{y,y'} \stackrel{\text{def}}{=} \mathbf{X}_{n(y),n(y')}$ for $y, y' \in \Sigma$.

Theorem 3.1. *Let \mathcal{A} be a PFSA with the state–string distribution $p_{\mathcal{A}}(q, \mathbf{y}_{\leq t})$ (cf. Eq. (9)). Then, there exists an Elman RNN \mathcal{R} with linearly bounded precision such that for all $q \in Q$ and all $\mathbf{y}_{\leq t} = y_1 \dots y_t \in \Sigma^*$*

$$\mathbf{h}(\mathbf{y}_{\leq t})_{(q,y)} = \mathbb{1}\{y = y_t\} p_{\mathcal{A}}(q, \mathbf{y}_{< t}). \quad (18)$$

Proof intuition. The proof translates the computation of the state–string probabilities under the PFSA (cf. Eq. (9)) into the recurrence of the Elman RNN. See App. B for the details. ■

3.1 Sparsemax-normalized Elman LMs

We now present the first result on the representational capacity of Elman LMs of linearly bounded precision: A lower bound showing that they can implement any regular LM, including those induced by non-deterministic PFSA. This is formally captured by the following corollary of Thm. 3.1.

Corollary 3.1. *Let $\mathcal{A} = (\Sigma, Q, \tau, \lambda, \rho)$ be a trim PFSA inducing the tight LM $p_{\mathcal{A}}$. Then, there exists a weakly equivalent, sparsemax-normalized non-linear output Elman LM.*

Proof intuition. Considering Eq. (8a), \mathbf{h}_{t-1} contains all the information needed to compute

$p_{\mathcal{A}}(\bar{y}_t \mid \mathbf{y}_{< t})$. In accordance with Eq. (8b), the hidden state is $\|\cdot\|_1$ -normalized and then multiplied by the output matrix \mathbf{E} containing the values $p_{\mathcal{A}}(\bar{y} \mid q)$. See App. C for the full derivation. ■

3.2 Softmax-normalized Elman LMs

Cor. 3.1 shows that sparsemax-normalized Elman LMs can simulate any non-deterministic PFSA after normalizing the hidden states to unit $\|\cdot\|_1$ norm. The sparsemax function, which is used in place of softmax, constitutes a crucial part of our construction. However, it is more standard to use the softmax in a practical implementation of an Elman RNN, but the softmax does not, unlike the sparsemax, equal the identity function on Δ^{N-1} ; $\text{softmax}(\mathbf{x}) \neq \mathbf{x}$ for $\mathbf{x} \in \Delta^{N-1}$. Luckily, by using a different output function \mathbf{F} , we can achieve weak equivalence using the softmax projection function as well.

Corollary 3.2. *Let $\mathcal{A} = (\Sigma, Q, \tau, \lambda, \rho)$ be a trim PFSA inducing the tight LM $p_{\mathcal{A}}$. Then, there exists a weakly equivalent, softmax-normalized non-linear output Elman LM with an $\overline{\mathbb{R}}$ -valued \mathbf{F} , where $\overline{\mathbb{R}} \stackrel{\text{def}}{=} \mathbb{R} \cup \{-\infty, \infty\}$ are the extended reals.*

Proof intuition. The idea behind this construction is similar to that of Cor. 3.1. Rather than the explicit $\|\cdot\|_1$ -normalization, however, this construction relies on an ($-\infty$ -augmented) log transformation of the hidden state before the softmax normalization, which computes the appropriate probabilities from Eq. (8a). See App. C for the details. ■

4 Approximate Simulation

§3 concerns itself with the exact simulation of regular LMs with Elman LMs. The softmax-normalized result from Cor. 3.2, particularly, shows that a non-linear output Elman LM with an extended-real-valued function \mathbf{F} and a softmax projection function can perfectly simulate any PFSA. The construction, however, relies on the use of the extended reals. Restricting the output function \mathbf{F} to the \mathbb{R} induces approximation errors due to the full support of the softmax on real vectors. To bring the results on the representational capacity of Elman LM even closer to practical implementations, we now discuss how an \mathbb{R} -vector-valued output function \mathbf{F} can be used to approximate regular LMs with Elman RNNs with vanishingly small approximation error. We first consider \mathbf{F} implemented by a custom non-linear function (the log) and later as a common multi-layer perceptron. We use total

variation distance to measure the approximation error between two language models:

Definition 4.1. Let p and q be two probability distributions over Σ^* . Then the **total variation distance** between p and q is given by

$$\text{TVD}(p, q) \stackrel{\text{def}}{=} \frac{1}{2} \sum_{\mathbf{y} \in \Sigma^*} |p(\mathbf{y}) - q(\mathbf{y})|. \quad (19)$$

4.1 Approximation with log

We first show how regular LMs can be approximated with \mathbb{R} -valued output functions. The following theorem states that the $-\infty$ from Cor. 3.1 can be avoided—all outputs can be \mathbb{R} -valued—if we allow for an (arbitrarily small) approximation error.

Theorem 4.1. For any PFSA \mathcal{A} with induced LM $p_{\mathcal{A}}$ and any $\epsilon > 0$, there exists a ReLU-activated non-linear output Elman LM $p_{\mathcal{R}}$ with an \mathbb{R} -vector-valued function $\mathbf{F}: \mathbb{R}^D \rightarrow \mathbb{R}^D$ such that $\text{TVD}(p_{\mathcal{A}}, p_{\mathcal{R}}) < \epsilon$.

Proof intuition. The proof relies on approximating the arbitrary PFSA \mathcal{A} with a *full support* PFSA \mathcal{A}' which approximates the LM induced by \mathcal{A} arbitrarily well in terms of the total variation distance. Since \mathcal{A}' has full support, it can be represented exactly by a softmax-normalized Elman RNN with the same mechanism as in Cor. 3.2, resulting in a recurrent LM approximating the original PFSA \mathcal{A} . See App. D for the full proof. ■

This provides an approximation analog to Cor. 3.1. Due to the absence of infinities, this result is more representative of practical implementations. Nevertheless, the use of the log output function is non-standard. We address this in the next section.

4.2 Approximation with Pinkus (1999)

§4.1 brings the results from §3 to a more practical setting by considering the approximation of string probabilities. The usage of the non-standard log transformation, however, is still unsatisfactory. In this section, we extend the result from the previous section to the most practically relevant setting, in which we consider the approximation of string probabilities under a regular LM using an Elman LM with an output function in the form of an MLP. Since such models are common practice in modern LMs (Pascanu et al., 2014), we consider this to be the most impactful result of our investigation.

Concretely, we show that the approximation abilities of ReLU-activated deep neural networks allow

us to approximate the conditional probabilities of a PFSA arbitrarily well, providing theoretical backing for the strong performance of softmax-normalized ERNNs on regular languages.⁸

Theorem 4.2. Let $p_{\mathcal{A}}$ be a regular LM induced by a PFSA \mathcal{A} and let $\epsilon > 0$. Then, there exists a ReLU-activated, softmax-normalized deep output Elman LM $p_{\mathcal{R}}$ such that $\text{TVD}(p_{\mathcal{A}}, p_{\mathcal{R}}) < \epsilon$.

Proof intuition. The final construction relies on approximating two functions: (1) the LM induced by the arbitrary \mathcal{A} with a full support PFSA \mathcal{A}' and (2) the log output function. Thm. 4.1 solves step (1) while the application of Thm. 2.1 to the approximation of log solves step (2). Importantly, the full support assumption provides the necessary compact domain over which log is approximated. See App. D for the full proof. ■

5 Discussion

§§ 3 and 4 present a series of results describing the probabilistic representational capacity of Elman LMs. The results range from theorems that are simple, but detached from practical implementations (Cors. 3.1 and 3.2), to more elaborate theorems that describe the *approximation* abilities of Elman LMs with modeling choices very close to those used in practice (Thm. 4.1). All results lower-bound the probabilistic representational capacity of Elman LMs, i.e., they state what Elman LM *can* do. We discuss the implications of these lower bounds here.

Approximation abilities of softmax-normalized Elman LMs.

The most practically relevant result in our paper is the approximation result of Thm. 4.2. Indeed, other results in this paper can be seen as a step-by-step build-up, each stripping away unpractical assumptions leading to this final approximation result. Thm. 4.2 shows that Elman LMs can approximate not only the individual conditional probabilities but rather any tight regular LM arbitrarily well as measured by total variation distance. Note that while arbitrarily good approximation of the *conditional* probabilities does not necessarily imply the arbitrarily good approximation of full string probabilities (string probabilities

⁸Our construction applies an approximation result by Pinkus (1999), which characterizes wide fixed-depth neural networks. An analogous result could naturally be stated for fixed-width deep ReLU networks by applying any of the approximation theorems that cover such cases (e.g., Montúfar et al., 2014; Yarotsky, 2017; Raghu et al., 2017; Hanin, 2019).

are computed by multiplying the conditional probabilities of the individual symbols arbitrarily many times, resulting in a potentially large error), the assumption of tightness nevertheless allows us to make the total variation arbitrarily small—since the probability mass of a sufficiently long string under a tight language model diminishes with the string length, the strings with accumulating error only contribute marginally to the total variation distance between the true regular LM and its neural approximation.

Non-determinism and neural LMs. One of the main motivations for this investigation was capturing the non-determinism of regular LMs with neural LMs. Non-determinism is a prevalent notion in formal language theory as it allows both for more expressive formalisms as well as a more concise and intuitive construction of formal models of computation (Hopcroft et al., 2006, Ch. 2). However, existing work has not addressed the implications of non-determinism on the link between regular LMs and neural LMs. To resolve this, the constructions presented here propose direct ways of implementing non-deterministic PFSA with Elman LMs by means of linearly increasing the precision of hidden states, which is able to model all possible decisions of the PFSA directly. This allows us to lower-bound the representational capacity of real-time Elman LMs (cf. Cor. 3.1). This lower bound implies Elman LMs with linearly bounded precision are strictly more expressive than their counterparts with finite precision, which are only able to express *deterministic* PFSA (Svete and Cotterell, 2023). To the best of our knowledge, our construction is the first connection that tackles non-determinism directly by emulating the computation of string probabilities in real time.⁹ Moreover, addressing non-determinism also allows us to *compress* otherwise determinizable automata into weakly equivalent non-deterministic ones and, thus, implement them in possibly exponentially smaller space (Buchsbaum et al., 2000) by directly simulating the smaller non-deterministic automaton with an Elman LM, a substantial improvement over existing work.¹⁰

⁹Alternatively, one could define neural LMs that simulate all accepting paths individually, which would necessitate the introduction of some form of non-determinism into the neural architecture (Nowak et al., 2023, 2024).

¹⁰While such an LM would still require a hidden state linear in the number of states (like in the deterministic case), the number of states of the non-deterministic automaton

On the role of real-time processing. We limit ourselves to the real-time processing of strings by Elman LMs, as this most realistically captures how most modern LMs work (Weiss et al., 2018). Moreover, real-time Elman LMs are not *upper*-bounded by regular LMs. Using the same stack encoding function as Nowak et al. (2023), one can show that Elman LMs with linearly bounded precision can probabilistically generate a weighting of a context-free Dyck language of nested parentheses because simulating the required stack can be performed in real-time.¹¹ Moreover, by simulating multiple stacks, Elman LMs can simulate real-time Turing machines with arbitrarily many tapes (Nowak et al., 2023), resulting in computational power exceeding that of PFSA (Rabin, 1963; Rosenberg, 1967). Going beyond real-time processing, a classical result by Siegelmann and Sontag (1992), and extended to the probabilistic case by Nowak et al. (2023), establishes that RNNs with unbounded precision can simulate Turing machines. However, the requirement of unbounded precision (and thereby computation time) is a departure from how RNNs function in practice. Nowak et al. (2023) place upper and lower bounds on the types of LMs ReLU-activated Elman RNNs can represent. Our real-time results stand in contrast to such Turing completeness theorems.

Space complexity of emulating regular LMs.

Our constructions show that hidden states of size $\mathcal{O}(|Q||\Sigma|)$ are sufficient to simulate arbitrary regular LMs. While we do not address the *lower bounds* on the space required for the simulation of regular LMs, we conjecture that existing lower bounds for deterministic PFSA apply here (Svete and Cotterell, 2023, Theorems 5.1 and 5.2). Concretely, the size of the Elman LM is lower-bounded by $\min(|Q|, |\bar{\Sigma}|)$ due to the **softmax bottleneck** principle (Yang et al., 2018; Svete et al., 2024; Borenstein et al., 2024). Because the PFSA inducing a regular LM can, in general, define $|Q|$ probability distributions over $\bar{\Sigma}$, a weakly equivalent Elman LM needs to model a $\Omega(\min(|Q|, |\bar{\Sigma}|))$ -dimensional space of logits to match the conditional probability distributions. This, in general, requires $\Omega(\min(|Q|, |\bar{\Sigma}|))$ -dimensional hidden states (Chang and McCallum, 2022).

would be smaller.

¹¹Note that all (probabilistic) context-free languages can be recognized in real-time (Greibach, 1965; Abney et al., 1999).

Probabilistic Regular Languages. The relationship between RNNs and FSAs has attracted a lot of attention, resulting in a well-understood connection between RNNs and binary regular languages (Kleene, 1956; Cleeremans et al., 1989; Korsky and Berwick, 2019; Merrill, 2019; Merrill et al., 2020), as well as between RNNs and *deterministic* probabilistic regular languages (Svete and Cotterell, 2023). To the best of our knowledge, Peng et al. (2018) are the first to discuss recurrent neural models that can simulate the computation of string probabilities under a set of distinct PFSAs. In particular, each dimension of the network’s hidden state represents the probability of the input string under a specific PFSA. Their construction, however, does not consider the *lower bound* of representational capacity of standard RNN architectures such as the Elman RNN, and, importantly, despite being weighted, is not connected to the language modeling setting. While some aspects of our construction resemble theirs, we notably extend it to the language modeling setting, where the generalization to the probabilistic case (Icard, 2020) is especially relevant.

6 Conclusion

We study a practically relevant formulation of Elman LMs with real-time processing and linear precision with respect to the string length and describe its probabilistic representational capacity in terms of regular LMs. Concretely, we show that Elman LMs with non-standard normalization and activation functions can simulate arbitrary non-deterministic PFSAs (Cors. 3.1 and 3.2). We then extend these exact simulation results to *approximation-based* ones, where we show that Elman LMs as used in practice can approximate regular LMs arbitrarily well as measured by total variation distance (Thm. 4.2). The intuitive constructions provide a better understanding of the representational capacity of Elman LMs and showcase the probabilistic representational capacity of Elman LMs in a new light.

Limitations

Since we only present lower bounds on the representational capacity of Elman LMs and no upper bounds, we do not provide a complete picture of their representational capacity. We also do not consider the *learnability* of regular LMs with Elman LMs. This crucial part of language modeling is a

much harder problem that remains to be tackled. However, many existing results from formal language theory show that learning regular LMs based on only positive examples (as done in modern approaches to language modeling) is hard or even impossible (Gold, 1967; Kearns and Valiant, 1994). The same naturally holds for the more expressive models of computation that Elman LMs can also emulate. This puts hard bounds on what can be learned in the language modeling framework from data alone, irrespective of the concrete parameterization of the model—be it RNNs, transformers, or any other model.

The restriction to the consideration of regular LMs of course fails to consider the non-regular phenomena of human language (Chomsky, 1957). While limits of human cognition might suggest that large portions of human language can be modeled by regular mechanisms (Hewitt et al., 2020; Svete et al., 2024), such formalisms lack the structure and interpretability of some mechanisms higher on the Chomsky hierarchy. However, the possibility of simulating many other real-time models of computation suggests that linearly bounded precision Elman LMs might indeed be able to capture more than just regular aspects of human language.

Ethics Statement

The paper advances our theoretical understanding of language models. The authors do not foresee any ethical implications of this paper.

Acknowledgements

Ryan Cotterell acknowledges support from the Swiss National Science Foundation (SNSF) as part of the “The Forgotten Role of Inductive Bias in Interpretability” project. Anej Svete is supported by the ETH AI Center Doctoral Fellowship.

References

- Steven Abney, David McAllester, and Fernando Pereira. 1999. [Relating probabilistic grammars and automata](#). In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 542–549, College Park, Maryland, USA. Association for Computational Linguistics.
- Joshua Ackerman and George Cybenko. 2020. [A survey of neural networks and formal languages](#). *arXiv preprint arXiv:2006.01338*.
- Alon Albalak, Yi-Lin Tuan, Pegah Jandaghi, Connor Pryor, Luke Yoffe, Deepak Ramachandran, Lise

- Getoor, Jay Pujara, and William Yang Wang. 2022. **FETA: A benchmark for few-sample task transfer in open-domain dialogue**. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 10936–10953, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.
- Cyril Allauzen and Mehryar Mohri. 2003. **Efficient algorithms for testing the twins property**. *Journal of Automata, Languages and Combinatorics*, 8:117–144.
- Noga Alon, A. K. Dewdney, and Teunis J. Ott. 1991. **Efficient simulation of finite automata by neural nets**. *Journal of the ACM*, 38(2):495–514.
- Nadav Borenstein, Anej Svete, Robin Shing Moon Chan, Josef Valvoda, Franz Nowak, Isabelle Augenstein, Eleanor Chodroff, and Ryan Cotterell. 2024. **What languages are easy to language-model? a perspective from learning probabilistic regular languages**. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Bangkok, Thailand. Association for Computational Linguistics.
- Stephen Boyd and Lieven Vandenbergh. 2004. *Convex Optimization*. Cambridge University Press.
- Adam L. Buchsbaum, Raffaele Giancarlo, and Jeffrey R. Westbrook. 2000. **On the determinization of weighted finite automata**. *SIAM Journal on Computing*, 30(5):1502–1531.
- Haw-Shiuan Chang and Andrew McCallum. 2022. **Softmax bottleneck makes language models unable to represent multi-mode word distributions**. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8048–8073, Dublin, Ireland. Association for Computational Linguistics.
- Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014a. **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. Association for Computational Linguistics.
- Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014b. **Learning phrase representations using RNN encoder–decoder for statistical machine translation**. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.
- Noam Chomsky. 1957. *Syntactic Structures*. De Gruyter Mouton, Berlin, Boston.
- Axel Cleeremans, David Servan-Schreiber, and James L. McClelland. 1989. **Finite state automata and simple recurrent networks**. *Neural Computation*, 1(3):372–381.
- G. Cybenko. 1989. **Approximation by superpositions of a sigmoidal function**. *Mathematics of Control, Signals and Systems*, 2(4):303–314.
- Li Du, Lucas Torroba Hennigen, Tiago Pimentel, Clara Meister, Jason Eisner, and Ryan Cotterell. 2023. **A measure-theoretic characterization of tight language models**. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9744–9770, Toronto, Canada. Association for Computational Linguistics.
- Jeffrey L. Elman. 1990. **Finding structure in time**. *Cognitive Science*, 14(2):179–211.
- Ken-Ichi Funahashi. 1989. **On the approximate realization of continuous mappings by neural networks**. *Neural Networks*, 2(3):183–192.
- Bolin Gao and Lacra Pavel. 2018. **On the properties of the softmax function with application in game theory and reinforcement learning**. *arXiv preprint arXiv:1704.00805*.
- E Mark Gold. 1967. **Language identification in the limit**. *Information and Control*, 10(5):447–474.
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>.
- Sheila A. Greibach. 1965. **A new normal-form theorem for context-free phrase structure grammars**. *J. ACM*, 12(1):42–52.
- Boris Hanin. 2019. **Universal function approximation by deep neural nets with bounded width and relu activations**. *Mathematics*, 7(10):992.
- John Hewitt, Michael Hahn, Surya Ganguli, Percy Liang, and Christopher D. Manning. 2020. **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. Association for Computational Linguistics.
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. **Long short-term memory**. *Neural Computation*, 9(8):1735–1780.
- John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2006. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA.
- Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. **Multilayer feedforward networks are universal approximators**. *Neural Networks*, 2(5):359–366.
- Thomas F. Icard. 2020. **Calibrating generative models: The probabilistic Chomsky–Schützenberger hierarchy**. *Journal of Mathematical Psychology*, 95:102308.

- P. Indyk. 1995. [Optimal simulation of automata by neural nets](#). In *STACS 95*, pages 337–348, Berlin, Heidelberg. Springer Berlin Heidelberg.
- Michael Kearns and Leslie Valiant. 1994. [Cryptographic limitations on learning boolean formulae and finite automata](#). *Journal of the ACM*, 41(1):67–95.
- S. C. Kleene. 1956. [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.
- Samuel A. Korsky and Robert C. Berwick. 2019. [On the computational power of RNNs](#). *arXiv preprint arXiv:1906.06349*.
- André F. T. Martins and Ramón F. Astudillo. 2016. [From softmax to sparsemax: A sparse model of attention and multi-label classification](#). In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICML'16*, page 1614–1623.
- William Merrill. 2019. [Sequential neural networks as automata](#). In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 1–13, Florence. Association for Computational Linguistics.
- William Merrill, Ashish Sabharwal, and Noah A. Smith. 2022. [Saturated transformers are constant-depth threshold circuits](#). *Transactions of the Association for Computational Linguistics*, 10:843–856.
- William Merrill and Nikolaos Tsilivis. 2022. [Extracting finite automata from RNNs using state merging](#). *arXiv preprint arXiv:2201.12451*.
- William Merrill, Gail Weiss, Yoav Goldberg, Roy Schwartz, Noah A. Smith, and Eran Yahav. 2020. [A formal hierarchy of RNN architectures](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 443–459, Online. Association for Computational Linguistics.
- Mehryar Mohri. 1997. [Finite-state transducers in language and speech processing](#). *Computational Linguistics*, 23(2):269–311.
- Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. 2014. [On the number of linear regions of deep neural networks](#). In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS'14*, page 2924–2932, Cambridge, MA, USA. MIT Press.
- Franz Nowak, Anej Svete, Alexandra Butoi, and Ryan Cotterell. 2024. [On the representational capacity of neural language models with chain-of-thought reasoning](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Bangkok, Thailand. Association for Computational Linguistics.
- Franz Nowak, Anej Svete, Li Du, and Ryan Cotterell. 2023. [On the representational capacity of recurrent neural language models](#). *arXiv preprint arXiv:2310.12942*.
- Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. [How to construct deep recurrent neural networks: Proceedings of the second international conference on learning representations \(iclr 2014\)](#). In *2nd International Conference on Learning Representations, ICLR 2014*.
- Hao Peng, Roy Schwartz, Sam Thomson, and Noah A. Smith. 2018. [Rational recurrences](#). In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1203–1214, Brussels, Belgium. Association for Computational Linguistics.
- Allan Pinkus. 1999. [Approximation theory of the MLP model in neural networks](#). *Acta Numerica*, 8:143–195.
- Jorge Pérez, Pablo Barceló, and Javier Marinkovic. 2021. [Attention is Turing-complete](#). *Journal of Machine Learning Research*, 22(75):1–35.
- Michael O. Rabin. 1963. [Real time computation](#). *Israel Journal of Mathematics*, 1(4):203–211.
- Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. 2017. [On the expressive power of deep neural networks](#). In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2847–2854. PMLR.
- Arnold L. Rosenberg. 1967. [Real-time definable languages](#). *Journal of the ACM*, 14(4):645–662.
- Timo Schick and Hinrich Schütze. 2021. [Exploiting cloze-questions for few-shot text classification and natural language inference](#). In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 255–269, Online. Association for Computational Linguistics.
- Hava T. Siegelmann and Eduardo D. Sontag. 1992. [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. Association for Computing Machinery.
- Xiaofei Sun, Xiaoya Li, Jiwei Li, Fei Wu, Shangwei Guo, Tianwei Zhang, and Guoyin Wang. 2023. [Text classification via large language models](#). In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 8990–9005, Singapore. Association for Computational Linguistics.
- Anej Svete, Robin Shing Moon Chan, and Ryan Cotterell. 2024. [A theoretical result on the inductive bias of RNN language models](#). *arXiv preprint arXiv:2402.15814*.

- Anej Svete and Ryan Cotterell. 2023. [Recurrent neural language models as probabilistic finite-state automata](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8069–8086, Singapore. Association for Computational Linguistics.
- Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. [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. Association for Computational Linguistics.
- Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. 2018. [Breaking the softmax bottleneck: A high-rank RNN language model](#). In *International Conference on Learning Representations*.
- Dmitry Yarotsky. 2017. [Error bounds for approximations with deep relu networks](#). *Neural Networks*, 94:103–114.

A Computation of String Probabilities Under PFSAs

In this section, we describe how a general PFSA computes string probabilities using standard linear algebra notation. This will allow for a concise connection to the Elman recurrence (cf. Eq. (10b)) later. Let $\mathcal{A} = (\Sigma, Q, \tau, \lambda, \rho)$ be a PFSA and let $\mathbf{y} \in \Sigma^*$ be a string. Let $\{\mathbf{T}^{(y)} \mid y \in \Sigma\}$ be the set of symbol-specific transition matrices of \mathcal{A} , i.e., the matrices with entries

$$\mathbf{T}_{(q,q')}^{(y)} \stackrel{\text{def}}{=} \begin{cases} w & \text{if } q \xrightarrow{y/w} q' \in \tau \\ 0 & \text{otherwise} \end{cases}. \quad (20)$$

That is, $\mathbf{T}_{(q,q')}^{(y)}$ denotes the probability of transitioning from state q to state q' with the symbol y . Then, the probability of a string $\mathbf{y} = y_1 \dots y_T \in \Sigma^*$ under \mathcal{A} is given by:

$$p_{\mathcal{A}}(\mathbf{y}) = \vec{\lambda}^{\top} \mathbf{T}^{(y_1)} \mathbf{T}^{(y_2)} \dots \mathbf{T}^{(y_T)} \vec{\rho}, \quad (21)$$

where $\vec{\lambda} \stackrel{\text{def}}{=} (\lambda(q))_{q \in Q}$ and $\vec{\rho} \stackrel{\text{def}}{=} (\rho(q))_{q \in Q}$ are vectors whose elements are the initial and final weights of the states of \mathcal{A} , respectively (Peng et al., 2018).

B Construction of an RNN to Simulate PFSAs

Theorem 3.1. *Let \mathcal{A} be a PFSA with the state–string distribution $p_{\mathcal{A}}(q, \mathbf{y}_{\leq t})$ (cf. Eq. (9)). Then, there exists an Elman RNN \mathcal{R} with linearly bounded precision such that for all $q \in Q$ and all $\mathbf{y}_{\leq t} = y_1 \dots y_t \in \Sigma^*$*

$$\mathbf{h}(\mathbf{y}_{\leq t})_{(q,y)} = \mathbb{1}\{y = y_t\} p_{\mathcal{A}}(q, \mathbf{y}_{< t}). \quad (18)$$

Proof. To prove Thm. 3.1 we must, for any (possibly non-deterministic) PFSA $\mathcal{A} = (\Sigma, Q, \tau, \lambda, \rho)$, construct an Elman RNN that keeps track of the state–string probabilities in its hidden states. At a high level, we want to show that the equality Eq. (18) holds for all string prefixes $\mathbf{y}_{\leq t} \in \Sigma^*$ for all time steps $t \in \mathbb{N}$, that is, it is an invariance that holds for all $t \in \mathbb{N}$. We provide the full construction of the RNN in multiple steps. The construction uses a hidden dimension of $D = |Q||\Sigma|$.

Initial condition. The initial hidden state, $\mathbf{h}_0 = \boldsymbol{\eta}$ encodes the initial probability distribution over Q —the one defined by the initial weighting function λ —but only for positions corresponding to a single arbitrary dummy input symbol $y_0 \in \Sigma$:¹²

$$\boldsymbol{\eta}_{(q,y_0)} \stackrel{\text{def}}{=} \lambda(q), \quad \forall q \in Q = p_{\mathcal{A}}(q, \varepsilon). \quad (22)$$

Encoding the transition function. The Elman recurrence (cf. Eq. (10b)) will effectively simulate the dynamics of the PFSA. Intuitively, the updates to the hidden state will be analogous to parts of the computation Eq. (21); the individual updates to the hidden state (through the multiplication with the recurrence matrix \mathbf{U}) are analogous to the multiplications with the matrices $\mathbf{T}^{(y)}$. However, since the RNN cannot *choose* among the possible transition matrices $\{\mathbf{T}^{(y)} \mid y \in \Sigma\}$ when updating the hidden state (unlike the PFSA), the hidden state update rule will perform *all possible* transitions in parallel and then mask out the transitions that are not applicable to the current symbol.

The recurrence matrix $\mathbf{U} \in \mathbb{R}^{D \times D}$ thus encodes the transition weights. By having access to the hidden state \mathbf{h}_t through the matrix–vector multiplication, \mathbf{U} performs all possible transitions of the PFSA, i.e., the transitions for all symbols, in parallel. That is, for each symbol $y \in \Sigma$, it computes a new distribution over states $p_{\mathcal{A}}(q, \mathbf{y}_{\leq t}y)$.

The entries of the recurrence matrix are set as follows, for all $y' \in \Sigma$:

$$\mathbf{U}_{(q',y'),(q,y)} \stackrel{\text{def}}{=} \begin{cases} w & \text{if } q \xrightarrow{y'/w} q' \in \tau \\ 0 & \text{otherwise} \end{cases}. \quad (23)$$

¹²Throughout the text, we assume that all non-specified entries of the parameters are set to 0.

Since y' is free, the product $\mathbf{U}\mathbf{h}_{t-1}$ copies $p_{\mathcal{A}}(q \mid \mathbf{y}_{<t})$ into every entry associated with q . Visually, the recurrence matrix \mathbf{U} consists of the transition matrices of each symbol, stacked vertically and copied horizontally, e.g., if $\Sigma = \{a, b, \dots, z\}$:

$$\mathbf{U} = \begin{bmatrix} \mathbf{U}^{(a)} & \mathbf{U}^{(b)} & \dots & \mathbf{U}^{(z)} \\ \mathbf{U}^{(a)} & \mathbf{U}^{(b)} & \dots & \mathbf{U}^{(z)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{U}^{(a)} & \mathbf{U}^{(b)} & \dots & \mathbf{U}^{(z)} \end{bmatrix} \in \mathbb{R}^{D \times D}, \quad (24)$$

where each transition matrix $\mathbf{U}^{(y)} \stackrel{\text{def}}{=} (\mathbf{T}^{(y)})^\top$ has entries

$$\mathbf{U}_{q',q}^{(y)} \stackrel{\text{def}}{=} \begin{cases} w & \text{if } q \xrightarrow{y/w} q' \in \tau \\ 0 & \text{otherwise} \end{cases}. \quad (25)$$

This way, multiplication with the recurrence matrix preserves the invariance in Eq. (18):

$$(\mathbf{U}\mathbf{h}_t)_{(q',y)} = \sum_{q \xrightarrow{y/w} q' \in \tau} w \cdot p_{\mathcal{A}}(q, \mathbf{y}_{\leq t}) \quad (26a)$$

$$= \sum_{q \in Q} p_{\mathcal{A}}(y, q' \mid q) \cdot p_{\mathcal{A}}(q, \mathbf{y}_{\leq t}) \quad (26b)$$

$$= \sum_{q \in Q} p_{\mathcal{A}}(y, q' \mid q, \mathbf{y}_{\leq t}) \cdot p_{\mathcal{A}}(q, \mathbf{y}_{\leq t}) \quad (26c)$$

$$= \sum_{q \in Q} p_{\mathcal{A}}(y, q' \mid q, \mathbf{y}_{\leq t}) \cdot p_{\mathcal{A}}(q \mid \mathbf{y}_{\leq t}) \cdot p_{\mathcal{A}}(\mathbf{y}_{\leq t}) \quad (26d)$$

$$= \left(\sum_{q \in Q} p_{\mathcal{A}}(y, q' \mid q, \mathbf{y}_{\leq t}) \cdot p_{\mathcal{A}}(q \mid \mathbf{y}_{\leq t}) \right) \cdot p_{\mathcal{A}}(\mathbf{y}_{\leq t}) \quad (26e)$$

$$= p_{\mathcal{A}}(y, q' \mid \mathbf{y}_{\leq t}) \cdot p_{\mathcal{A}}(\mathbf{y}_{\leq t}) \quad (26f)$$

$$= p_{\mathcal{A}}(y, q', \mathbf{y}_{\leq t}) \quad (26g)$$

$$= p_{\mathcal{A}}(q', \mathbf{y}_{\leq t} y). \quad (26h)$$

We now have a distribution over states and symbols given the history. However, at time step t we also observe the actual input symbol, meaning that we want to only retain the distribution over states for the observed symbol.

We use the input matrix \mathbf{V} and the bias vector \mathbf{b} to zero out parts of the hidden state so that it only retains the probabilities of states reachable by reading the current symbol y_t and sets the rest to 0. Since the ReLU activation function zeroes out negative values, it will be sufficient to subtract an appropriate bias term from the entries associated with unobserved symbols. This can be achieved by defining, for any $q \in Q$ and $y \in \Sigma$:

$$\mathbf{V}_{(q,y),(y)} \stackrel{\text{def}}{=} 1, \quad (27a)$$

$$\mathbf{b} \stackrel{\text{def}}{=} -\mathbf{1}_D, \quad (27b)$$

where $\mathbf{1}_D$ is a D -dimensional vector of ones. This ensures that the negative bias values are balanced by the product $\mathbf{V}\mathbf{r}(y_t)$ exactly for the tuple entries associated with symbol y_t :

$$\mathbf{h}(\mathbf{y}_{\leq t} y)_{(q',y')} = \text{ReLU}(\mathbf{U}_{(q',y')} \mathbf{h}(\mathbf{y}_{\leq t}) + \mathbf{V}_{(q',y')} \llbracket y \rrbracket + \mathbf{b}_{(q',y')}) \quad (28a)$$

$$= \text{ReLU}(p_{\mathcal{A}}(q', \mathbf{y}_{\leq t} y) + \mathbb{1}\{y' = y\} - 1) \quad (28b)$$

$$= \mathbb{1}\{y' = y\} p_{\mathcal{A}}(q', \mathbf{y}_{\leq t} y). \quad (28c)$$

Since probability scores cannot be greater than 1, the values corresponding to symbols that were not observed will be zeroed out by the activation function.

Linearly Bounded Precision. Finally, we want to show that the RNN requires linearly bounded precision, i.e., there exist $T_0 \in \mathbb{N}$ and $C \in \mathbb{R}_{\geq 0}$ such that for all $\mathbf{y} \in \Sigma^*$ with $|\mathbf{y}| \geq T_0$, $\psi(\mathbf{y}) \leq C|\mathbf{y}|$. In the following, we write the precision required by a rational $x \in \mathbb{Q}$ as

$$\psi(x) \stackrel{\text{def}}{=} \min_{\substack{p, q \in \mathbb{N}, \\ \frac{p}{q} = x}} [\log_2 p] + [\log_2 q]. \quad (29)$$

Note that $\psi(xy) \leq \psi(x) + \psi(y)$ for all $x, y \in \mathbb{Q}$.

Two remarks will be important for the proof:

1. The number of bits required to represent any dimension of $\mathbf{Vr}(y_t) + \mathbf{b}$ is upper bounded by a constant C' independent of t , given by

$$C' \stackrel{\text{def}}{=} \max_{y \in \Sigma} \max_{d \in [D]} \psi([\mathbf{Vr}(y) + \mathbf{b}]_d). \quad (30)$$

2. The ReLU either leaves a dimension of the hidden state unchanged or sets it to 0, meaning it never increases the required precision of its argument.

We show linear precision by induction over $|\mathbf{y}|$ for $\mathbf{y} \in \Sigma^*$ with $|\mathbf{y}| \geq T_0 \stackrel{\text{def}}{=} 1$, choosing C as

$$C \stackrel{\text{def}}{=} \psi(D) + \max_{i, j \in [D]} \psi(\mathbf{U}_{ij}) + \max_d \psi(\boldsymbol{\eta}_d) + C'. \quad (31)$$

- **Base case:** For $\mathbf{y} = y$ for some $y \in \Sigma$,

$$\psi(y) = \max_{d \in [D]} \psi(\mathbf{h}_{1,d}) \quad (32a)$$

$$= \max_{d \in [D]} \psi(\text{ReLU}(\mathbf{U}\boldsymbol{\eta} + \mathbf{Vr}(y) + \mathbf{b})_d) \quad (32b)$$

$$\leq \max_{d \in [D]} \psi((\mathbf{U}\boldsymbol{\eta})_d) + C' \quad (32c)$$

$$\leq \max_{d, i, j \in [D]} \psi(D\mathbf{U}_{i,j}\boldsymbol{\eta}_d) + C' \quad (32d)$$

$$\leq \psi(D) + \max_{i, j \in [D]} \psi(\mathbf{U}_{ij}) + \max_{d \in [D]} \psi(\boldsymbol{\eta}_d) + C' \quad (32e)$$

$$= C. \quad (32f)$$

- **Induction step:** Assume that $\psi(\mathbf{y}) \leq C|\mathbf{y}|$ for all $\mathbf{y} \in \Sigma^*$ with $|\mathbf{y}| = T$. Let $\mathbf{y} \in \Sigma^*$ with $|\mathbf{y}| = T$ be such a string. Then, for $\mathbf{y}' \stackrel{\text{def}}{=} \mathbf{y}y'$ with $y' \in \Sigma$ (a string of length $T + 1$), we have

$$\psi(\mathbf{y}') = \max_{d \in [D]} \psi(\mathbf{h}_{t+1,d}) \quad (33a)$$

$$= \max_{d \in [D]} \psi(\text{ReLU}(\mathbf{U}\mathbf{h}_t + \mathbf{Vr}(y') + \mathbf{b})_d) \quad (33b)$$

$$\leq \max_{d \in [D]} \psi((\mathbf{U}\mathbf{h}_t)_d) + C' \quad (33c)$$

$$\leq \psi(D) + \max_{i, j \in [D]} \psi(\mathbf{U}_{ij}) + \underbrace{\max_{d \in [D]} \psi(\mathbf{h}_{t-1,d})}_{=\psi(\mathbf{y})} + C' \quad (33d)$$

$$\leq C + C|\mathbf{y}| \quad (33e)$$

$$= C(\mathbf{y} + 1) \quad (33f)$$

$$= C\mathbf{y}'. \quad (33g)$$

Therefore, any ReLU-activated Elman RNN over the rationals can be represented with linear precision. Note that for $t = 0$, $\mathbf{h}_0 = \boldsymbol{\eta}$, which requires constant precision depending only

on the choice of $\boldsymbol{\eta}$ which is included in C , so above we actually showed a stronger bound of $\psi(t) \leq C(t+1)$ that applies to all time steps. Observe also that we use the fact that each computational step in our model corresponds to reading a symbol in Σ ; this is in contrast to models such as those investigated in Nowak et al. (2023), which are allowed to consume the empty string ε .

■

C Proofs of Exact Simulation

Corollary 3.1. *Let $\mathcal{A} = (\Sigma, Q, \tau, \lambda, \rho)$ be a trim PFSA inducing the tight LM $p_{\mathcal{A}}$. Then, there exists a weakly equivalent, sparsemax-normalized non-linear output Elman LM.*

Proof. Let $p_{\mathcal{A}}(q, \mathbf{y})$ be \mathcal{A} 's state-string distribution (cf. Eq. (9)). By Thm. 3.1, there exists an Elman RNN $\mathcal{R} = (\mathbb{Q}^D, \Sigma, \sigma, \mathbf{U}, \mathbf{V}, \mathbf{b}, \boldsymbol{\eta})$ with the hidden states

$$\mathbf{h}(\mathbf{y}_{<t})_{(q,y)} = \mathbb{1}\{y = y_{t-1}\} p_{\mathcal{A}}(q, \mathbf{y}_{<t}) \quad (34)$$

for all $\mathbf{y}_{<t} \in \Sigma^*$.

We now define a simple non-linear transformation \mathbf{F} that transforms $\mathbf{h}(\mathbf{y}_{<t})$ into values that are later normalized to $p_{\mathcal{A}}(y_t | \mathbf{y}_{<t})$ by the sparsemax. The function \mathbf{F} is a composition of two functions: (1) $\|\cdot\|_1$ -normalization of the hidden state and (2) linear transformation of the normalized hidden state that sums the outgoing transitions from $q \in Q$ that emit \bar{y}_t . \mathbf{F} will thus take the form

$$\mathbf{F}(\mathbf{h}) \stackrel{\text{def}}{=} \mathbf{E} \frac{\mathbf{h}}{\|\mathbf{h}\|_1} \quad (35)$$

for some $\mathbf{E} \in \mathbb{R}^{|\bar{\Sigma}| \times D}$ that we define below.

Defining $\mathbf{h} \stackrel{\text{def}}{=} \mathbf{h}_t = \mathbf{h}(\mathbf{y}_{<t})$ and applying $\|\cdot\|_1$ normalization to \mathbf{h} as $\mathbf{h}' \stackrel{\text{def}}{=} \frac{\mathbf{h}}{\|\mathbf{h}\|_1}$ results in

$$\mathbf{h}'_{(q,y)} = \frac{\mathbb{1}\{y = y_{t-1}\} p_{\mathcal{A}}(q, \mathbf{y}_{<t})}{\sum_{d'} |\mathbf{h}_{d'}|} \quad (36a)$$

$$= \frac{\mathbb{1}\{y = y_{t-1}\} p_{\mathcal{A}}(q, \mathbf{y}_{<t})}{\sum_{q' \in Q, y' \in \Sigma} |\mathbf{h}(q', y')|} \quad (36b)$$

$$= \frac{\mathbb{1}\{y = y_{t-1}\} p_{\mathcal{A}}(q, \mathbf{y}_{<t})}{\sum_{q' \in Q, y' \in \Sigma} \mathbb{1}\{y' = y_{t-1}\} p_{\mathcal{A}}(q', \mathbf{y}_{<t})} \quad (36c)$$

$$= \frac{\mathbb{1}\{y = y_{t-1}\} p_{\mathcal{A}}(q, \mathbf{y}_{<t})}{\sum_{q' \in Q} p_{\mathcal{A}}(q', \mathbf{y}_{<t})} \quad (36d)$$

$$= \mathbb{1}\{y = y_{t-1}\} p_{\mathcal{A}}(q | \mathbf{y}_{<t}), \quad (36e)$$

i.e., the \mathbf{h}' holds a distribution over states given all the input symbols up to time step $t-1$. We define \mathbf{E} for $q \in Q, y \in \Sigma, \bar{y} \in \bar{\Sigma}$ as

$$\mathbf{E}_{\bar{y},(q,y)} \stackrel{\text{def}}{=} \begin{cases} \sum_{q \xrightarrow{y/w} q' \in \tau} w & \text{if } \bar{y} \in \Sigma \\ \rho(q) & \text{if } \bar{y} = \text{EOS} \end{cases} = \begin{cases} p_{\mathcal{A}}(y | q) & \text{if } \bar{y} \in \Sigma \\ p_{\mathcal{A}}(\text{EOS} | q) & \text{if } \bar{y} = \text{EOS} \end{cases}. \quad (37)$$

Computing the matrix–vector product $\mathbf{E} \mathbf{h}'(\mathbf{y}_{<t})$ reveals that $(\mathbf{E} \mathbf{h}'(\mathbf{y}_{<t}))_{\bar{y}} = p_{\mathcal{A}}(\bar{y} | \mathbf{y}_{<t})$:

$$(\mathbf{E} \mathbf{h}'(\mathbf{y}_{<t}))_{\bar{y}} = \sum_{d=1}^D \mathbf{E}_{\bar{y},d} \mathbf{h}'(\mathbf{y}_{<t})_d \quad (38a)$$

$$= \sum_{q \in Q, y \in \Sigma} \mathbf{E}_{\bar{y},(q,y)} \mathbf{h}'(\mathbf{y}_{<t})_{(q,y)} \quad (38b)$$

$$= \sum_{q \in Q, y \in \Sigma} p_{\mathcal{A}}(\bar{y} | q) \mathbb{1}\{y = y_{t-1}\} p_{\mathcal{A}}(q | \mathbf{y}_{<t}) \quad (38c)$$

$$= \sum_{q \in Q} p_{\mathcal{A}}(\bar{y} | q) p_{\mathcal{A}}(q | \mathbf{y}_{<t}) \quad (38d)$$

$$= \sum_{q \in Q} p_{\mathcal{A}}(\bar{y} | q, \mathbf{y}_{<t}) p_{\mathcal{A}}(q | \mathbf{y}_{<t}) \quad (38e)$$

$$= p_{\mathcal{A}}(\bar{y} | \mathbf{y}_{<t}). \quad (38f)$$

The identity of the sparsemax on $\Delta^{|\bar{\Sigma}|-1}$ then gives us

$$p_{\mathcal{R}}(\bar{y} | \mathbf{y}_{\leq t}) \stackrel{\text{def}}{=} \text{sparsemax}(\mathbf{E} \mathbf{h}(\mathbf{y}_{\leq t}))_{\bar{y}} \quad (39a)$$

$$= p_{\mathcal{A}}(\bar{y} | \mathbf{y}_{\leq t}) \quad (39b)$$

for all $\mathbf{y}_{<t} \in \Sigma^*$, $\bar{y} \in \bar{\Sigma}$. This implies by Eq. (1) that $p_{\mathcal{R}}$ and $p_{\mathcal{A}}$ are weakly equivalent. \blacksquare

Corollary 3.2. *Let $\mathcal{A} = (\Sigma, Q, \tau, \lambda, \rho)$ be a trim PFSA inducing the tight LM $p_{\mathcal{A}}$. Then, there exists a weakly equivalent, softmax-normalized non-linear output Elman LM with an $\bar{\mathbb{R}}$ -valued \mathbf{F} , where $\bar{\mathbb{R}} \stackrel{\text{def}}{=} \mathbb{R} \cup \{-\infty, \infty\}$ are the extended reals.*

Proof. Like in the proof of Cor. 3.1, we construct the function \mathbf{F} that transforms the hidden state \mathbf{h}_{t-1} into $|\bar{\Sigma}|$ values that are later normalized into $p_{\mathcal{A}}(y_t | \mathbf{h}_t)$, this time with the softmax. Concretely, we want to compute the autoregressive next-symbol probabilities $p_{\mathcal{A}}(y_t | \mathbf{y}_{<t})$. Assuming that all values are positive (we relax this later), we can write:

$$p_{\mathcal{A}}(\bar{y}_t | \mathbf{y}_{<t}) = \frac{p_{\mathcal{A}}(\mathbf{y}_{<t} \bar{y}_t)}{p_{\mathcal{A}}(\mathbf{y}_{<t})} \quad (40a)$$

$$= \frac{\sum_{q \in Q} p_{\mathcal{A}}(q, \mathbf{y}_{<t} \bar{y}_t)}{\sum_{q \in Q} p_{\mathcal{A}}(q, \mathbf{y}_{<t})} \quad (40b)$$

$$= \frac{\sum_{q \in Q} p_{\mathcal{A}}(q, \mathbf{y}_{<t}) p_{\mathcal{A}}(\bar{y}_t | q, \mathbf{y}_{<t})}{\sum_{q \in Q} p_{\mathcal{A}}(q, \mathbf{y}_{<t})} \quad (40c)$$

$$= \frac{\sum_{q \in Q} p_{\mathcal{A}}(q, \mathbf{y}_{<t}) p_{\mathcal{A}}(\bar{y}_t | q)}{\sum_{q \in Q} p_{\mathcal{A}}(q, \mathbf{y}_{<t})} \quad (40d)$$

$$= \exp \left(\log \left(\sum_{q \in Q} p_{\mathcal{A}}(q, \mathbf{y}_{<t}) p_{\mathcal{A}}(\bar{y}_t | q) \right) - \log \left(\sum_{q \in Q} p_{\mathcal{A}}(q, \mathbf{y}_{<t}) \right) \right). \quad (40e)$$

Intuitively, $p_{\mathcal{A}}(y_t | \mathbf{y}_{<t})$ can thus be computed by normalizing the values $\log \left(\sum_{q \in Q} p_{\mathcal{A}}(q, \mathbf{y}_{<t}) p_{\mathcal{A}}(\bar{y}_t | q) \right) - \log \left(\sum_{q \in Q} p_{\mathcal{A}}(q, \mathbf{y}_{<t}) \right)$ for $\bar{y}_t \in \bar{\Sigma}$ with the softmax. Defining the vector $\mathbf{v}_{\bar{y}} \stackrel{\text{def}}{=} \log \left(\sum_{q \in Q} p_{\mathcal{A}}(q, \mathbf{y}_{<t}) p_{\mathcal{A}}(\bar{y}_t | q) \right) - \log \left(\sum_{q \in Q} p_{\mathcal{A}}(q, \mathbf{y}_{<t}) \right)$ for $\bar{y}_t \in \bar{\Sigma}$, we therefore have

$$p_{\mathcal{A}}(y_t | \mathbf{y}_{<t}) = \sigma(\mathbf{v})_{\bar{y}}. \quad (41)$$

However, due to the invariance of the softmax to the addition of vectors of the form $c \cdot \mathbf{1}$, for any $c \in \mathbb{R}$, we also have that

$$p_{\mathcal{A}}(y_t | \mathbf{y}_{<t}) = \sigma \left(\mathbf{v} + \log \left(\sum_{q \in Q} p_{\mathcal{A}}(q, \mathbf{y}_{<t}) \right) \cdot \mathbf{1} \right)_{\bar{y}}. \quad (42)$$

Entries of $\mathbf{v}' \stackrel{\text{def}}{=} \mathbf{v} + \log \left(\sum_{q \in Q} p_{\mathcal{A}}(q, \mathbf{y}_{<t}) \right) \cdot \mathbf{1}$ equal

$$\mathbf{v}'_{\bar{y}} \stackrel{\text{def}}{=} \mathbf{v}_{\bar{y}} + \log \left(\sum_{q \in Q} p_{\mathcal{A}}(q, \mathbf{y}_{<t}) \right) = \log \left(\sum_{q \in Q} p_{\mathcal{A}}(q, \mathbf{y}_{<t}) p_{\mathcal{A}}(\bar{y}_t | q) \right). \quad (43)$$

The goal, then, is to compute the values $\sum_{q \in Q} p_{\mathcal{A}}(q, \mathbf{y}_{<t}) p_{\mathcal{A}}(\bar{y}_t | q)$, logarithmically transform them, and then normalize them with the softmax function. To achieve this, we take the same output matrix $\mathbf{E} \in \mathbb{R}^{|\Sigma| \times D}$ as in the proof of Cor. 3.1 (cf. Eq. (37)). The matrix–vector product $\mathbf{E}\mathbf{h}$ contains the values

$$(\mathbf{E}\mathbf{h})_y = \sum_{d=1}^D \mathbf{E}'_{\bar{y},d} \mathbf{h}(\mathbf{y}_{<t})_d \quad (44a)$$

$$= \sum_{q \in Q, y \in \Sigma} \mathbf{E}'_{\bar{y},(q,y)} \mathbf{h}(\mathbf{y}_{<t})_{(q,y)} \quad (44b)$$

$$= \sum_{q \in Q, y \in \Sigma} p_{\mathcal{A}}(\bar{y} | q) \mathbb{1}\{y = y_{t-1}\} p(q, \mathbf{y}_{<t}) \quad (44c)$$

$$= \sum_{q \in Q} p_{\mathcal{A}}(\bar{y} | q) p(q, \mathbf{y}_{<t}). \quad (44d)$$

These are the values we are interested in (from Eq. (40e)).

To allow for zero-valued arguments to the log, we now define the non-linear transformation $\overline{\log}$ as

$$\overline{\log}(\mathbf{x})_d \stackrel{\text{def}}{=} \begin{cases} \log(\mathbf{x})_d & \text{if } \mathbf{x}_d > 0 \\ -\infty & \text{otherwise} \end{cases}. \quad (45)$$

Note that the use of $-\infty$ here is what necessitates us to require the extended reals, in contrast to in Cor. 3.1. This motivates the definition of $\mathbf{F}: \mathbb{R}^{\mathbf{h}} \rightarrow \mathbb{R}^{|\Sigma|}$ as

$$\mathbf{F}(\mathbf{h}) \stackrel{\text{def}}{=} \overline{\log}(\mathbf{E}'\mathbf{h}). \quad (46)$$

The application of the softmax to $\mathbf{F}(\mathbf{h})$ results in

$$p_{\mathcal{R}}(\bar{y} | \mathbf{y}_{<t}) = \sigma(\mathbf{F}(\mathbf{h}))_{\bar{y}} \quad (47a)$$

$$= \frac{\exp(\overline{\log}(p_{\mathcal{A}}(\mathbf{y}_{<t}\bar{y})))}{\sum_{\bar{y}' \in \Sigma} \exp(\overline{\log}(p_{\mathcal{A}}(\mathbf{y}_{<t}\bar{y}')))} \quad (47b)$$

$$= \frac{p_{\mathcal{A}}(\mathbf{y}_{<t}\bar{y})}{\sum_{\bar{y}' \in \Sigma} p_{\mathcal{A}}(\mathbf{y}_{<t}\bar{y}')} \quad (47c)$$

$$= p_{\mathcal{A}}(\bar{y} | \mathbf{y}_{<t}), \quad (47d)$$

implying that $p_{\mathcal{A}}$ and $p_{\mathcal{R}}$ are weakly equivalent, as desired. ■

D Proof of Approximate Simulation by Elman LMs with Softmax over the Real Numbers

Recall that we assume, without loss of generality, that for each pair of states q, q' and each symbol $y \in \Sigma$ there exists a transition $q \xrightarrow{y/w} q'$, albeit potentially with zero weight, $w = 0$. For any PFSA, we define the following corresponding fully connected¹³ automaton:

Definition D.1. Given a PFSA $\mathcal{A} = (\Sigma, Q, \tau, \lambda, \rho)$ and $\delta \in \mathbb{R}$, \mathcal{A} 's δ -perturbed PFSA $\mathcal{A}_{\delta} = (\Sigma, Q, \tau_{\delta}, \lambda, \rho_{\delta})$ is a fully connected PFSA derived from \mathcal{A} by

- (1.) adding δ to all transition weights and final weights and
- (2.) locally re-normalizing the outgoing transition weights and the final weight of each state such that they sum to 1.

Algorithm 1 Conversion of a PFSA to a δ -perturbed PFSA

1. **def** δ -perturbed PFSA ($\mathcal{A} = (\Sigma, Q, \tau, \lambda, \rho), \delta$):
 2. $\tau_\delta \leftarrow \{\}$
 3. $\rho_\delta(q) \leftarrow 0 \quad \forall q \in Q$
 4. **for** $q \xrightarrow{y/w} q' \in \mathcal{A}$:
 5. $w_\delta \leftarrow \frac{w+\delta}{1+(\|\Sigma\||Q|+1)\delta}$
 6. Add transition $q \xrightarrow{y/w_\delta} q'$ to τ_δ .
 7. **for** $q \in Q$:
 8. $\rho_\delta(q) \leftarrow \frac{\rho(q)+\delta}{1+(\|\Sigma\||Q|+1)\delta}$
 9. $\mathcal{A}_\delta \leftarrow (\Sigma, Q, \tau_\delta, \lambda, \rho_\delta)$
 10. **return** \mathcal{A}_δ
-

See Alg. 1 for the pseudocode of transforming a PFSA to a δ -perturbed PFSA.

Definition D.2. Given two LMs p and q over Σ^* and a subset $S \subset \Sigma^*$, the **restricted TVD** on S is defined as

$$\text{TVD}_S(p, q) \stackrel{\text{def}}{=} \frac{1}{2} \sum_{\mathbf{y} \in S} |p(\mathbf{y}) - q(\mathbf{y})|. \quad (48)$$

Lemma D.1. Let $\mathcal{A} = (\Sigma, Q, \tau, \lambda, \rho)$ be a PFSA inducing the LM $p_{\mathcal{A}}$ and \mathcal{A}_δ \mathcal{A} 's δ -perturbed PFSA inducing the LM $p_{\mathcal{A}_\delta}$ for some $\delta \in \mathbb{R}$.¹⁴ For any finite subset of strings $S \subset \Sigma^*$, we have that the restricted total variation distance $\text{TVD}_S(p_{\mathcal{A}}, p_{\mathcal{A}_\delta})$ is continuous at $\delta = 0$.

Proof. Let $P(\mathbf{y})$ be the set of all paths in \mathcal{A} that accept \mathbf{y} . Because \mathcal{A} and \mathcal{A}_δ have the same topology, we will also use $P(\mathbf{y})$ for the set of all strings in \mathcal{A}_δ that accept \mathbf{y} . Note that $\varphi(\mathbf{p})$ is a finite set since both \mathcal{A} and \mathcal{A}_δ is a real-time PFSA. Now, to show that $\text{TVD}(p_{\mathcal{A}}, p_{\mathcal{A}_\delta})$ is continuous at $\delta = 0$, we need to show $\lim_{\delta \rightarrow 0} \text{TVD}_S(p_{\mathcal{A}}, p_{\mathcal{A}_\delta})$ exists. This follows directly from the following manipulations:

$$\lim_{\delta \rightarrow 0} \text{TVD}_S(p_{\mathcal{A}}, p_{\mathcal{A}_\delta}) = \lim_{\delta \rightarrow 0} \frac{1}{2} \sum_{\mathbf{y} \in S} |p_{\mathcal{A}}(\mathbf{y}) - p_{\mathcal{A}_\delta}(\mathbf{y})| \quad (49a)$$

$$= \lim_{\delta \rightarrow 0} \frac{1}{2} \sum_{\mathbf{y} \in S} \left| \sum_{\mathbf{p} \in P(\mathbf{y})} \prod_{n=0}^{|\mathbf{y}|-1} w_n - \sum_{\mathbf{p} \in P(\mathbf{y})} \prod_{n=0}^{|\mathbf{y}|-1} w_n^\delta \right| \quad (49b)$$

$$= \lim_{\delta \rightarrow 0} \frac{1}{2} \sum_{\mathbf{y} \in S} \left| \sum_{\mathbf{p} \in P(\mathbf{y})} \prod_{n=0}^{|\mathbf{y}|-1} w_n - \sum_{\mathbf{p} \in P(\mathbf{y})} \prod_{n=0}^{|\mathbf{y}|-1} \frac{w_n + \delta}{1 + \delta(\|\Sigma\||Q| + 1)} \right| \quad (49c)$$

$$= \frac{1}{2} \sum_{\mathbf{y} \in S} \left| \sum_{\mathbf{p} \in P(\mathbf{y})} \prod_{n=0}^{|\mathbf{y}|-1} w_n - \sum_{\mathbf{p} \in P(\mathbf{y})} \prod_{n=0}^{|\mathbf{y}|-1} \frac{w_n + \lim_{\delta \rightarrow 0} \delta}{1 + \lim_{\delta \rightarrow 0} \delta(\|\Sigma\||Q| + 1)} \right| \quad (49d)$$

$$= \frac{1}{2} \sum_{\mathbf{y} \in S} \left| \sum_{\mathbf{p} \in P(\mathbf{y})} \prod_{n=0}^{|\mathbf{y}|-1} w_n - \sum_{\mathbf{p} \in P(\mathbf{y})} \prod_{n=0}^{|\mathbf{y}|-1} \frac{w_n + 0}{1 + 0} \right| \quad (49e)$$

$$= \frac{1}{2} \sum_{\mathbf{y} \in S} \left| \sum_{\mathbf{p} \in P(\mathbf{y})} \prod_{n=0}^{|\mathbf{y}|-1} w_n - \sum_{\mathbf{p} \in P(\mathbf{y})} \prod_{n=0}^{|\mathbf{y}|-1} w_n \right| \quad (49f)$$

$$= 0, \quad (49g)$$

¹³Fully connected here means the PFSA has nonzero transition probability for all symbols between all pairs of strings, and nonzero final weights for all states.

¹⁴In this paper, we are only interested in small $\delta > 0$. However, the weighted automaton \mathcal{A}_δ is well defined for any $\delta \in \mathbb{R}$, albeit with negative weights in some cases. In the case of negative weights, \mathcal{A}_δ is no longer a *probabilistic* FSA.

where the step from Eq. (49d) to Eq. (49c) follows from the basic limit laws (continuous functions preserve limits), as desired. ■

Lemma D.2. *Let \mathcal{A} be a trim PFSA inducing the LM $p_{\mathcal{A}}$. For any $\epsilon > 0$, we can partition the set of all strings Σ^* into a finite set S and its complement $\Sigma^* \setminus S$ such that $p_{\mathcal{A}}(\Sigma^* \setminus S) = \sum_{\mathbf{y} \in \Sigma^* \setminus S} p_{\mathcal{A}}(\mathbf{y}) < \epsilon$.*

Proof. By Du et al. (2023, Thm 5.3), a PFSA induces a tight language model if and only if every accessible state is also co-accessible. Thus, because we assume \mathcal{A} is trim, the above-stated condition holds and $p_{\mathcal{A}}$ is tight. Moreover, because $p_{\mathcal{A}}$ is tight, we have

$$1 = \sum_{\mathbf{y} \in \Sigma^*} p_{\mathcal{A}}(\mathbf{y}) = \sum_{n=0}^{\infty} \sum_{\mathbf{y} \in \Sigma^n} p_{\mathcal{A}}(\mathbf{y}) \quad (50)$$

Since the infinite sum converges, we have that for any $\epsilon > 0$, there exists an $M < \infty$ such that

$$\sum_{n=M+1}^{\infty} \sum_{\mathbf{y} \in \Sigma^n} p_{\mathcal{A}}(\mathbf{y}) < \epsilon \quad (51)$$

and thus

$$\left| 1 - \sum_{n=0}^M \sum_{\mathbf{y} \in \Sigma^n} p_{\mathcal{A}}(\mathbf{y}) \right| < \epsilon. \quad (52)$$

Defining $S \stackrel{\text{def}}{=} \bigcup_{n=0}^M \Sigma^n$ completes the proof since $\sum_{n=0}^M \sum_{\mathbf{y} \in \Sigma^n} p_{\mathcal{A}}(\mathbf{y}) = \sum_{\mathbf{y} \in S} p_{\mathcal{A}}(\mathbf{y}) \geq 1 - \epsilon$ and $\sum_{\mathbf{y} \in \Sigma^* \setminus S} p_{\mathcal{A}}(\mathbf{y}) < \epsilon$. ■

Theorem D.1. *For any PFSA \mathcal{A} inducing the LM $p_{\mathcal{A}}$ and any $\epsilon > 0$, there exists a rational $\eta > 0$ such that $\text{TVD}(p_{\mathcal{A}}, p_{\mathcal{A}_\eta}) < \epsilon$, where $p_{\mathcal{A}_\eta}$ is the LM of \mathcal{A}_η , the η -perturbed \mathcal{A} .*

Proof. Let $\epsilon > 0$. Choose a finite set $S \subset \Sigma^*$ such that $p(S) > 1 - \frac{\epsilon}{2}$ and $p(\Sigma^* \setminus S) < \frac{\epsilon}{2}$. By Lem. D.2, we can always find such a set S . Next, by Lem. D.1, $\text{TVD}_S(p_{\mathcal{A}}, p_{\mathcal{A}_\delta}) : \mathbb{R} \rightarrow \mathbb{R}$ is continuous at $\delta = 0$. Thus, there exists a δ' such that $|\eta - 0| < \delta'$ implies $\text{TVD}_S(p_{\mathcal{A}}, p_{\mathcal{A}_\eta}) < \frac{\epsilon}{4}$. Choose a rational η positive such that $\eta < \delta'$. Next, we partition $\text{TVD}(p_{\mathcal{A}}, p_{\mathcal{A}_\eta})$ into three additive terms and bound each term individually

$$\text{TVD}(p_{\mathcal{A}}, p_{\mathcal{A}_\eta}) = \frac{1}{2} \sum_{\mathbf{y} \in \Sigma^*} |p_{\mathcal{A}}(\mathbf{y}) - p_{\mathcal{A}_\eta}(\mathbf{y})| \quad (53a)$$

$$= \frac{1}{2} \sum_{\mathbf{y} \in S} |p_{\mathcal{A}}(\mathbf{y}) - p_{\mathcal{A}_\eta}(\mathbf{y})| + \frac{1}{2} \sum_{\mathbf{y} \in \Sigma^* \setminus S} |p_{\mathcal{A}}(\mathbf{y}) - p_{\mathcal{A}_\eta}(\mathbf{y})| \quad (53b)$$

$$\leq \frac{1}{2} \sum_{\mathbf{y} \in S} |p_{\mathcal{A}}(\mathbf{y}) - p_{\mathcal{A}_\eta}(\mathbf{y})| + \frac{1}{2} \sum_{\mathbf{y} \in \Sigma^* \setminus S} |p_{\mathcal{A}}(\mathbf{y})| + \frac{1}{2} \sum_{\mathbf{y} \in \Sigma^* \setminus S} |p_{\mathcal{A}_\eta}(\mathbf{y})| \quad (53c)$$

$$= \underbrace{\frac{1}{2} \sum_{\mathbf{y} \in S} |p_{\mathcal{A}}(\mathbf{y}) - p_{\mathcal{A}_\eta}(\mathbf{y})|}_{< \epsilon/4} + \underbrace{\frac{1}{2} \sum_{\mathbf{y} \in \Sigma^* \setminus S} p_{\mathcal{A}}(\mathbf{y})}_{< \epsilon/2} + \underbrace{\frac{1}{2} \sum_{\mathbf{y} \in \Sigma^* \setminus S} p_{\mathcal{A}_\eta}(\mathbf{y})}_{< \epsilon} \quad (53d)$$

$$< \frac{1}{4}\epsilon + \frac{1}{4}\epsilon + \frac{1}{2}\epsilon \quad (53e)$$

$$= \epsilon, \quad (53f)$$

where, in the step from Eq. (53d) to Eq. (53e), the first two terms are bounded by the choice of S while the bound on the third term follows from the claim below. This proves the result. ■

Claim D.1. *We claim that $\epsilon > p_{\mathcal{A}_\eta}(\Sigma^* \setminus S)$.*

Proof. The proof follows from simple manipulations:

$$\frac{\epsilon}{4} > \text{TVD}_S(p_{\mathcal{A}}, p_{\mathcal{A}_\eta}) = \frac{1}{2} \sum_{\mathbf{y} \in S} |p_{\mathcal{A}}(\mathbf{y}) - p_{\mathcal{A}_\eta}(\mathbf{y})| \geq \frac{1}{2} \left| \sum_{\mathbf{y} \in S} (p_{\mathcal{A}}(\mathbf{y}) - p_{\mathcal{A}_\eta}(\mathbf{y})) \right| \quad (54a)$$

$$= \frac{1}{2} |p_{\mathcal{A}}(S) - p_{\mathcal{A}_\eta}(S)| = \frac{1}{2} |1 - p_{\mathcal{A}}(\Sigma^* \setminus S) - 1 - p_{\mathcal{A}_\eta}(\Sigma^* \setminus S)| \quad (54b)$$

$$= \frac{1}{2} |p_{\mathcal{A}}(\Sigma^* \setminus S) - p_{\mathcal{A}_\eta}(\Sigma^* \setminus S)|. \quad (54c)$$

This results in $\frac{\epsilon}{2} > |p_{\mathcal{A}}(\Sigma^* \setminus S) - p_{\mathcal{A}_\eta}(\Sigma^* \setminus S)|$. Additionally, we have $\frac{\epsilon}{2} > |p_{\mathcal{A}}(\Sigma^* \setminus S)|$. Summing these together, we arrive at

$$\epsilon > |p_{\mathcal{A}}(\Sigma^* \setminus S) - p_{\mathcal{A}_\eta}(\Sigma^* \setminus S)| + |p_{\mathcal{A}}(\Sigma^* \setminus S)| \quad (55a)$$

$$\geq |p_{\mathcal{A}}(\Sigma^* \setminus S) - p_{\mathcal{A}_\eta}(\Sigma^* \setminus S) + p_{\mathcal{A}}(\Sigma^* \setminus S)| = |p_{\mathcal{A}_\eta}(\Sigma^* \setminus S)| \quad (55b)$$

Thus, we arrive at $\epsilon > p_{\mathcal{A}_\eta}(\Sigma^* \setminus S)$. ■

Next, we define the notion of Lipschitz continuity.

Definition D.3. Let $\mathcal{K} \subseteq \mathbb{R}^N$ be a subset of \mathbb{R}^N . A mapping $f: \mathcal{K} \rightarrow \mathbb{R}^M$ is Lipschitz (or L -Lipschitz) with respect to a norm $\|\cdot\|_p$ if there exists a constant $L > 0$ such that for all $\mathbf{x}, \mathbf{y} \in \mathcal{K}$,

$$\|f(\mathbf{x}) - f(\mathbf{y})\|_2 \leq L \|\mathbf{x} - \mathbf{y}\|_2. \quad (56)$$

Proposition D.1 (Gao and Pavel (2018)). The softmax function σ is λ -Lipschitz with respect to $\|\cdot\|_2$ i.e., for all $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^N$,

$$\|\sigma(\mathbf{x}) - \sigma(\mathbf{x}')\|_2 \leq \lambda \|\mathbf{x} - \mathbf{x}'\|_2, \quad (57)$$

where λ is the inverse temperature parameter as defined in Eq. (12).

Softmax-normalized Elman LMs (cf. Def. 2.5) define the conditional next-symbol probabilities $p(y_t | \mathbf{y}_{<t})$ by normalizing $\mathbf{F}(\mathbf{h}(\mathbf{y}_{<t}))$. To abstract away the particular hidden states and their transformations, we now introduce the following definition.

Definition D.4. Let $\mathbf{y} \in \Sigma^*$ be a string of length T . Let $\mathbf{X} \in \mathbb{R}^{D \times (T+1)}$ be a matrix of $T+1$ column vectors of size D :

$$\mathbf{X} \stackrel{\text{def}}{=} \begin{bmatrix} | & | & \cdots & | \\ \mathbf{x}(1) & \mathbf{x}(2) & \cdots & \mathbf{x}(T+1) \\ | & | & & | \end{bmatrix}. \quad (58)$$

Define the function $f_{\mathbf{y}}: \mathbb{R}^{D \times (T+1)} \rightarrow [0, 1]$ that induces an autoregressive LM p :

$$f_{\mathbf{y}}(\mathbf{X}) = p(\text{EOS} | \mathbf{y}) \prod_{t=1}^T p(\bar{y}_t | \mathbf{y}_{<t}) \quad (59a)$$

$$= \sigma(\mathbf{x}(T+1))_{\text{EOS}} \prod_{t=1}^T \sigma(\mathbf{x}(t))_{\bar{y}_t} \quad (59b)$$

$$= \prod_{t=1}^{T+1} \sigma(\mathbf{x}(t))_{\bar{y}_t} \quad (59c)$$

$$\stackrel{\text{def}}{=} \prod_{t=1}^{T+1} \frac{\exp \mathbf{x}(t)_{\bar{y}_t}}{\sum_{\bar{y} \in \Sigma} \exp \mathbf{x}(t)_{\bar{y}}}, \quad (59d)$$

where $\bar{y}_{T+1} \stackrel{\text{def}}{=} \text{EOS}$.

Lemma D.3. The function $f_{\mathbf{y}}$ from Def. D.4 is Lipschitz continuous with respect to $\|\cdot\|_\infty$ with Lipschitz constant $L_{\mathbf{y}} = \mathcal{O}(|\mathbf{y}|)$, which, as the notation suggests, depends on the string \mathbf{y} .

Proof. The function $f_{\mathbf{y}}(\mathbf{X})$ is a product of softmax functions. Therefore, we divide the proof into two Steps. In Step 1, we start by showing that a single softmax σ is Lipschitz continuous with respect to $\|\cdot\|_\infty$. In Step 2, we show that products of softmax functions, e.g., $f_{\mathbf{y}}(\mathbf{X})$, are Lipschitz continuous by induction.

Step 1: Lipschitz continuity of σ . Prop. D.1 states that $\sigma: \mathbb{R}^N \rightarrow \mathbb{R}^N$ is Lipschitz with respect to the norm $\|\cdot\|_2$, with Lipschitz constant λ . This implies the following inequality

$$|\sigma(\mathbf{x})_n - \sigma(\mathbf{y})_n| \leq \lambda \|\mathbf{x} - \mathbf{y}\|_2 \quad \forall n \in \mathbb{N}. \quad (60)$$

We generalize this result to $\|\cdot\|_\infty$ (and, indeed, all norms on \mathbb{R}^N) by noting the equivalence of norms on finite-dimensional vector spaces, i.e., there exists a constant C such that $\|\mathbf{x}\|_2 \leq C \|\mathbf{x}\|_\infty$ for all $\mathbf{x} \in \mathbb{R}^N$. Thus, we get

$$|\sigma(\mathbf{x})_n - \sigma(\mathbf{y})_n| \leq \lambda \|\mathbf{x} - \mathbf{y}\|_2 \leq \underbrace{C\lambda}_{\stackrel{\text{def}}{=} L_\sigma} \|\mathbf{x} - \mathbf{y}\|_\infty. \quad (61)$$

Taking a max over the left-hand side of Eq. (61), we get

$$\|\sigma(\mathbf{x}) - \sigma(\mathbf{y})\|_\infty \leq L_\sigma \|\mathbf{x} - \mathbf{y}\|_\infty. \quad (62)$$

Step 2: Lipschitz continuity of $f_{\mathbf{y}}(\mathbf{X})$. The function $f_{\mathbf{y}}(\mathbf{X})$, as defined in Eq. (59c) is a product of $T + 1$ softmax functions. By definition, each component of the softmax σ_d for $d \in [D]$ has a bounded range $[0, 1]$. We now prove by induction that the product of T softmax functions is Lipschitz continuous. Define $f_N: \mathbb{R}^{D \times T+1} \rightarrow \mathbb{R}$ as $f_N(\mathbf{X}) \stackrel{\text{def}}{=} \prod_{t=1}^N \|\sigma(\mathbf{x}(t))\|_\infty$, where $N \leq T+1$ and $\mathbf{x}(t)$ is the t^{th} column vector of \mathbf{X} . Clearly, $f_T(\mathbf{X})$ upper bounds $f_{\mathbf{y}}(\mathbf{X})$.

- **Base case:** For $N = 1$, $f_1(\mathbf{X}) = \|\sigma(\mathbf{x}(1))\|_\infty$. Therefore,

$$|f_1(\mathbf{X}) - f_1(\mathbf{Y})| = \|\sigma(\mathbf{x}(1)) - \sigma(\mathbf{y}(1))\|_\infty \leq L_\sigma \|\mathbf{x}(1) - \mathbf{y}(1)\|, \quad (63)$$

where L_σ is the Lipschitz constant as shown in Step 1. Note that $\mathbf{x}(1)$ and $\mathbf{y}(1)$ are submatrices of \mathbf{X} and \mathbf{Y} , respectively, so $\|\mathbf{x}(1) - \mathbf{y}(1)\|_\infty \leq \|\mathbf{X} - \mathbf{Y}\|_\infty$. Therefore,

$$|f_1(\mathbf{X}) - f_1(\mathbf{Y})| \leq L_\sigma \|\mathbf{X} - \mathbf{Y}\|_\infty. \quad (64)$$

- **Induction step:** Assume that for N , $f_N(\mathbf{X}) = \prod_{t=1}^N \|\sigma(\mathbf{x}(t))\|_\infty$ is Lipschitz with constant NL_σ . We show that $f_{N+1}(\mathbf{X}) = \prod_{t=1}^{N+1} \|\sigma(\mathbf{x}(t))\|_\infty$ is Lipschitz with constant $(N+1)L_\sigma$.

Consider $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{D \times T+1}$. By the definition of f ,

$$|f_{N+1}(\mathbf{X}) - f_{N+1}(\mathbf{Y})| \quad (65a)$$

$$= |f_N(\mathbf{X}) \|\sigma(\mathbf{x}(N+1))\|_\infty - f_N(\mathbf{Y}) \|\sigma(\mathbf{y}(N+1))\|_\infty| \quad (65b)$$

$$= |f_N(\mathbf{X}) (\|\sigma(\mathbf{x}(N+1))\|_\infty - \|\sigma(\mathbf{y}(N+1))\|_\infty) + (f_N(\mathbf{X}) - f_N(\mathbf{Y})) \|\sigma(\mathbf{x}(N+1))\|_\infty| \quad (65c)$$

$$\leq |f_N(\mathbf{X})| |\|\sigma(\mathbf{x}(N+1))\|_\infty - \|\sigma(\mathbf{y}(N+1))\|_\infty| + |f_N(\mathbf{X}) - f_N(\mathbf{Y})| \|\sigma(\mathbf{x}(N+1))\|_\infty \quad (65d)$$

$$\leq |\|\sigma(\mathbf{x}(N+1))\|_\infty - \|\sigma(\mathbf{y}(N+1))\|_\infty| + |f_N(\mathbf{X}) - f_N(\mathbf{Y})| \quad (65e)$$

$$\leq |\|\sigma(\mathbf{x}(N+1)) - \sigma(\mathbf{y}(N+1))\|_\infty| + |f_N(\mathbf{X}) - f_N(\mathbf{Y})| \quad (65f)$$

$$\leq L_\sigma \|\mathbf{x}(N+1) - \mathbf{y}(N+1)\|_\infty + |f_N(\mathbf{X}) - f_N(\mathbf{Y})| \quad (65g)$$

$$\leq L_\sigma \|\mathbf{x}(N+1) - \mathbf{y}(N+1)\|_\infty + NL_\sigma \|\mathbf{X} - \mathbf{Y}\|_\infty \quad (65h)$$

$$\leq L_\sigma \|\mathbf{X} - \mathbf{Y}\|_\infty + NL_\sigma \|\mathbf{X} - \mathbf{Y}\|_\infty \quad (65i)$$

$$= (N+1)L_\sigma \|\mathbf{X} - \mathbf{Y}\|_\infty, \quad (65j)$$

where in Eq. (65e) we use the fact that $\|\sigma(\cdot)\|_\infty \leq 1$, and hence also $f_N(\mathbf{X}) \leq 1$ for all $N \in \mathbb{N}$. In Eq. (65f), we use the inverse triangle inequality, in Eq. (65g) we use the result from Step 1, and in Eq. (65h), we use the induction hypothesis. Finally, in Eq. (65i), we use the fact that the infinity norm of a matrix bounds the infinity norm of a subset of that matrix.

Thus, for $\mathbf{y} \in \Sigma^*$, we have $f_{\mathbf{y}}$ is Lipschitz continuous with Lipschitz constant $L_{\mathbf{y}} = |T+1|L_\sigma = |T+1|C\lambda = \mathcal{O}(|\mathbf{y}|)$. ■

Theorem 4.1. For any PFSA \mathcal{A} with induced LM $p_{\mathcal{A}}$ and any $\epsilon > 0$, there exists a ReLU-activated non-linear output Elman LM $p_{\mathcal{R}}$ with an \mathbb{R} -vector-valued function $\mathbf{F}: \mathbb{R}^D \rightarrow \mathbb{R}^D$ such that $\text{TVD}(p_{\mathcal{A}}, p_{\mathcal{R}}) < \epsilon$.

Proof. Let \mathcal{A} be a PFSA with induced LM $p_{\mathcal{A}}$, and be the LM induced by the \mathcal{A} and $\epsilon > 0$. By Thm. D.1, we can construct an η -perturbed PFSA \mathcal{A}_{η} inducing the full-support LM $p_{\mathcal{A}_{\eta}}$ such that $\text{TVD}(p_{\mathcal{A}}, p_{\mathcal{A}_{\eta}}) < \epsilon$. Using the same construction as that in the proof of Cor. 3.2, except that we replace $\overline{\log}$ in Eq. (45) with \log , we can construct a non-linear output Elman LM $p_{\mathcal{R}}$ (the Elman RNN identical to the one in Thm. 3.1) that is *weakly equivalent* to the full-support $p_{\mathcal{A}_{\eta}}$. This is because, since $p_{\mathcal{A}_{\eta}}$ has full support, all outputs of \mathbf{F} are positive, and for any $x > 0$, $\log(x) = \overline{\log}(x)$. We then have that

$$\text{TVD}(p_{\mathcal{A}}, p_{\mathcal{R}}) \leq \underbrace{\text{TVD}(p_{\mathcal{A}}, p_{\mathcal{A}_{\eta}})}_{< \epsilon} + \underbrace{\text{TVD}(p_{\mathcal{A}_{\eta}}, p_{\mathcal{R}})}_{=0} < \epsilon, \quad (66)$$

which finishes the proof. ■

E Proof of Approximate Simulation by Softmax-normalized, Deep Output Elman LMs

Theorem 4.2. Let $p_{\mathcal{A}}$ be a regular LM induced by a PFSA \mathcal{A} and let $\epsilon > 0$. Then, there exists a ReLU-activated, softmax-normalized deep output Elman LM $p_{\mathcal{R}}$ such that $\text{TVD}(p_{\mathcal{A}}, p_{\mathcal{R}}) < \epsilon$.

Proof. Fix $\epsilon > 0$ and let $p_{\mathcal{A}_{\delta}}$ be the language model induced from a δ -perturbation of a PFSA \mathcal{A} weakly equivalent to $p_{\mathcal{A}}$; we will specify δ at a later point in the proof. Our goal is to show that there exists a ReLU-activated, softmax-normalized deep output Elman LM $p_{\mathcal{R}}$ such that $\text{TVD}(p_{\mathcal{A}}, p_{\mathcal{R}}) = \frac{1}{2} \sum_{\mathbf{y} \in \Sigma^*} |p_{\mathcal{A}}(\mathbf{y}) - p_{\mathcal{R}}(\mathbf{y})|$. The RNN \mathcal{R} will be the same as the one in Thm. 3.1; we only seek to approximate the *output* function of the Elman LM. To do so, we first partition $\text{TVD}(p_{\mathcal{A}}, p_{\mathcal{R}})$ into four additive terms and bound each term individually:

$$\text{TVD}(p_{\mathcal{A}}, p_{\mathcal{R}}) = \frac{1}{2} \sum_{\mathbf{y} \in \Sigma^*} |p_{\mathcal{A}}(\mathbf{y}) - p_{\mathcal{R}}(\mathbf{y})| \quad (67a)$$

$$= \frac{1}{2} \sum_{\mathbf{y} \in \Sigma^*} |p_{\mathcal{A}}(\mathbf{y}) - p_{\mathcal{A}_{\delta}}(\mathbf{y}) + p_{\mathcal{A}_{\delta}}(\mathbf{y}) - p_{\mathcal{R}}(\mathbf{y})| \quad (67b)$$

$$\leq \frac{1}{2} \sum_{\mathbf{y} \in \Sigma^*} |p_{\mathcal{A}}(\mathbf{y}) - p_{\mathcal{A}_{\delta}}(\mathbf{y})| + \frac{1}{2} \sum_{\mathbf{y} \in \Sigma^*} |p_{\mathcal{A}_{\delta}}(\mathbf{y}) - p_{\mathcal{R}}(\mathbf{y})| \quad (67c)$$

$$= \frac{1}{2} \sum_{\mathbf{y} \in \Sigma^*} |p_{\mathcal{A}}(\mathbf{y}) - p_{\mathcal{A}_{\delta}}(\mathbf{y})| + \frac{1}{2} \sum_{\mathbf{y} \in S} |p_{\mathcal{A}_{\delta}}(\mathbf{y}) - p_{\mathcal{R}}(\mathbf{y})| + \frac{1}{2} \sum_{\mathbf{y} \in \Sigma^* \setminus S} |p_{\mathcal{A}_{\delta}}(\mathbf{y}) - p_{\mathcal{R}}(\mathbf{y})| \quad (67d)$$

$$\leq \underbrace{\frac{1}{2} \sum_{\mathbf{y} \in \Sigma^*} |p_{\mathcal{A}}(\mathbf{y}) - p_{\mathcal{A}_{\delta}}(\mathbf{y})|}_{< \frac{\epsilon}{4} \text{ by Step 1}} + \underbrace{\frac{1}{2} \sum_{\mathbf{y} \in S} |p_{\mathcal{A}_{\delta}}(\mathbf{y}) - p_{\mathcal{R}}(\mathbf{y})|}_{< \frac{\epsilon}{4} \text{ by Step 2}} + \underbrace{\frac{1}{2} \sum_{\mathbf{y} \in \Sigma^* \setminus S} |p_{\mathcal{A}_{\delta}}(\mathbf{y})|}_{< \frac{\epsilon}{4} \text{ by Step 3}} + \underbrace{\frac{1}{2} \sum_{\mathbf{y} \in \Sigma^* \setminus S} |p_{\mathcal{R}}(\mathbf{y})|}_{< \frac{\epsilon}{4} \text{ by Step 4}} < \epsilon. \quad (67e)$$

The bounds are shown in the corresponding steps below.

Caution: Steps 1, 2, 3, and 4 all depend on δ , and Steps 2, 3, and 4 all depend on S . To decouple the individual steps (for expository purposes), in each step, we will choose a separate δ_i and then assign $\delta = \min(\delta_1, \delta_2, \delta_3, \delta_4)$ and a separate set S_i and then assign $S = S_2 \cup S_3 \cup S_4$.

Step 1. By Thm. D.1, we can choose a δ_1 such that $\frac{1}{2} \sum_{\mathbf{y} \in \Sigma^*} |p_{\mathcal{A}}(\mathbf{y}) - p_{\mathcal{A}_{\delta_1}}(\mathbf{y})| < \frac{\epsilon}{4}$.

Step 2. Let S_2 be any finite subset of Σ^* and choose $\delta_2 \in (0, 1)$ arbitrarily. We first note that the δ_2 -perturbed language model $p_{\mathcal{A}_{\delta_2}}$ can be written as follows

$$p_{\mathcal{A}_{\delta_2}}(\mathbf{y}) = p_{\mathcal{A}_{\delta_2}}(\text{EOS} \mid \mathbf{y}) \prod_{t=1}^T p_{\mathcal{A}_{\delta_2}}(y_t \mid \mathbf{y}_{<t}) \quad (68a)$$

$$= \sigma(\log(\mathbf{Eh}(\mathbf{y})))_{\text{EOS}} \prod_{t=1}^T \sigma(\log(\mathbf{Eh}(\mathbf{y}_{<t})))_{y_t}. \quad (68b)$$

Moreover, since $p_{\mathcal{A}_{\delta_2}}$ is induced from an δ_2 -perturbed PFSA, $\mathbf{h}(\mathbf{y})_q > 0$ for all $q \in Q$ and all $\mathbf{y} \in \Sigma^*$. Next, define the following constants

$$\xi_1 \stackrel{\text{def}}{=} \min_{\mathbf{y} \in S_2} \min_{q \in Q} \mathbf{h}(\mathbf{y})_q > 0 \quad (69a)$$

$$\xi_2 \stackrel{\text{def}}{=} \min_{\mathbf{y} \in S_2} \min_{\bar{y} \in \bar{\Sigma}} \log(\mathbf{Eh}(\mathbf{y}))_{\bar{y}} < 0 \quad (69b)$$

Finally, we will define

$$L = \max_{\mathbf{y} \in S_2} L_{\mathbf{y}} \quad (70)$$

where $L_{\mathbf{y}}$ is the Lipschitz constant of the function $f_{\mathbf{y}}$ (cf. Def. D.4) given in Lem. D.3. Our goal is to find a ReLU MLP \mathbf{F} that is an arbitrarily good approximator of $\mathbf{h} \mapsto \log(\mathbf{Eh}): [\xi_1, 1]^{|Q|} \rightarrow [\xi_2, 0]^{\bar{\Sigma}}$ where the log is applied element-wise. Because $[\xi_1, 1]^{|Q|} \ni \mathbf{h} \mapsto \log(\mathbf{Eh})$ is continuous and its domain is compact, Thm. 2.1 tells us there does exist a one-layer MLP \mathbf{F}_1 such that the following bound holds

$$\max_{\mathbf{h} \in [\xi_1, 1]^{|Q|}} \|\log(\mathbf{Eh}) - \mathbf{F}_1(\mathbf{h})\|_{\infty} < \frac{\epsilon}{2L|S|} \quad (71)$$

In order to make sure that the one-layer MLP \mathbf{F} only receives inputs in its domain $[\xi_1, 1]^{|Q|}$, we compose \mathbf{F}_1 with another ReLU layer \mathbf{F}_2 defined element-wise as follows

$$\mathbf{F}_2(\mathbf{h})_i \stackrel{\text{def}}{=} \text{ReLU}(0, \mathbf{h}_i - \xi_1) + \xi_1 \quad (72)$$

which computes the function $\max(\xi_1, x)$. Because of the definition of ξ_i in Eq. (69a), the composition of \mathbf{F}_1 with \mathbf{F}_2 has no effect on arguments taken from the set $\{\mathbf{h}(\mathbf{y}) \mid \mathbf{y} \in S_2\}$. Now, define $\mathbf{F}(\mathbf{h}) \stackrel{\text{def}}{=} \mathbf{F}_1(\mathbf{F}_2(\mathbf{h}))$. Then, making use of the finiteness of S_2 , for any $\mathbf{y} \in S_2$, we have that $f_{\mathbf{y}}$ is $L_{\mathbf{y}}$ -Lipschitz continuous and a fortiori L -Lipschitz continuous by Lem. D.1. Combining the result with the Eq. (71), we have that

$$|f_{\mathbf{y}}(\log \mathbf{Eh}(\mathbf{y})) - f_{\mathbf{y}}(\mathbf{F}(\mathbf{h}(\mathbf{y})))| < \frac{\epsilon}{2L|S_2|} L = \frac{\epsilon}{2|S_2|} \quad (73)$$

which, with no more than an adjustment of notation (see Def. D.4), tells us that

$$|p_{\mathcal{A}_{\delta_2}}(\mathbf{y}) - p_{\mathcal{R}}(\mathbf{y})| < \frac{\epsilon}{2L|S_2|} L = \frac{\epsilon}{2|S_2|} \quad (74)$$

Because we chose a finite set S_2 of strings, we arrive at the following bound

$$\frac{1}{2} \sum_{\mathbf{y} \in S_2} |p_{\mathcal{A}_{\delta_2}}(\mathbf{y}) - p_{\mathcal{R}}(\mathbf{y})| < \frac{1}{2} \sum_{\mathbf{y} \in S_2} \frac{\epsilon}{|S_2|} = \frac{1}{2} |S_2| \frac{\epsilon}{|S_2|} = \frac{\epsilon}{4} \quad (75)$$

as desired.

Step 3. Again by Thm. D.1, we can choose δ_3 such that $\frac{1}{2} \sum_{\mathbf{y} \in \Sigma^*} |p_{\mathcal{A}}(\mathbf{y}) - p_{\mathcal{A}_{\delta_3}}(\mathbf{y})| < \frac{\epsilon}{8}$. The proof of Thm. D.1 yields a set S_3 such that $p_{\mathcal{A}}(S_3) = 1 - \frac{\epsilon}{16}$. Then, we can apply Claim D.1, which says $\frac{\epsilon}{2} > p_{\mathcal{A}_{\delta_3}}(\Sigma^* \setminus S_3)$.

Step 4. By Step 2, for any finite set $S_4 \subset \Sigma^*$, we can choose a $\delta_4 > 0$ such that $\frac{\epsilon}{8} > \frac{1}{2} \sum_{\mathbf{y} \in S_4} |p_{\mathcal{A}_{\delta_4}}(\mathbf{y}) - p_{\mathcal{R}}(\mathbf{y})|$. Then, consider the following chain of inequalities

$$\frac{\epsilon}{8} > \frac{1}{2} \sum_{\mathbf{y} \in S_4} |p_{\mathcal{A}_{\delta_4}}(\mathbf{y}) - p_{\mathcal{R}}(\mathbf{y})| \quad (76a)$$

$$\geq \frac{1}{2} \left| \sum_{\mathbf{y} \in S_4} (p_{\mathcal{A}_{\delta_4}}(\mathbf{y}) - p_{\mathcal{R}}(\mathbf{y})) \right| \quad (76b)$$

$$= \frac{1}{2} |p_{\mathcal{A}_{\delta_4}}(S_4) - p_{\mathcal{R}}(S_4)| \quad (76c)$$

$$= \frac{1}{2} |1 - p_{\mathcal{A}_{\delta_4}}(\Sigma^* \setminus S_4) - (1 - p_{\mathcal{R}}(\Sigma^* \setminus S_4))| \quad (76d)$$

$$= \frac{1}{2} |p_{\mathcal{A}_{\delta_4}}(\Sigma^* \setminus S_4) - p_{\mathcal{R}}(\Sigma^* \setminus S_4)|. \quad (76e)$$

This results in $\frac{\epsilon}{4} > |p_{\mathcal{A}_{\delta_4}}(\Sigma^* \setminus S_4) - p_{\mathcal{R}}(\Sigma^* \setminus S_4)|$.

So far, we only required S_4 to be finite but did not fix its value. Now, by Claim D.1, we can choose S_4 such that we have $\frac{\epsilon}{4} > |p_{\mathcal{A}_{\delta_4}}(\Sigma^* \setminus S_4)|$. Summing both of these together, we arrive at

$$\frac{\epsilon}{2} > |p_{\mathcal{A}_{\delta_4}}(\Sigma^* \setminus S_4) - p_{\mathcal{R}}(\Sigma^* \setminus S_4)| + |p_{\mathcal{A}_{\delta_4}}(\Sigma^* \setminus S_4)| \quad (77a)$$

$$\geq |p_{\mathcal{A}_{\delta_4}}(\Sigma^* \setminus S_4) - p_{\mathcal{R}}(\Sigma^* \setminus S_4) + p_{\mathcal{A}_{\delta_4}}(\Sigma^* \setminus S_4)| = |p_{\mathcal{R}}(\Sigma^* \setminus S_4)|. \quad (77b)$$

Thus, we have $\frac{\epsilon}{2} > p_{\mathcal{R}}(\Sigma^* \setminus S_4)$ because probabilities are non-negative. ■