What Languages are Easy to Language-Model? A Perspective from Learning Probabilistic Regular Languages

Anonymous ACL submission

Abstract

What can large language models learn? By definition, language models (LM) are distributions over strings. Therefore, an intuitive way of addressing the above question is to formalize it as a matter of learnability of classes of distributions over strings. While prior work in this direction focused on assessing the theoretical limits, we seek to understand the empirical learnability. Unlike prior empirical work, we evaluate LMs on their home ground—learning probability distributions over strings—rather than as classifiers of formal languages. In particular, we investigate the learnability of finite-state LMs (FSLMs). We first theoretically quantify the minimal representation size of a neural LM necessary for learning an FSLM in terms of its *rank*, which corresponds to the size of linear space spanned by the logits of its conditional distributions. We then empirically test the learnability of FSLMs and find that the rank is a strong predictor of learnability for both Transformers and RNNs, but the significance of other properties of the FSLM differs between Transformers and RNNs.

1 Introduction

Language models are, definitionally, distributions over strings. However, not all neural LMs are capable of learning or even representing all possible distributions. This raises two natural questions: What classes *can* neural LMs represent and what can they learn from training examples? In terms of the first question, which distributions over strings recurrent neural LMs can encode has been subject to study for over three decades (e.g., McCulloch and Pitts, 1943; Kleene, 1956; Siegelmann and Sontag, 1992; Hao et al., 2018; Korsky and Berwick, 2019; Merrill, 2019; Merrill et al., 2020; Hewitt et al., 2020; Chung and Siegelmann, 2021; Merrill et al., 2022; Merrill and Tsilivis, 2022; Svete and Cotterell, 2023; Nowak et al., 2023). Moreover, the prevalence of Transformer-based LMs has led to a recent body of work investigating their representational capacity (e.g., Hahn, 2020; Ebrahimi et al., 2020; Bhattamishra et al., 2020; Merrill and Sabharwal, 2023). However, almost all of this work is theoretical, i.e., researchers seek theorems that give exact limitations for the capacity of specific neural LMs. While such work provides a good characterization of what neural LMs could, in principle, learn, it does not speak to what LMs can learn in practice.

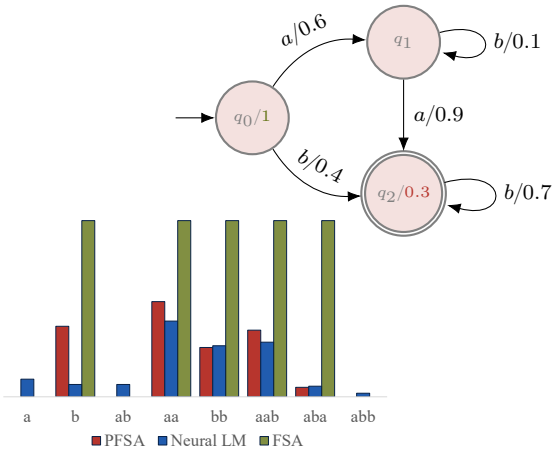

Figure 1: A finite-state automaton (an unweighted version of the one shown here) defines a *set* of strings by assigning string binary weights. A probabilistic finite-state automaton such as the one in the figure and a neural LM such as an RNN or a Transformer LM, however, define a *probability distribution* over strings.

In contrast to a more theoretically minded researcher, an empirically minded researcher might prefer to run a series of controlled experiments. Their goal is to empirically characterize what classes of formal LMs, e.g., probabilistic finite-state automata, neural LMs *are* able to learn in practice. Such work informs our understanding of what types of languages larger LMs trained on human-written text might represent—specifically, what grammatical structures they can recognize, and how efficiently they can do so. All of the above is crucial for quantifying the practical capa-

bilities, and limits, of neural LMs. While plenty of empirical work has provided insights into the linguistic capabilities of modern LMs (e.g., Linzen et al., 2016; Hewitt and Manning, 2019; Jawahar et al., 2019; Liu et al., 2019; Icard, 2020; Manning et al., 2020; Rogers et al., 2021; Belinkov, 2022), real-world datasets give us limited insight into the types of distributions a neural LM can learn because the true distribution that the neural LMs is modeling is often unclear. For instance, fitting an LM to Wikipedia leaves it open to interpretation exactly *which* probability distribution over strings the neural LM is modeling. In contrast, learning a probabilistic formal language in a controlled situation offers an unparalleled level of control.

A close look at existing work testing the representation and empirical learnability of formal languages (see App. A for an overview) reveals a categorical mismatch between what LMs are, i.e., *probability distributions* over strings, and what learning a formal language means, i.e., classifying strings as members of a specific language, i.e., a *set* of strings (Ebrahimi et al., 2020; Deletang et al., 2023; Wang and Steinert-Threlkeld, 2023). Prior work has also benchmarked the learnability of non-probabilistic finite-state transducers by sequence-to-sequence models (Valvoda et al., 2022). We propose to investigate the practical representation capacity of neural LMs by testing their ability to learn *distributions* over strings. By sampling languages from probabilistic finite-state automata (PFSA) LMs and training neural LMs on them, we can ask precise questions about the learnability.

Our paper contributes a large empirical study, sampling datasets of 20k strings from 6500 randomly generated PFSA, and training Transformer and RNN LM models with a varying hidden state size on our datasets. The empirical study is informed by various theoretical results regarding the representational capacity of RNNs concerning probabilistic finite-state automata. We assess the learnability by approximating the KL divergence between neural LMs and PFSA. We find that a large number of properties of the automaton, e.g., the number of states, the number of transitions, the rank of its emission matrix, and its entropy contribute to learnability. However, no one factor appears to be decisively more predictive than another. Moreover, similar to Deletang et al. (2023), we also find that RNNs are suited to modeling formal languages, in comparison to Transformers, which necessitate language-specific hyperparameter tuning.

2 Preliminaries

We begin with an introduction of the relevant mathematical preliminaries.

Definition 2.1. *An **alphabet** Σ is a finite, non-empty set of **symbols**. Its **Kleene closure** Σ^* is the set of all strings of its symbols. The **length** of the string $\boldsymbol{y} = y_1 \dots y_T \in \Sigma^*$, denoted by $|\boldsymbol{y}| = T$, is the number of symbols it contains.*

Definition 2.2. *A **language model** p is a probability distribution over Σ^*. Two LMs p and q are **weakly equivalent** if $p(\boldsymbol{y}) = q(\boldsymbol{y})$ for all $\boldsymbol{y} \in \Sigma^*$.*

Most modern LMs define $p(\boldsymbol{y})$ as a product of conditional probability distributions:

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

where $\text{EOS} \notin \Sigma$ is a special end of sequence symbol. We denote $\overline{\Sigma} \stackrel{\text{def}}{=} \Sigma \cup \{\text{EOS}\}$.

2.1 Neural Language Models

Neural LMs define the conditional distributions $p(y_t \mid \boldsymbol{y}_{<t})$ through a linearly transformed and softmax-normalized **hidden state** $\mathbf{h}_{t-1} \in \mathbb{R}^D$—a representation of the string $\boldsymbol{y}_{<t}$—that is computed by a neural network:

$$p(y_t \mid \boldsymbol{y}_{<t}) \stackrel{\text{def}}{=} \text{softmax}(\mathbf{E}\mathbf{h}_{t-1})_{y_t} \qquad (2)$$

$$\stackrel{\text{def}}{=} \frac{\exp(\mathbf{E}\mathbf{h}_{t-1})_{y_t}}{\sum_{y \in \overline{\Sigma}} \exp(\mathbf{E}\mathbf{h}_{t-1})_y} \qquad (3)$$

We will call $\mathbf{E} \in \mathbb{R}^{|\overline{\Sigma}| \times D}$ the **output matrix**.

Neural LMs differ in how $\mathbf{h}_{t-1}$ is computed based on $\boldsymbol{y}_{<t}$. In this paper, we consider the two most popular modern language modeling architectures: Transformers (Vaswani et al., 2017), where $\mathbf{h}_{t-1}$ is computed with self-attention, and recurrent neural networks (Elman, 1990) (specifically the LSTM variant (Hochreiter and Schmidhuber, 1997)), where $\mathbf{h}_t$ is computed recurrently.

2.2 Finite-state Language Models

A classic formalism for defining LMs is **probabilistic finite-state automata** (PFSAs), a probabilistic version of finite-state automata that defines string probabilities. Intuitively, a PFSA defines a *finite* number of conditional next-symbol distributions $p(y \mid q)$ based on a finite number of states $q \in Q$

that summarize string prefixes analogous to how the hidden state $\mathbf{h}_t$ of an RNN summarizes the prefix $y_1 \ldots y_t$. A PFSA moves between its states based on the input symbols according to the transitions defined by a transition relation. It **accepts** a string with the probability equal to the product of the transition weights along the string's path in the automaton and the last state's final weight (or the sum over all paths if there are multiple paths accepting the string).[1] It is **deterministic** (a DPFSA) if the transition relation is a *function* of the current state and symbol—if, for all $q \in Q$ and $y \in \Sigma$, there exists at most one $q' \in Q$ such that $p(q' \mid q, y) > 0$. A PFSA is **minimal** if it has the smallest number of states among all its weakly equivalent PFSAs.[2] The minimal DPFSA is unique up to a renaming of the states. This gives the distribution encoded by a DPFSA a distinct canonical distribution.

Definition 2.3. *The LM p is **finite-state** (an FSLM) if there exists a weakly equivalent PFSA.*

Fig. 1 shows an example of a FSLM defining a distribution over $\{a, b\}^*$ with $p(ab^n ab^m) = 1 \cdot 0.6 \cdot 0.1^n \cdot 0.9 \cdot 0.7^m \cdot 0.3$ and $p(bb^m) = 1 \cdot 0.4 \cdot 0.7^m \cdot 0.3$.

3 Representing FSLMs with Neural LMs

Neural LMs have demonstrated an ability to model human language well. However, they are notoriously challenging to analyze, making it difficult to state any formal claims on what they are (in)capable of. To amend this, a large body of work has linked neural LMs to formal models of computation. DPFSAs feature particularly often in this line of research (Merrill, 2019; Merrill et al., 2020; Svete and Cotterell, 2023). To facilitate a detailed inspection of how neural LMs can represent DPFSAs, we now formalize DPFSAs in a way that is particularly easy to connect to neural LMs.

The conditional distributions $p(y \mid q)$ defined by a DPFSA can, in general, be arbitrary distributions over $\overline{\Sigma}$—a DPFSA therefore defines $|Q|$ distributions, each with $|\overline{\Sigma}| - 1$ degrees of freedom. As we will see, such a parameterization makes the connection to neural LMs—which define conditional distributions in terms of shared parameters of the neural network and the output matrix $\mathbf{E}$—somewhat tricky. To facilitate a formal connection, we define parametrized PFSAs.

[1] Final weights of states are analogous to the EOS symbol which signals the end of string generation in neural LMs.

[2] PFSAs and their relationships to LMs are discussed in more detail in App. B.

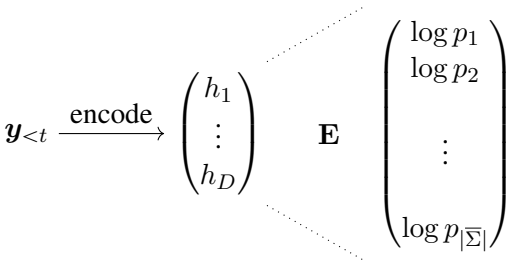

Figure 2: Linearly transforming $\mathbf{h}$ using $\mathbf{E}$ results in vector of logits $\in \mathbb{R}^{|\overline{\Sigma}|}$. $\mathbf{Eh}$, however, defines at most D-dimensional subspace in $\mathbb{R}^{|\overline{\Sigma}|}$.

Definition 3.1. *Let Σ be an alphabet, $\mathcal{A}$ a PFSA with states Q, and $\mathbf{T} \in \mathbb{R}^{(|\Sigma|+1) \times |Q|}$ a matrix of rank R. We say that $\mathcal{A}$ is **rank** R if $p(y \mid q) = \mathrm{softmax}(\mathbf{T}_{:,q})_y$ for all $q \in Q$ and $y \in \overline{\Sigma}$.*[3]

Our study focuses on *deterministic* PFSAs, whose relationship to neural LMs is better understood. Let $\mathcal{A}$ be a rank-R DPFSA with states Q over the alphabet Σ and p a neural LM. Further, let $\boldsymbol{y} \in \Sigma^*$, $q \in Q$ the state reached by $\mathcal{A}$ when reading $\boldsymbol{y}$, and $\mathbf{h}$ the hidden state encoding $\boldsymbol{y}$ by p. If we want the LM p to match $\mathcal{A}$'s distribution (for them to be weakly equivalent), it has to hold that

$$\mathrm{softmax}(\mathbf{Eh})_y = \mathrm{softmax}(\mathbf{T}_{:,q})_y \tag{4}$$

for all $y \in \overline{\Sigma}$. Due to the additive invariance property of the softmax function, this is equivalent to

$$\mathbf{Eh} = \underbrace{\mathbf{T}_{:,q} + \mathbf{c}_q}_{\stackrel{\text{def}}{=} \mathbf{u}_q} \tag{5}$$

where $\mathbf{c}_q = c_q \mathbf{1} \in \mathbb{R}^{|\overline{\Sigma}|}$ and $c_q \in \mathbb{R}$. We define $\mathbf{U} \in \mathbb{R}^{|\overline{\Sigma}| \times |Q|}$ as the matrix with columns $\mathbf{u}_q$.

Minimal representation size. Assuming $|\overline{\Sigma}| \leqslant |Q|$, the columns of $\mathbf{U}$ will, in general, span $\mathbb{R}^{|\overline{\Sigma}|}$. In the special case of rank-R DPFSAs, they will span at most a $(\mathrm{R}+1)$-dimensional subspace of $\mathbb{R}^{|\overline{\Sigma}|}$. For Eq. (5) to hold, it is, therefore, necessary that $D \geqslant \mathrm{R} + 1$. If that is not the case, the neural LM will naturally *not* be able to match all the conditional distributions defined by the states of the DPFSA, which leads us to the following theorem.

Theorem 3.1. *Let p be a finite-state LM and that can be represented by a rank-R DPFSA $\mathcal{A}$ which is minimal for p. Let q be a neural LM with a hidden state $\mathbf{h}$. If p and q are weakly equivalent, then $\mathbf{h}$ must in general be of size at least $\mathrm{R} + 1$.*

[3] $\mathbf{T}_{:,q}$ is the column of the matrix $\mathbf{T}$ corresponding to q.

Given a rank-R DPFSA $\mathcal{A}$, Thm. 3.1 says that a weakly equivalent neural LM needs a hidden state of size at least $\mathrm{R}+1$, establishing a general lower bound on an LM's hidden state size for weak equivalence with a DPFSA. Note that a hidden state of size $\mathrm{R}+1$, however, does not mean that the neural LM can implement the *transitions dynamics* of the PFSA with a hidden state of size $\mathrm{R}+1$—it does not guarantee that the LM is capable of implementing the transitions between the subsequent hidden states capturing the individual states of the automaton. Indeed, the size of the hidden state must also, in some cases, scale linearly with the number of states.

Theorem 3.2 (Svete and Cotterell (2023), Thms. 5.1 and 5.2)**.** *There exist families of DPFSAs such that the representation size of any weakly equivalent finite-precision RNN LM must scale linearly with $|Q|$ and $|\Sigma|$.*

Beyond representational capacity. We see that the representational capacity of neural LMs can be theoretically described relatively comprehensively in terms of formal models of computation. However, existing theoretical work only considers the question of which distributions can be *represented* by a neural LM. This leaves us with a large gap in understanding what distributions are *learnable* by neural LMs. Compared to pure representational capacity results, formal claims about learning are much more difficult to make due to the dependence on factors such as the learning algorithm and aspects of the training data. To nevertheless gain valuable insights into the problem, we now focus on the learnability of DPFSAs empirically.

4 Practical Learnability of FSLMs

Our main goal is to provide a principled study of the ability of neural LMs to learn FSLMs. We now describe and justify our experimental setup and then evaluate RNN and Transformer LMs on their ability to learn FSLMs based on it.

4.1 A Critique of Learning Formal Languages

As discussed in §1, plenty of empirical work has investigated the ability of neural language models to learn formal languages such as those described by finite-state automata, i.e., how well a neural language model can be used to assess membership of individual strings in a set. There are multiple workarounds for this discrepancy. Most solutions involve measuring some sort of *accuracy* of next-symbol prediction. For example, Suzgun et al. (2019a,b) and Bhattamishra et al. (2020) evaluate neural LMs on the next-symbol prediction task, which, intuitively, measures whether all allowed continuations of the string under the formal model achieve a large enough probability under the neural LM. Deletang et al. (2023) evaluate the models with the proportion of correctly predicted tokens (where the argmax of the neural LM prediction has to match the ground-truth label). Unfortunately, all these approaches inevitably shoehorn a neural LM into a sort of classifier, mismatching the type of an LM—a probability distribution—and a discrete format language—a set, as illustrated in Fig. 1. Ideally, we would like to measure precisely how the neural LM has learned the *distribution* induced by a formal LM. In this section, we outline and motivate a possible way to approach this challenge.

4.2 Evaluating Probabilistic Learnability

At a high level, we test the learnability of *random* FSLMs by training neural LMs on strings sampled from randomly generated DPFSAs and measuring the distance between the neural LM and the FSLM. Crucially, unlike most existing work, we do not have to rely on classification or next-symbol prediction accuracy-based metrics but rather directly measure the similarity of distributions which presents a much cleaner way of evaluating model similarity. Concretely, given an FSLM p and a neural LM q, we measure the KL divergence between the FSLM and the neural LM:

$$\mathrm{D}_{\mathrm{KL}}\left(p \,||\, q\right) \stackrel{\text{def}}{=} \sum_{\boldsymbol{y} \in \Sigma^*} p\left(\boldsymbol{y}\right) \log \frac{p\left(\boldsymbol{y}\right)}{q\left(\boldsymbol{y}\right)} \tag{6}$$

$$= \mathrm{H}(p, q) - \mathrm{H}(p). \tag{7}$$

The KL divergence is an established and well-understood measure of the distance[4] between two *distributions*. As such, it lends itself naturally to evaluating the difference between LMs; in our case, measuring how well the neural LM has captured the distribution of the FSLM. Such a holistic treatment of the difference between two LMs gives us a tangible and interpretable way of understanding how they differ. To compute, $\mathrm{D}_{\mathrm{KL}}\left(p \,||\, q\right)$, we use Eq. (7). We estimate the first term $\mathrm{H}\left(p, q\right)$ by computing $\widehat{\mathrm{H}}\left(p, q\right)$, the empirical cross-entropy between p and q. The second term can be computed exactly by dynamic programming (Eisner, 2002;

[4]Note, that KL divergence is not a *true* distance, as it is not symmetric and does not fulfill the triangle inequality.

Predictor	Interpretation
$\lvert Q\rvert$	The number states.
$\lvert Q\rvert\lvert\Sigma\rvert$	The number of transitions.
$\lvert\Sigma\rvert$	Alphabet size.
R	The size of the space that $\log p\left(\cdot \mid q\right)$ span for $q \in Q$.
Avg. length	Average length of strings generated by the PFSA.
$\min(\lvert Q\rvert, \lvert\Sigma\rvert)$	Upper bound of R.
$\mathrm{H}(\mathcal{A})$	Entropy of the PFSA.

Table 1: The PFSA-related predictor variables used to estimate KL divergence with their interpretation.

Indep. Var.	$\widehat{\beta}$	SE	p-value
Intercept	7.67	0.08	< 0.001
$\lvert Q\rvert$	4.84	0.19	< 0.001
$\lvert Q\rvert\lvert\Sigma\rvert$	3.48	0.21	< 0.001
$\lvert\Sigma\rvert$	1.34	0.21	< 0.001
R	6.29	0.10	< 0.001
D	0.18	0.08	< 0.05
Avg. len.	-0.36	0.17	< 0.05
$\min(\lvert Q\rvert, \lvert\Sigma\rvert)$	-1.98	0.32	< 0.001
$\widehat{\mathrm{H}}(\mathcal{A})$	4.65	0.22	< 0.001

Table 2: Estimated beta coefficients ($\widehat{\beta}$), standard errors (SE), and p-values for D_{KL} generated with a linear regression model for RNNs.

Zmigrod et al., 2021). However, due to numerical instability, we find using a Monte Carlo estimator more accurate. See App. C.4 for further details on the computation of these evaluation metrics.

4.3 Generating Random DPFSAs

We evaluate neural LMs on their ability to learn *random* FSLMs, which we construct by randomly generating DPFSAs. We vary $|Q| \in \{2, 4, 8, 16, 32\}$ and $|\Sigma| \in \{2, 4, 8, 16, 32\}$. We then randomly select the outgoing neighbors of each of the states (one for each $y \in \Sigma$): For each $q \in Q$ and $y \in \Sigma$ we randomly choose $q' \in Q$ and add the transition $q \xrightarrow{y} q'$ to $\mathcal{A}$. We add weights to the transition function of $\mathcal{A}$ as follows. We generate a random matrix $\mathbf{T} \in \mathbb{R}^{(|\Sigma|+1)\times|Q|}$. For each $\mathrm{R} \in \{2^r \mid 2^r \leqslant \min\left(|Q|, |\Sigma|\right), r \in \{0, \ldots, 5\}\}$, we compute $\mathbf{T}^{\mathrm{R}}$ by reducing the rank of $\mathbf{T}$ to R using SVD. We then set transition probability of $q \xrightarrow{y} q'$ to $\mathrm{w}_{q,y} = \mathrm{softmax}\big(\mathbf{T}^{\mathrm{R}}{}_{:,q}\big)_y$. Finally, we set $\rho\left(q\right) = \mathrm{softmax}\big(\mathbf{T}^{\mathrm{R}}_{:,q}\big)_{\mathrm{EOS}}$. This process results in the generation of up to six random DPFSAs, all sharing the same Q, Σ, and underlying transition function. They differ, however, in the rank of the matrix $\mathbf{T}^{\mathrm{R}}$ that defines the weights of the transitions. Furthermore, the construction of exactly one transition for each q and y ensures that the DPFSA mirrors the nature of a neural LM, which also defines full-support next-symbol probabilities for any prefix of the string. Altogether, this allows us to precisely control the quantities from Tab. 1 and thus the complexity of the DPFSA. At the same time, the DPFSA are determined through the shared parameters of the $\mathbf{T}$, making them easy to connect to neural LMs. See App. C.1 for additional details.

5 Results

5.1 Statistical Evaluation

There are many natural metrics to measure the complexity of DPFSAs. We present the most relevant ones in Tab. 1. Naturally, we expect the difficulty of learning and the required size of the hidden state to increase with all the quantities. We evaluate the size of this effect by fitting a linear model that estimates the learnability (as measured by the KL divergence) from the DPFSA properties shown in Tab. 1 and the neural LM's hidden state size D.

Following the experimental setup outlined in App. C, we obtain D_{KL} results for 6500 RNN and 6500 Transformer LMs trained on strings sampled from random DPFSAs with specific sets of complexity parameters. The linear regression model was fit to the data, separately for the RNN output and for the Transformer output, to quantitatively assess the variation in the empirical D_{KL} divergence. Each of the predictors was standardized using a z-score transformation for an interpretable comparison of the estimated coefficients.

5.2 RNN Findings

As shown in Tab. 2, the linear regression reveals significant effects of each of the included predictors for the RNN output. Of these, the number of states, the number of symbols, the number of transitions, the rank, the PFSA perplexity, and the hidden state size were positive in their direction, indicating an increase in KL divergence with an increase in the predictor of interest. The average string length and minimum of the number of states and symbols were negative in influence, indicating a decrease in KL with an increase in the respective predictor. Overall, the DPFSA rank had the strongest influence on KL divergence, followed in order

Indep. Var.	$\widehat{\beta}$	SE	p-value
Intercept	30.1	0.27	< 0.001
$\lvert Q \rvert$	0.43	0.60	0.47
$\lvert Q \rvert \lvert \Sigma \rvert$	-2.43	0.60	< 0.001
$\lvert \Sigma \rvert$	7.80	0.69	< 0.001
R	12.5	0.33	< 0.001
D	-10.86	0.27	< 0.001
Avg. len.	31.2	0.54	< 0.001
$\min(\lvert Q \rvert, \lvert \Sigma \rvert)$	-1.89	1.05	0.07
$\widehat{\mathrm{H}}(\mathcal{A})$	-14.5	0.71	< 0.001

Table 3: Estimated beta coefficients ($\widehat{\beta}$), standard errors (SE), and p-values for $\mathrm{D_{KL}}$ generated with a linear regression model for Transformers.

$\mathrm{D_{KL}}$	$\lvert Q \rvert$				
	2	4	8	16	32
RNNs	0.44	1.24	5.11	10.41	16.38
Transformers	29.99	33.84	31.00	28.06	29.59

Table 4: KL divergence for RNNs and Transformers as a function of the number of states of the DPFSA.

by number of states and PFSA perplexity, then number of transitions. The remaining predictors were smaller in influence, regardless of direction.

5.3 Transformer Findings

For the Transformer results, the linear regression reveals significant effects of the number of symbols, the number of transitions, the rank, the average string length, the PFSA perplexity, and the hidden state size (see Tab. 3). The number of states and minimum of the number of states and symbols did not reach significance. Of the significant predictors, the number of symbols, rank, and average string length were positive in their influence, indicating an increase in KL divergence as the predictor of interest increased. The number of transitions, PFSA perplexity, and hidden state size were negative in influence, indicating a decrease in KL divergence with an increase in the predictor of interest. Of the positive relationships, average string length had the largest influence, followed in order by rank, then number of symbols. Of the negative relationships, PFSA perplexity had the largest influence, followed by hidden state size, then the number of transitions.

6 Discussion

Comparison of the RNN and Transformer LMs. The linear models revealed an overall disparate pattern of effects between RNNs and Transformers. First to note is the overall performance as revealed by the model intercept (see Tab. 2, Tab. 3, and Fig. 7 in App. D). RNNs tend to outperform Transformers in this task, demonstrating lower average loss. This difference in performance could be attributed to two main factors: 1) As previous research has shown, RNNs are better suited to modeling formal languages (Deletang et al., 2023), and 2) Transformers necessitate careful training involving language-specific hyperparameter tuning, which poses a severe computational challenge. Despite the potential suboptimal training of Transformers, we anticipate that the trend observed here would persist even with optimal training.

There were some similarities in the pattern of the model effects, in that the number of symbols and rank were significant and positive in their influence on KL for each of the RNN and Transformer outputs. Otherwise, the influence of the predictors was fairly different. In particular, several predictors had opposite and significant influences on KL divergence for each of the LM types. Whereas the average string length had a negative influence on KL divergence for the RNN output, it had a positive, and notably, the strongest influence on KL divergence for the Transformer output. In addition, the number of transitions, PFSA perplexity, and hidden state size were positive for the RNN output, but negative for the Transformer KL divergence. The average string length also differed between the two LM types, with a negative influence for RNN output, but positive for Transformer output.

Implications of Thm. 3.1. Thm. 3.1 concretely quantifies the size of the representation space of *any* neural LM required for the correct representation of finite-state LMs. To the best of our knowledge, this is the first result of this generality. Practical implementations of FSLMs might use state spaces and alphabets of sizes ranging from thousands to hundreds of thousands (Mohri and Riley, 1999), which is much larger than the representations used by most modern neural LMs, which tend to be in the order of a few thousand dimensions (Groeneveld et al., 2024). The good performance of much smaller neural LMs on similar datasets indicates that those LMs are indeed low-rank and can thus be approximated well using smaller hidden representations. Nevertheless, Thm. 3.1 provides an interesting limitation on what distributions neural LMs of finite size can represent and points out the limitations of parameter sharing in representing formal models of computation; while neural LMs

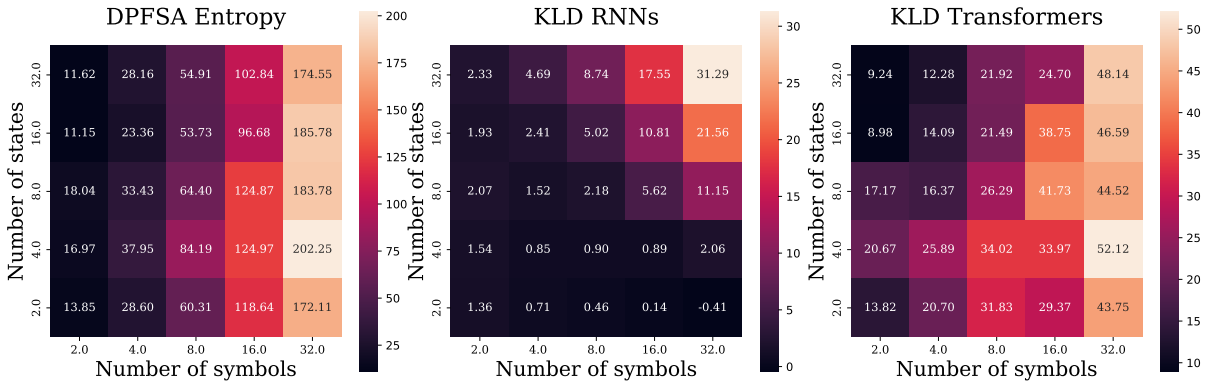

Figure 3: DPFSA's entropy $\widehat{\mathrm{H}}$ and the D_{KL} between the neural LMs and the DPFSAs as a function of $|Q|$ and $|\Sigma|$.

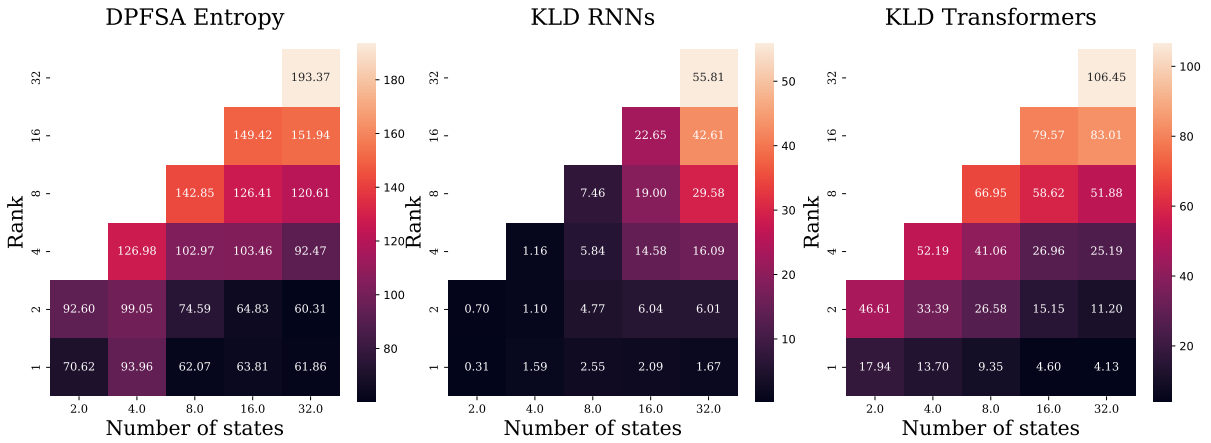

Figure 4: DPFSA's entropy $\widehat{\mathrm{H}}$ and the D_{KL} between the neural LMs and the DPFSAs as a function of $|Q|$ and R.

are good at approximating such models of computation, their inability to represent them *exactly* implies that, with increasing string lengths, their errors will unavoidably accumulate. This leads to poor length generalization often observed in prior work (Weiss et al., 2018; Suzgun et al., 2019b; Bhattamishra et al., 2020; Deletang et al., 2023).

Takeaways from the empirical results. The empirical results in §5.2 complement the theoretical discussion from §3 and the growing field of literature characterizing the representational capacity of neural LMs. In line with the theoretical setting and in contrast to related work, our approach directly evaluates the KL divergence between neural LMs and FSLMs, instead of relying on classification or next-token prediction accuracy measures. Comparing distributions over strings offers a more holistic view of a neural LM's overall ability to emulate FSLMs allowing us to provide compelling insights into what aspects of distributions affect the learnability of formal LMs by controlling for various properties of the FSLMs being learned. Neatly, the observed effects of the rank on the KL divergence align with the theoretical results derived in Thm. 3.1, in that, as the rank of an FSLM grows, a larger hidden state is required in the neural LM to model it appropriately. Surprisingly, in contrast to theoretical results on representations capacity (Indyk, 1995; Svete and Cotterell, 2023), the experiments also show that for RNNs, learnability is *unaffected* by the hidden state size. That is, FSLMs defined by DPFSAs with a large number of states are well-approximated by RNN LMs of size smaller than predicted by theory. The theory developed in this paper and in related work, however, investigates *exact* representation of the FSLMs, not their approximation. The good performance of RNN LMs suggests that RNN LMs manage to learn good approximations of languages that they theoretically can not fully represent. This strongly encourages further research into the *approximation* abilities of neural LMs; judging from our results, those would be more relevant for practical scenarios. Nevertheless, the dependence of the

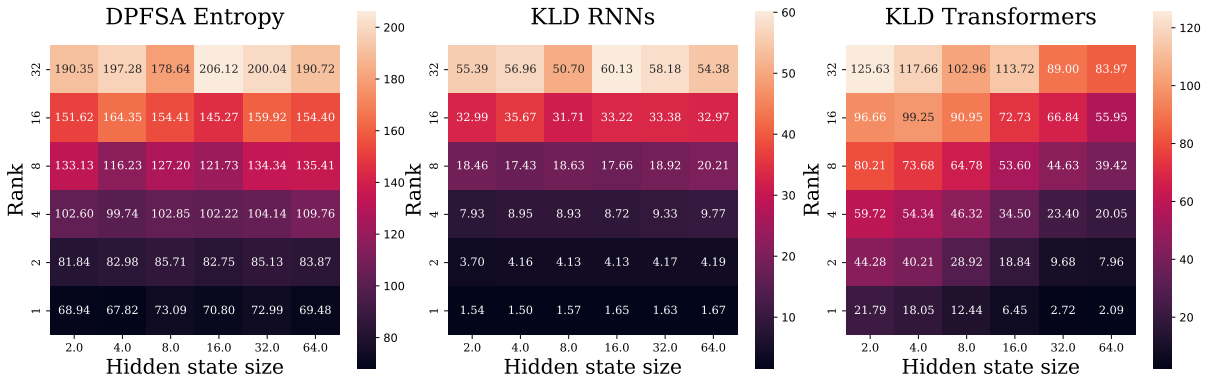

Figure 5: DPFSA's entropy $\widehat{\mathrm{H}}$ and the D_{KL} between the neural LMs and the DPFSAs as a function of D and R.

performance on the rank of the DPFSA seens to demonstrate the utility of formal language theory in providing interpretable insights into the learning abilities of neural LMs.

Extensions. We focus on the learnability of deterministic PFSAs. This makes the theoretical results from §3 particularly interpretable. Extensions to the non-deterministic automata, however, are an interesting next step. Note that in this case, the PFSA rank analysis is slightly more nuanced. A non-deterministic PFSA can, at any point, be in any of the $|Q|$ states (with a probability distribution over them), meaning that the probability of the next symbol is a convex combination of the individual conditional probability distributions (not their logits). This makes the analysis trickier and less interpretable; we leave it for future work to make the current exposition more concise. A further interesting follow-up is also the study of the learnability of (deterministic) *context-free* LMs represented by probabilistic pushdown automata (PPDAs). PPDAs augment PFSAs by implementing a stack that gives the automaton infinitely many configurations. Despite the infinitely many configurations, controlling for their rank analogously to the rank of a PFSA could elucidate how efficiently they are representable by neural LMs.

7 Conclusion

We provide a comprehensive empirical study of the learnability of FSLMs by neural LMs. More concretely, we investigate how well LMs learn to match the distributions over strings generated by FSLMs of varying complexity. For this purpose, we first propose to use KL divergence between such distributions over strings as a more holistic measure of evaluating the similarity of LMs. We establish that for weak equivalence, a neural LM's hidden state size is theoretically lower-bounded by the DPFSA's rank. We find this to be consistent with the results of our controlled experiments on the effects of FSLM properties on learnability. Other theoretical results on the representational capacity of neural LMs (the dependence of the representation size on the number of states and the size of the alphabet), however, seem to be less relevant to the learnability. Overall, our results showcase the utility of using formal language theory to create interpretable insights into the learning abilities of neural LMs but call for theoretical investigations closer to practical applications.

Limitations

We point out some limitations of the presented study. To keep our work concise and results self-contained, we focus only on deterministic FSLMs. Similar and more comprehensive investigations could of course include non-deterministic automata and languages higher up on the Chomsky hierarchy, such as context-free LMs, or even context-sensitive LMs. Our experiments also omit the effect of training dataset size, which might be an interesting quantity to consider when training neural LMs. We leave those considerations to future work.

Moreover, due to computational constraints and the substantial computation load imposed by our experiments, we could not fine-tune our models with language-specific hyperparameters, which are particularly important for transformers. For the same reason, we had to refrain from optimising larger and more capable models. However, we believe that this should not impair the validity of our results, as the trend we observed would hold even with optimal training.

References

Ekin Akyürek, Bailin Wang, Yoon Kim, and Jacob Andreas. 2024. In-context language learning: Architectures and algorithms.

Yonatan Belinkov. 2022. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. 2020. On the Ability and Limitations of Transformers to Recognize Formal Languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116, Online. Association for Computational Linguistics.

Stephen Chung and Hava Siegelmann. 2021. Turing completeness of bounded-precision recurrent neural networks. In *Advances in Neural Information Processing Systems*, volume 34, pages 28431–28441. Curran Associates, Inc.

Gregoire Deletang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. 2023. Neural networks and the Chomsky hierarchy. In *The Eleventh International Conference on Learning Representations*.

A. K. Dewdney. 1977. Threshold matrices and the state assignment problem for neural nets. In *Proceedings of the 8th SouthEastern Conference on Combinatorics, Graph Theory and Computing*, pages 227–245, Baton Rouge, La, USA.

Javid Ebrahimi, Dhruv Gelda, and Wei Zhang. 2020. How can self-attention networks recognize Dyck-n languages? In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4301–4306, Online. Association for Computational Linguistics.

Jason Eisner. 2002. Parameter estimation for probabilistic finite-state transducers. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 1–8, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science*, 14(2):179–211.

Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Raghavi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, Tushar Khot, William Merrill, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Valentina Pyatkin, Abhilasha Ravichander, Dustin Schwenk, Saurabh Shah, Will Smith, Emma Strubell, Nishant Subramani, Mitchell Wortsman, Pradeep Dasigi, Nathan Lambert, Kyle Richardson, Luke Zettlemoyer, Jesse Dodge, Kyle Lo, Luca Soldaini, Noah A. Smith, and Hannaneh Hajishirzi. 2024. Olmo: Accelerating the science of language models.

Michael Hahn. 2020. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171.

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

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.

John Hewitt and Christopher D. Manning. 2019. A structural probe for finding syntax in word representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4129–4138, Minneapolis, Minnesota. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

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.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3651–3657, Florence, Italy. Association for Computational Linguistics.

Jaap Jumelet and Willem Zuidema. 2023. Transparency at the source: Evaluating and interpreting language models with access to the true distribution. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 4354–4369, Singapore. Association for Computational Linguistics.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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. *CoRR*, abs/1906.06349.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2873–2882. PMLR.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535.

Bingbin Liu, Jordan T Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. 2023. Transformers learn shortcuts to automata.

Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith. 2019. Linguistic knowledge and transferability of contextual representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1073–1094, Minneapolis, Minnesota. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2018. Fixing weight decay regularization in adam.

Christopher D. Manning, Kevin Clark, John Hewitt, Urvashi Khandelwal, and Omer Levy. 2020. Emergent linguistic structure in artificial neural networks trained by self-supervision. *Proceedings of the National Academy of Sciences*, 117(48):30046–30054.

Warren S. McCulloch and Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

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. 2023. Formal languages and the NLP black box. In *Developments in Language Theory*, pages 1–8, Cham. Springer Nature Switzerland.

William Merrill and Ashish Sabharwal. 2023. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545.

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 and Michael Riley. 1999. Integrated context-dependent networks in very large vocabulary speech recognition. pages 811–814. Publisher Copyright: © 1999 6th European Conference on Speech Communication and Technology, EUROSPEECH 1999. All rights reserved.; 6th European Conference on Speech Communication and Technology, EUROSPEECH 1999 ; Conference date: 05-09-1999 Through 09-09-1999.

Richard Montague. 1970. Universal grammar. *Theoria*.

Franz Nowak, Anej Svete, Li Du, and Ryan Cotterell. 2023. On the representational capacity of recurrent neural language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7011–7034, Singapore. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2021. A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8:842–866.

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.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. 2023. Transformers as recognizers of formal languages: A survey on expressivity. *arXiv preprint arXiv:2311.00208*.

Mirac Suzgun, Yonatan Belinkov, Stuart Shieber, and Sebastian Gehrmann. 2019a. LSTM networks can perform dynamic counting. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 44–54, Florence. Association for Computational Linguistics.

Mirac Suzgun, Yonatan Belinkov, and Stuart M. Shieber. 2019b. On evaluating the generalization of LSTM models in formal languages. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2019*, pages 277–286.

Anej Svete and Ryan Cotterell. 2023. Recurrent neural language models as probabilistic finite-state automata. *arXiv preprint arXiv:2310.05161*.

Josef Valvoda, Naomi Saphra, Jonathan Rawski, Adina Williams, and Ryan Cotterell. 2022. Benchmarking compositionality with formal languages. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 6007–6018, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Shunjie Wang and Shane Steinert-Threlkeld. 2023. Evaluating transformer's ability to learn mildly context-sensitive languages. In *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 271–283, 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.

Jennifer C. White and Ryan Cotterell. 2021. Examining the inductive bias of neural language models with artificial languages. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 454–463, Online. Association for Computational Linguistics.

Ran Zmigrod, Tim Vieira, and Ryan Cotterell. 2021. Efficient computation of expectations under spanning tree distributions. *Transactions of the Association for Computational Linguistics*, 9:675–690.

A Additional Related Work

A.1 Representational Capacity of Neural LMs

Plenty of theoretical work has investigated the representational capacity of various neural LM architectures (Merrill, 2023; Strobl et al., 2023). Finite-state languages (and, to a lesser extent, finite-state LMs) have been linked to neural LMs particularly often, especially to recurrent neural LMs, but similar connections have also been made for Transformers (Merrill, 2019; Merrill et al., 2020; Liu et al., 2023). Distinctively interesting are the bounds on the space requirements for emulating FSAs (Dewdney, 1977; Indyk, 1995; Hewitt et al., 2020; Svete and Cotterell, 2023). This work bridges the theoretical work with practice, tests its applicability, and uses its insights for an informed study of the practical representational capacity of neural LMs.

A.2 Learning Formal Languages

Work similar to ours in spirit is that of Jumelet and Zuidema (2023), where the authors train and evaluate neural LMs with probabilistic context-free grammars. They use the underlying data-generating distribution (the probabilistic grammar) to evaluate how well the model has learned the distribution. Moreover, the knowledge of grammar allows them to probe the model for the encodings of individual constituents, similar to how we probe for the states of the automaton. In contrast to our work, however, Jumelet and Zuidema (2023) focus on learning human-language-based grammars, which do not provide a holistic picture of the representability of general formal LMs by neural LMs.

Deletang et al. (2023) provide a comprehensive survey of the learnability of diverse formal languages. Unlike us, they focus on learning discrete languages, particularly from the perspective of learning *algorithms* and investigating LMs' inductive biases. They formulate this as a *transduction*—a string-to-string mapping. They arrive at interesting results showing that popular neural LMs are hard to place on the standard Chomsky hierarchy of languages. This can partly be explained by the mismatch of the training task—transduction—and the probabilistic nature of a neural LM, since the probabilistic Chomsky hierarchy is known to differ from the discrete one (Icard, 2020). In contrast to our work, Deletang et al. (2023) also only consider a limited set of hand-picked languages which, while providing algorithmic insights into how LMs work, do not extensively probe the learnability of the language classes.

Testing the compositional generalization of NNs, Valvoda et al. (2022) sample an infinite number of finite languages. Thereby they can draw conclusions about the learnability of an entire class of languages—sub-regular ones encoded by subsequential finite state transducers. Their work connects Montague's theory of compositional generalization (Montague, 1970) with the popular SCAN benchmark of compositional behavior (Lake and Baroni, 2018). Unlike our work, they investigate deterministic transducers and seq2seq models.

Another similar work is that of White and Cotterell (2021), who use artificial languages to identify the biases of neural LMs. By modifying a base grammar, they experiment with the learnability of 64 languages. Unlike us, their work focuses solely on topological aspects of the language, which limits their findings to observations over the word order.

In a different line of work, Akyürek et al. (2024) evaluate neural LMs' abilities to learn finite-state languages *in context*. Rather than learning one particular distribution from the training dataset, they train neural LMs to model the language of any finite-state automaton given a number of samples from it—that is, to infer the generating mechanism from the context. They consider only discrete languages (even though their generative setup is probabilistic) and due to the in-context learning setting, they do not analyze the dynamics of the neural LM implementing individual languages.

B Probabilistic Finite-state Automata

We begin by more formally defining the notion of probabilistic finite-state automata (PFSAs), which were only informally introduced in §2.

Definition B.1. *A **probabilistic finite-state automaton** (PFSA) is a 5-tuple* $(\Sigma, Q, \delta, \lambda, \rho)$ *where* Σ *is an alphabet,* Q *a finite set of states,* $\delta \subseteq Q \times \Sigma \times [0, 1] \times Q$ *a finite set of weighted transitions and*

$\lambda, \rho\colon Q \to [0,1]$ *the initial and final weighting functions. Moreover,* δ, λ *and* ρ *are required to satisfy that* $\sum_{q \in Q} \lambda(q) = 1$*, and, for all* $q \in Q$*,* $\sum_{(q,y,w,q') \in \delta} w + \rho(q) = 1$.

We denote $(q, y, w, q') \in \delta$ with $q \xrightarrow{y/w} q'$.

Definition B.2. *A* ***path*** $\boldsymbol{\pi}$ *in a PFSA* $\mathcal{A}$ *is a sequence of consecutive transitions* $q_1 \xrightarrow{y_1/w_1} q_2, \cdots, q_N \xrightarrow{y_N/w_N} q_{N+1}$*. Its* ***length*** $|\boldsymbol{\pi}|$ *is the number of transitions in it and its* ***scan*** $\boldsymbol{s}(\boldsymbol{\pi})$ *the concatenation of the symbols on them. We denote with* $\Pi(\mathcal{A})$ *the set of all paths in* $\mathcal{A}$ *and with* $\Pi(\mathcal{A}, \boldsymbol{y})$ *the set of all paths that scan* $\boldsymbol{y} \in \Sigma^*$.

The weights of the transitions along a path are multiplicatively combined to form the weight of the path. The weights of all the paths scanning the same string are combined additively to form the weight of that string.

Definition B.3. *The* ***path weight*** *of* $\boldsymbol{\pi} \in \Pi(\mathcal{A})$ *is* $\mathrm{w}(\boldsymbol{\pi}) = \lambda(q_1) \left[\prod_{n=1}^{N} w_n\right] \rho(q_{N+1})$*. The* ***stringsum*** *of* $\boldsymbol{y} \in \Sigma^*$ *is* $\mathcal{A}(\boldsymbol{y}) \stackrel{\text{def}}{=} \sum_{\boldsymbol{\pi} \in \Pi(\mathcal{A}, \boldsymbol{y})} \mathrm{w}(\boldsymbol{\pi})$.

It is easy to see that the final weights $\rho(q)$ play an analogous role to the EOS symbol in the context of autoregressive LMs—they both correspond to the probabilities of ending the generation of the string.

Definition B.4. *A PFSA* $\mathcal{A} = (\Sigma, Q, \delta, \lambda, \rho)$ *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* $q \xrightarrow{y/w} q' \in \delta$ *with* $w > 0$.

In general, there can be infinitely many PFSAs that define a given FSLM. However, in the deterministic case, there is a unique minimal DPFSA.

Definition B.5. *A DPFSA* $\mathcal{A} = (\Sigma, Q, \delta, \lambda, \rho)$ *is* ***minimal*** *for the FSLM* p *if there is no weakly equivalent DPFSA* $\mathcal{A}' = (\Sigma, Q', \lambda', \rho', \delta')$ *with* $|Q'| < |Q|$.

C Experimental Details

C.1 Sampling DPFSAs of varying complexity

The DPFSAs we used in our experiments were sampled with $|Q| \in \{2, 4, 8, 16, 32\}$ over alphabets alphabets of sizes $|\Sigma| \in \{2, 4, 8, 16, 32\}$. Given a sampled DPFSA $\mathcal{A}$ with $|Q|$ states over an alphabet Σ, we randomly set its unweighted transition function. That is, for each $q \in Q$ and $y \in \Sigma$ we randomly choose $q' \in Q$ and add the transition $q \xrightarrow{y} q'$ to $\mathcal{A}$.

We add weights to the transition function of $\mathcal{A}$ as follows. We generate a random matrix $\mathbf{T} \in \mathbb{R}^{(|\Sigma|+1) \times |Q|} \sim \mathcal{N}(\mu = 0, \sigma^2 = 4)$, and define $\mathrm{R}_{\max} = \mathrm{rank}(\mathbf{T})$ (Note that $\mathrm{R}_{\max} \leqslant \min(|Q|, |\Sigma|)$). For each $\mathrm{R} \in \{2^r | 2^r \leqslant \mathrm{R}_{\max}, r \in \{0, 1, 2, 3, 4, 5\}\}$, we compute $\mathbf{T}^{\mathrm{R}}$ by reducing the rank of $\mathbf{T}$ to R using SVD. Next, we add weights to the transition function of $\mathcal{A}$ by replacing each unweighted transition $q \xrightarrow{y} q'$ with $q \xrightarrow{y/\mathrm{w}_{q,y}} q'$, where $\mathrm{w}_{q,y} = \mathrm{softmax}\big(\mathbf{T}^{\mathrm{R}}{}_{:,q}\big)_y$. Finally, we set $\rho(q) = \mathrm{softmax}\big(\mathbf{T}^{\mathrm{R}}_{:,q}\big)_{\mathrm{EOS}}$. This process results with the generation of up to six[5] random DPFSAs, all sharing the same Q, Σ and underlying transition function. They differ, however, in the rank of the matrix $\mathbf{T}^{\mathrm{R}}$ that defines the weights of the transitions.

C.2 Generating the Data

For a given DPFSA $\mathcal{A}$, we sample 20k random strings, terminating the generation process of each string when EOS is sampled. We divide the dataset into train and test splits, such that no string is shared between the sets, and the test set has at least 2k strings. We truncate the strings to 128 symbols to accommodate the limited context length of the Transformer model we used. Fig. 6 shows a histogram of the average length of strings generated for each DPFSA.

[5] $|\{2^r | 2^r \leqslant \mathrm{R}_{\max}, r \in \{0, 1, 2, 3, 4, 5\}\}| \leqslant 6$

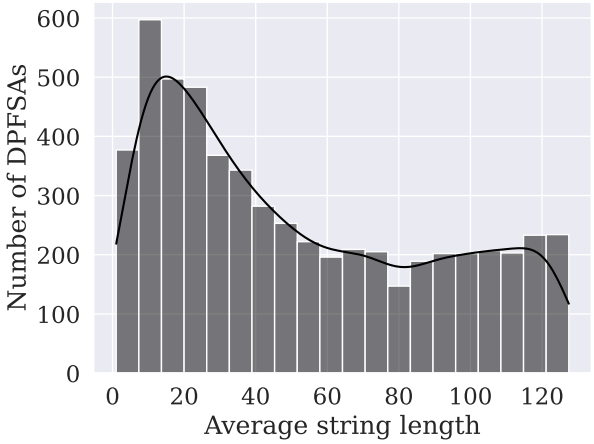

Figure 6: The statistics of the training dataset.

C.3 Training the Neural LMs

We train neural LMs on our dataset using the following procedure, repeated 6500 times:

1. Sample a random DPFSA $\mathcal{A}$ with $|Q|$, $|\Sigma|$ and rank R using the process described in App. C.1.
2. Sample 20k strings from $\mathcal{A}$ and split them to train set and test set using the process described in App. C.2.
3. Train an RNN with a random hidden state size D sampled from $\{2, 4, 8, 16, 32, 64\}$ on the train set strings.
4. Train a Transformer model with the same hidden state size D on the train set strings.
5. Compute the D_{KL} between $\mathcal{A}$ and each of the two trained neural LMs on the test set strings.

We train the RNN and Transformer models using the following hyperparameters:

- **RNNs**: We use a unidirectional LSTM with two hidden layers, each with 64-dimensional hidden states and an embedding size of 64. We trained each model for two epochs using a batch size of 32 and a learning rate of 0.001, an Adam optimizer with default settings, and a standard cross-entropy loss (Kingma and Ba, 2014).
- **Transformers**: We use the GPT-2 model architecture (Radford et al., 2019) with six attention layers, each with four attention heads and 256-dimensional representations. We use an embedding size of 64 and an input context length of 128. We trained each model for three epochs using a batch size of 32, an AdamW (Loshchilov and Hutter, 2018) optimizer with default settings, and a standard cross-entropy loss.

C.4 Evaluation

$\widehat{\mathrm{H}}\left(p(\mathcal{D}_{\text{test}})\right)$ is calculated by aggregating all the weights along the paths of each string in $\mathcal{D}_{\text{test}} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\}$. That is,

$$\widehat{\mathrm{H}}\left(p(\mathcal{D}_{\text{test}})\right) = \frac{1}{N}\sum_{i=1}^{N} -\log\left(p(\boldsymbol{\pi}_i)\right) = \frac{1}{N}\sum_{i=1}^{N} -\log\left(w(\boldsymbol{\pi}_i)\right) \tag{8}$$

where $\boldsymbol{\pi}_i$ is the unique path in $\mathcal{A}$ accepting $\boldsymbol{y}_i$.[6]

[6] $\boldsymbol{\pi}_i$ is unique as $\mathcal{A}$ is deterministic.

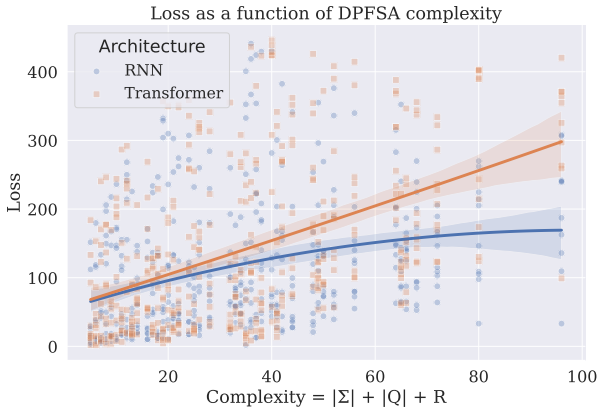

Figure 7: Validation performance of RNNs and Transformers as a function of the PFSA's complexity, computed as $|\Sigma| + |Q| + \mathrm{R}$.

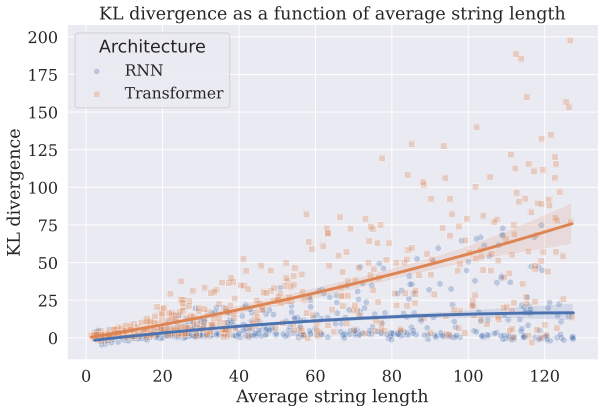

Figure 8: D_{KL} of RNNs and Transformers as a function of the average string length of the DPFSA.

Similarly, we calculate

$$\widehat{\mathrm{H}}\left(p(\mathcal{D}_{\text{test}}), q(\mathcal{D}_{\text{test}})\right) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{|\boldsymbol{y}_i|}\mathrm{H}\left(p\left(y_t \mid \boldsymbol{y}_{<t}\right), q\left(y_t \mid \boldsymbol{y}_{<t}\right)\right) \tag{9}$$

$$= -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{|\boldsymbol{y}_i|} p\left(y_t \mid \boldsymbol{y}_{<t}\right) \log q\left(y_t \mid \boldsymbol{y}_{<t}\right) \tag{10}$$

where $p\left(y_t \mid \boldsymbol{y}_{<t}\right) = \mathrm{w}_t$ is the weight of the transition $q_{t-1} \xrightarrow{y_t/w_t} q_t$ in $\boldsymbol{\pi}_i$, and $q\left(y_t \mid \boldsymbol{y}_{<t}\right)$ is $\mathrm{softmax}(\mathbf{E}\mathbf{h}_{t-1})_{y_t}$ given by the neural LM.

D Additional Results

This section includes figures presenting the results of additional experiments augmenting and supporting the claims in the main paper:

Fig. 7 Overall performance of the neural LM models as a function of the "total complexity" of the PFSA they were optimised for, which we define as the sum of $|\Sigma|$, $|Q|$, and R. Performance is measured as the cross-entropy of the neural model on a held-out test set. We compute the loss by summing it over symbols and dividing this sum by the number of sequences in the test set. We can see that RNNs tend to overperform Transformers, especially for more complex PFSAs.

Fig. 8 The D_{KL} of RNNs and Transformers as a function of the average string length of the DPFSA. Similarly to Tab. 3, we see that Transformers are much more sensitive to string length compared to RNNs.