Can Transformers Learn n-gram Language Models?

Anej Svete🚡   **Nadav Borenstein**🚠

Mike Zhou🔔   **Isabelle Augenstein**🚠   **Ryan Cotterell**🚡

🚡ETH Zürich   🚠University of Copenhagen   🔔University of Pennsylvania

{nb, augenstein}@di.ku.dk   mikezhou@seas.upenn.edu

{asvete, ryan.cotterell}@inf.ethz.ch

Abstract

Much theoretical work has described the ability of transformers to represent formal languages. However, linking theoretical results to empirical performance is not straightforward due to the complex interplay between the architecture, the learning algorithm, and training data. To test whether theoretical lower bounds imply *learnability* of formal languages, we turn to recent work relating transformers to n-gram language models (LMs). We study transformers' ability to learn random n-gram LMs of two kinds: ones with arbitrary next-symbol probabilities and ones where those are defined with shared parameters. We find that classic estimation techniques for n-gram LMs such as add-λ smoothing outperform transformers on the former, while transformers perform better on the latter, outperforming methods specifically designed to learn n-gram LMs.

⭗ github.com/rycolab/learning-ngrams

1 Introduction

A large body of work has investigated the ability of transformers (Vaswani et al., 2017) to represent formal languages (Strobl et al., 2023). Such results tell us what languages transformers can *represent*, but not how well they can *learn* them from data. Existing work has tested the learning abilities of transformers as *classifiers* mapping strings to language membership decisions (e.g., Bhattamishra et al., 2020; Delétang et al., 2023). However, language models (LMs) are not classifiers of strings but rather *distributions* over them. In this light, a recent line of work has argued for a more direct evaluation of LMs as distributions (Svete and Cotterell, 2023; Nowak et al., 2023; Svete et al., 2024a,b; Borenstein et al., 2024; Nowak et al., 2024).

In the pursuit to understand transformers as probability distributions, **n-gram LMs** have recently emerged as a useful playground for analyzing transformers' interpretability (Liu et al., 2024; Voita et al., 2024), learning behavior (Edelman et al., 2024; Chen et al., 2024), in-context learning (Olsson et al., 2022; Arora et al., 2024; Akyürek et al., 2024; Nguyen, 2024), and representational capacity (Svete and Cotterell, 2024). Svete and Cotterell (2024), in particular, show that transformers can encode any n-gram LM, lower-bounding their representational capacity and suggesting that encoding n-gram statistics is indeed a feasible mechanism for transformers to process language. However, being able to *represent* n-gram statistics is insufficient to rely on them in practice. For that, *learning* them is necessary. This motivates the question: Given that transformers can represent n-gram LMs, how good are they at learning them? Answering this brings us closer to a practical understanding of transformers, showing whether n-gram statistics are practically relevant for their behavior and illuminating what theoretical results on neural architectures imply about their ability to learn formal languages.

We study how well transformers can learn n-gram LMs. On the one hand, we find that they generalize well on n-gram LMs whose next-symbol probabilities depend on a linear combination of features of the n-gram symbols (representation-based n-gram LMs), outperforming baselines such as count-based techniques and models with hand-crafted features. On the other, we find that transformers perform worse than count-based techniques on n-gram LMs with arbitrary next-symbol probabilities. This illuminates transformers' inductive biases: It shows that they struggle to approximate even simple probability distributions whose next-symbol probabilities are arbitrary while being good at learning representation-based LMs.

2 Preliminaries

We begin by introducing some preliminaries. An **alphabet** Σ is a finite, non-empty set of **symbols**. The **Kleene closure** Σ^* of the alphabet Σ is the set of all strings of the symbols in Σ. The **length** of the string $\boldsymbol{y} = y_1 \ldots y_T \in \Sigma^*$, denoted by $|\boldsymbol{y}| = T$, is the number of symbols it contains. We will

9851

use $\boldsymbol{y}_i^j \stackrel{\text{def}}{=} y_i \cdots y_j$ to denote substrings between positions i and j inclusive.

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 $\boldsymbol{y} \in \Sigma^*$. Many LMs define $p(\boldsymbol{y})$ autoregressively:

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

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

When defined autoregressively, next-symbol distributions $p(y_t \mid \boldsymbol{y}_{<t})$ in an n-gram LM satisfy the n-gram assumption, stated formally below.

Assumption 2.1. *The **n-gram assumption** states that the conditional probability of the symbol y_t given $\boldsymbol{y}_{<t}$ only depends on $n-1$ previous symbols:*

$$p_n(y_t \mid \boldsymbol{y}_{<t}) = p_n(y_t \mid \boldsymbol{y}_{t-n+1}^{t-1}). \quad (2)$$

*We will refer to n as the **order** of the n-gram LM and $\boldsymbol{y}_{t-n+1}^{t-1}$ as the **history** of y_t.*

We will denote a length-$(n-1)$ history as $\boldsymbol{y}^n$ whenever its position t in the string is unimportant. Additionally, we assume that the histories are padded with the beginning-of-string symbol $\text{BOS} \notin \Sigma$ and denote $\underline{\Sigma} \stackrel{\text{def}}{=} \Sigma \cup \{\text{BOS}\}$ for convenience.

Representation-based n-gram LMs. The next-symbol probabilities $p_n(y \mid \boldsymbol{y}^n)$ of n-gram LMs can be arbitrary. This requires storing $\mathcal{O}(|\Sigma|^{n-1})$ parameters and does not capture the intuitive notion that similar n-grams define similar next-symbol probabilities. This can be addressed by **representation-based** n-gram LMs, which define $p_n(y \mid \boldsymbol{y}^n)$ as a function of a **representation** of $\boldsymbol{y}^n$.

Definition 2.1. *An n-gram LM p_n is **representation-based** if there exist a **representation function** $\boldsymbol{h} \colon \underline{\Sigma}^{n-1} \to \mathbb{R}^D$ and an **output matrix** $\mathbf{E} \in |\overline{\Sigma}| \times D$ for some $D \in \mathbb{N}$ such that*

$$p_n(\overline{y} \mid \boldsymbol{y}^n) = \text{softmax}(\mathbf{E}\, \boldsymbol{h}(\boldsymbol{y}^n))_{\overline{y}} \quad (3)$$

for all $\overline{y} \in \overline{\Sigma}$ and $\boldsymbol{y}^n \in \underline{\Sigma}^{n-1}$ where

$$\text{softmax}(\mathbf{x})_{\overline{y}} \stackrel{\text{def}}{=} \frac{\exp(x_{\overline{y}})}{\sum_{\overline{y}' \in \overline{\Sigma}} \exp(x_{\overline{y}'})}. \quad (4)$$

A transformer LM computes the $p(y_t \mid \boldsymbol{y}_{<t})$ using the self-attention mechanism, which builds the representation by attending to different symbols in the preceding string (Vaswani et al., 2017; Radford et al., 2019).[1] This can intuitively be connected to the way n-gram LMs compute the next-symbol probabilities by attending to the last $n-1$ symbols.

Transformers can represent n-gram LMs. Svete and Cotterell (2024, Thms. 3.1 and 3.2) show that, for any n-gram LM, there exists a weakly equivalent transformer LM. They describe intuitive mechanisms in which $n-1$ heads (Thm. 3.1) or $n-1$ layers (Thm. 3.2) attend to the previous $n-1$ symbols using *sparse* attention that computes attention weights with the sparsemax function (Martins and Astudillo, 2016). The use of sparse attention is particularly noteworthy since it departs from the standard soft-attention transformers usually used in practice (Vaswani et al., 2017). While the theory does not consider soft-attention transformers concretely, it does suggest that being able to assign no attention to irrelevant tokens, which is impossible with soft attention, could be beneficial.

3 Learnability of n-gram LMs

At a high level, we study how well transformers can learn n-gram LMs. We do that by training transformers on strings sampled from randomly generated n-gram LMs and measuring a notion of distance between the trained and the ground-truth LM.

3.1 A Framework for Evaluating LMs

Given an n-gram LM p_n and a transformer LM $p_{\mathcal{T}}$, their **KL divergence** is defined as

$$D_{\text{KL}}(p_n \mid\mid p_{\mathcal{T}}) \stackrel{\text{def}}{=} \sum_{\boldsymbol{y} \in \Sigma^*} p_n(\boldsymbol{y}) \log \frac{p_n(\boldsymbol{y})}{p_{\mathcal{T}}(\boldsymbol{y})}. \quad (5)$$

The KL divergence is an established measure of similarity between probability distributions.[2] As such, it lends itself naturally to measuring the difference between LMs; in our case, measuring how well a neural LM matches an n-gram LM.[3] This sidesteps the reliance on proxy measures of fit such as the next-symbol prediction accuracy, which is often used for evaluating the learnability of formal languages (Borenstein et al., 2024). We evaluate model performance by approximating D_{KL} on held-out sets; see App. B.3 for details.

---

[1] For space reasons, we do not give a detailed description of the transformer architecture. See App. B.2.1 for the details of the models used in the experiments.

[2] KL divergence is not a distance, as it is not symmetric and does not fulfill the triangle inequality.

[3] D_{KL} is also the quantity minimized when training LMs under the MLE objective (Cotterell et al., 2024).

3.2 Experimental Setup

We study the learnability of n-gram LMs by *(1)* sampling n-gram LMs of three types described in §3.2.1, *(2)* generating train[4] and test datasets, *(3)* training baselines and transformers, and *(4)* computing the KL divergence between the trained LMs and the ground-truth LMs.

Axes of comparison. We investigate the learnability of n-gram LMs along two axes:

Axis 1: Whether the n-gram LM is representation-based or not and the degree to which its parameters are shared, and

Axis 2: Measures of complexity of the n-gram LM:
  - the order n of the n-gram LM,
  - the size $|\Sigma|$ of the alphabet, and
  - the rank R of the output matrix $\mathbf{E}$.

The first axis is motivated by the parameter-sharing nature of transformers while the second is motivated by the wish to understand the effect of the complexity of the n-gram LM on its learnability.

3.2.1 Data Generation

We generate training and test datasets by sampling strings from randomly generated n-gram LMs of three types: *(1)* non-representation-based, general, n-gram LMs whose conditional next-symbol probability distributions are arbitrary, *(2)* sparse representation-based n-gram LMs, and *(3)* dense representation-based n-gram LMs. The difference between sparse and dense representation-based n-gram LMs lies in the degree to which the parameters of the n-gram LM are shared—in the former, individual symbols of the alphabet define independent parameters (meaning that the size of the representations grows with $|\Sigma|$), whereas, in the latter, the parameters are shared across symbols (meaning that the size of the representations is independent of $|\Sigma|$). See App. B.1 for more details.

We control the complexity of the n-gram LM through the following parameters: n, $|\Sigma|$, and, for representation-based LMs, the rank R of the output matrix $\mathbf{E}$.[5] Concretely, we generate representation-based datasets with $n \in \{4, 8, 12\}$ and $|\Sigma| \in \{64, 128, 256\}$. For dense representation-based n-gram LMs, we vary $\mathbf{E}$'s rank $R \in \{2, 8, 16\}$. Due to the space complexity of storing $|\Sigma|^{n-1}$ different

	Parameter	Description
p_n	$n \in \mathbb{N}$	Order of p_n
p_n	$|\Sigma| \in \mathbb{N}$	Size of Σ
p_n	$\mathrm{H}(p_n) \in \mathbb{R}$	Entropy of p
p_n	$R \in \mathbb{N}$	Rank of $\mathbf{E}$
p_n	Dense $\in \{0, 1\}$	Dense representations
$p_{\mathcal{T}}$	$L \in \mathbb{N}$	Number of layers
$p_{\mathcal{T}}$	$H \in \mathbb{N}$	Number of heads
$p_{\mathcal{T}}$	sparsemax $\in \{0, 1\}$	Sparse attention

Table 1: Predictors of $D_{\mathrm{KL}}(p_n \,||\, p_{\mathcal{T}})$.

conditional distributions, we generate general n-gram LMs with $n \in \{2, 4, 6\}$ and $|\Sigma| = \{8, 12, 16\}$ and additional dense representation-based n-gram LMs of the same complexity for comparison. We generate five random n-gram LMs for each of the configurations and sample disjoint training and test datasets of size $N_{\mathrm{Train}} = 50k$ and $N_{\mathrm{Test}} = 30k$, respectively. See App. B.1 for more details.

3.2.2 Models

Transformer models. We use transformers (TF) of different sizes. Inspired by the theoretical constructions considered, we are particularly interested in how the number of attention heads and layers affects the learnability of n-gram LMs. We investigate this by varying the number of heads and layers. Moreover, since the theoretical constructions rely on *sparse* attention mechanisms, we also investigate the effect of using the sparsemax (Martins and Astudillo, 2016) for computing attention weights. More details are given in App. B.2.

Classic techniques. Count-based estimators such as the maximum-likelihood n-gram estimate have been used in NLP for decades. They compute the next-symbol probabilities by counting the occurrences of (lower-order) n-grams in the training data. They are thus well-suited for learning n-gram LMs, making them a difficult-to-beat baseline. *Smoothing* estimates the probabilities of unseen n-grams by redistributing the probability mass of seen n-grams, regularizing count-based estimators (Katz, 1987; Witten and Bell, 1991; Ney et al., 1994; Gale and Sampson, 1995; Malagutti et al., 2024).[6] We

---

[4] The development dataset which we use to select the hyperparameters is taken from the training dataset.

[5] The rank R has been identified as a significant predictor of the learnability of general LMs (Borenstein et al., 2024).

[6] We expect that the fact that the ground-truth n-gram LMs are of a high order and representation-based makes the task for traditional smoothing techniques more difficult and may require a large degree of smoothing.

consider add-λ, absolute discounting (Ney et al., 1994), and Witten–Bell (Witten and Bell, 1991) smoothing along with the standard maximum likelihood solution. These techniques are described in App. A; see also App. B.2 for details.

Two additional baselines. We study two additional baselines: a **log-linear model** (LL) with fixed, sparse representations of the histories and a **neural n-gram LM** (Neural) that learns dense representations of the histories. The log-linear model represents the history $\boldsymbol{y}^n$ as a concatenation of the one-hot encodings of its symbols. It learns the appropriate output matrix $\widehat{\mathbf{E}} \in \mathbb{R}^{|\overline{\Sigma}| \times (n-1)|\underline{\Sigma}|}$ such that $p_n(\overline{y} \mid \boldsymbol{y}^n) = \mathrm{softmax}(\widehat{\mathbf{E}}\, \boldsymbol{h}\,(\boldsymbol{y}^n))_{\overline{y}}$ approximates the training data well. The neural n-gram LM is based on previous work exploring neural n-gram LMs (Bengio et al., 2000, 2003, 2006; Sun and Iyyer, 2021). It learns static symbol representations $r\,(y) \in \mathbb{R}^D$ for $y \in \Sigma$ and concatenates $r\,(y_i)$ for $\boldsymbol{y}^n = y_1 \cdots y_{n-1}$ before non-linearly transforming the concatenated representations with an MLP to compute $\boldsymbol{h}\,(\boldsymbol{y}^n)$. See also App. A.

3.2.3 Statistical Analysis

We determine the importance of n-gram LM parameters and transformer components on the learnability of n-gram LMs by evaluating how predictive they are of the transformer performance. Concretely, we investigate the predictors listed in Tab. 1 and fit a linear regression model to predict the test KL divergence. See App. B.4 for more details.

4 Experimental Results and Discussion

This section presents the experimental results. All tables contain the mean and standard deviation of the estimate of the KL divergence $\widehat{D_{\mathrm{KL}}}$ between the ground truth and trained LMs estimated on the test dataset.[7] Negative values indicate an approximation error of the (low) D_{KL}.

4.1 The Effect of Parameter Sharing

Tab. 2 compares the performance of transformers and baselines on representation- and non-representation-based n-gram LMs with $n = 6$. As expected, simple counting-based methods that do not assume representation-based ground-truth LMs perform equally well regardless of whether or not the n-gram LM is representation-based. Models that assume representation-based n-gram LMs (log-linear, neural n-gram, and transformer LMs)

---

perform better on representation-based ones. Transformers outperform log-linear models, possibly due to their better fit for dense representation-based models, and the simple neural n-gram LMs perform best. App. C.1 further analyzes the effect of the degree to which the parameters of the n-gram LM are shared, showing that sparse-representation-based n-gram LMs are more difficult to approximate with the neural n-gram model and transformers than dense-representation-based ones.

Our results provide a complementary view of the theory connecting transformers to n-gram LMs. They show the importance of *parameter sharing* in the ground-truth model for transformers to match the distribution well—in the case of no parameter sharing, simple count-based methods perform better. The good performance of transformers is particularly interesting compared to the simple and more structured log-linear model.

4.2 The effect of n-gram LM Complexity

We next investigate the effect of the n-gram LM parameters described in §3.2. Tab. 3 show the performance of the best model configurations for different combinations of n and $|\Sigma|$ on n-gram LMs with dense representations and of rank 16. As expected, the performance of all tested methods decays with increasing n-gram complexity. While the structured neural n-gram models perform best, transformers do not fare much worse, outperforming classical methods on all but the smallest models and the log-linear model in all settings. Rank is further investigated in App. C.2, where we show that higher-rank models are more difficult to learn for neural n-gram and transformer LMs.

4.3 Predictors of Learnability

Tab. 4 shows the linear model coefficients predicting the transformer performance. According to the absolute values of the coefficients, the most significant predictor is whether the representations are dense or sparse, followed by the n-gram LM order and the alphabet size. The entropy is *negatively* correlated with the KL divergence, indicating that higher-entropy n-gram LMs are easier to learn, in line with existing work (Borenstein et al., 2024). As suggested by theoretical work, we find the number of heads and layers to be significant positive predictors of the learnability of the n-gram LM.

Soft vs. sparse attention transformers. The distinction between soft- and sparse-attention trans-

---

[7]See App. B.3 for details on the approximation.

$|\Sigma|$	8		12		16	
Parameter sharing	No	Yes	No	Yes	No	Yes
Classic	**3.04** (±1.46)	2.36 (±0.23)	**17.34** (±1.01)	2.45 (±0.96)	**50.35** (±1.87)	3.11 (±0.42)
LL	101.42 (±3.46)	21.96 (±3.38)	109.60 (±2.06)	27.10 (±4.95)	117.83 (±2.37)	28.37 (±3.53)
Neural	60.87 (±2.87)	**1.50** (±0.20)	79.77 (±1.63)	**1.43** (±0.78)	90.01 (±1.18)	**1.63** (±0.47)
TF	67.06 (±2.91)	4.63 (±2.00)	77.95 (±1.74)	2.80 (±0.72)	86.38 (±1.88)	3.68 (±1.33)

Table 2: Learnability of general and dense representation-based n-gram LMs for $n = 6$.

n	4			8			12		
$|\Sigma|$	64	128	256	64	128	256	64	128	256
Classic	2.11 (±0.39)	3.40 (±0.81)	5.00 (±0.72)	25.72 (±8.48)	54.95 (±4.17)	67.64 (±5.25)	58.17 (±9.36)	74.09 (±6.78)	89.90 (±6.96)
LL	32.05 (±3.14)	37.10 (±9.53)	40.61 (±3.93)	40.45 (±11.01)	58.26 (±3.64)	62.00 (±5.23)	43.40 (±5.46)	49.75 (±4.12)	61.60 (±5.45)
Neural	**1.14** (±0.64)	**1.98** (±0.91)	**2.17** (±1.46)	**5.96** (±1.86)	**9.13** (±0.88)	**9.06** (±0.68)	**7.71** (±0.75)	**9.87** (±1.28)	**11.20** (±1.09)
TF	2.43 (±0.51)	5.04 (±1.75)	5.63 (±1.80)	9.52 (±1.98)	14.72 (±0.95)	16.89 (±1.33)	14.73 (±1.70)	22.79 (±6.42)	33.99 (±9.06)

Table 3: The effect of the order n and the alphabet size $|\Sigma|$ on the performance of the best performing models.

Predictor	$\widehat{\beta}$	p-value
Intercept	0.99 (±0.03)	$< 10^{-6}$
n	0.63 (±0.01)	$< 10^{-6}$
$|\Sigma|$	0.28 (±0.01)	$< 10^{-6}$
R	0.17 (±0.02)	$< 10^{-6}$
H (p_n)	-0.05 (±0.02)	0.034
Dense	-1.28 (±0.04)	$< 10^{-6}$
L	-0.33 (±0.01)	$< 10^{-6}$
H	-0.15 (±0.01)	$< 10^{-6}$
sparsemax	-0.06 (±0.02)	0.002

Table 4: Estimated coefficients ($\widehat{\beta}$) and p-values of the linear model predicting $\widetilde{D_{\mathrm{KL}}}$. The R^2 value is 0.665.

formers made by Svete and Cotterell (2024) (and the distinction between soft- and hard-attention transformers in theoretical literature) makes the impact of using sparse attention particularly interesting. As shown by Tab. 4, using sparse attention is a significant predictor of *better* performance, which agrees with the intuition that zero attention weights are useful for representing n-gram LMs. This is further investigated in App. C.3, showing the superior performance of sparse-attention transformers. While these results agree with the intuition regarding the importance of sparse attention made by Svete and Cotterell (2024), they depart from

the softmax-computed attention weights normally used in practice (Vaswani et al., 2017). This encourages further investigation into the possible benefits of sparse attention mechanisms in transformers, particularly in the context of learning formal languages—could sparse attention, for example, aid generalization in situations where non-zero attention weights could lead to accumulating errors, such as those in counting (Weiss et al., 2018)?

5 Conclusion

We compare the performance of transformers to that of classical n-gram estimation techniques and hand-crafted baselines at learning artificially generated n-gram LMs. Transformers show a good inductive bias towards learning representation-based n-gram LMs but fare worse than classic estimation techniques at learning general n-gram LMs, underlining the importance of parameter sharing in the ground-truth model for transformers to learn it well. Moreover, the impact of the number of heads and layers on the performance agrees with existing theoretical results. The better performance of sparse-attention transformers motivates further investigation into how sparse attention could be used when learning formal languages with transformers.

Limitations

We highlight some limitations of our work. First, note that the concrete theoretical results this paper investigates rely on a formulation of the transformer architecture that is not practical for training—most notably, Svete and Cotterell's (2024) results either assume the use of hard attention or the use of the more practical sparse attention but unbounded positional encodings, which are not commonly used in practice. We replace these modeling assumptions with more practical ones—we use the softmax and sparsemax normalization functions and bounded positional encodings. We also note that the LMs we use to evaluate transformers—n-gram LMs—are very simple, which makes the results less generalizable to more complex LMs. This is done on purpose to stay as close as possible to the theoretical results connecting transformers to n-gram LMs.

Acknowledgements

References

Ekin Akyürek, Bailin Wang, Yoon Kim, and Jacob Andreas. 2024. In-context language learning: Architectures and algorithms. *arXiv preprint arXiv:2401.12973*.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. 2024. Zoology: Measuring and improving recall in efficient language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2000. A neural probabilistic language model. In *Advances in Neural Information Processing Systems*, volume 13. MIT Press.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155.

Yoshua Bengio, Holger Schwenk, Jean-Sébastien Senécal, Fréderic Morin, and Jean-Luc Gauvain. 2006. *Neural Probabilistic Language Models*, pages 137–186. Springer Berlin Heidelberg, Berlin, Heidelberg.

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.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Nadav Borenstein, Anej Svete, Robin 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. *arXiv preprint arXiv:2406.04289*.

Angelica Chen, Ravid Shwartz-Ziv, Kyunghyun Cho, Matthew L. Leavitt, and Naomi Saphra. 2024. Sudden drops in the loss: Syntax acquisition, phase transitions, and simplicity bias in mlms. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Stanley F. Chen and Joshua Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–394.

Ryan Cotterell, Anej Svete, Clara Meister, Tianyu Liu, and Li Du. 2024. Formal aspects of language modeling. *arXiv preprint arXiv:2311.04329*.

Grégoire Delétang, 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 *11th International Conference on Learning Representations*.

Benjamin L. Edelman, Ezra Edelman, Surbhi Goel, Eran Malach, and Nikolaos Tsilivis. 2024. The evolution of statistical induction heads: In-context learning markov chains. *arXiv preprint arXiv:2402.11004*.

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.

William A. Gale and Geoffrey Sampson. 1995. Good-turing frequency estimation without tears. *Journal of Quantitative Linguistics*, 2(3):217–237.

S. Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Jiacheng Liu, Sewon Min, Luke Zettlemoyer, Yejin Choi, and Hannaneh Hajishirzi. 2024. Infini-gram: Scaling unbounded n-gram language models to a trillion tokens. *arXiv preprint arXiv:2401.17377*.

Ilya Loshchilov and Frank Hutter. 2018. Fixing weight decay regularization in Adam. *arXiv.org*.

Luca Malagutti, Andrius Buinovskij, Anej Svete, Clara Meister, Afra Amini, and Ryan Cotterell. 2024. The role of n-gram smoothing in the age of neural networks. *arXiv preprint arXiv:2403.17240*.

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.

Hermann Ney, Ute Essen, and Reinhard Kneser. 1994. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, 8(1):1–38.

Timothy Nguyen. 2024. Understanding transformers via n-gram statistics. *arXiv preprint arXiv:2407.12034*.

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. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7011–7034, Singapore. Association for Computational Linguistics.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. 2022. In-context learning and induction heads. *Transformer Circuits Thread*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.

Alec Radford, Jeff Wu, Rewon Child, D. Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

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*.

Simeng Sun and Mohit Iyyer. 2021. Revisiting simple neural probabilistic language models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5181–5188, Online. Association for Computational Linguistics.

Anej Svete, Robin Shing Moon Chan, and Ryan Cotterell. 2024a. 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.

Anej Svete and Ryan Cotterell. 2024. Transformers can represent n-gram language models. *arXiv preprint arXiv:2404.14994*.

Anej Svete, Franz Nowak, Anisha Sahabdeen, and Ryan Cotterell. 2024b. Lower bounds on the expressivity of recurrent neural language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6820–6844, Mexico City, Mexico. Association for 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.

Elena Voita, Javier Ferrando, and Christoforos Nalmpantis. 2024. Neurons in large language models: Dead, n-gram, positional. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 1288–1301, Bangkok, Thailand and virtual meeting. 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.

Ian H. Witten and Timothy C. Bell. 1991. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1084–1094.

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 Classic n-gram Estimation Techniques

A.1 Maximum Likelihood Estimation (MLE)

The maximum likelihood n-gram LM estimate of order n computes the next-symbol probabilities

$$q_{\text{MLE}}^n(y \mid \boldsymbol{y}^n) \overset{\text{def}}{=} \frac{\#(\boldsymbol{y}^n y)}{\#(\boldsymbol{y}^n)}, \tag{6}$$

for $y \in \overline{\Sigma}$ and $\boldsymbol{y}^n \in \Sigma^{n-1}$, which counts the total number of times the string $\boldsymbol{y}^n y \in \Sigma^n$ occurs and divides by the number of total occurrences of the history $\boldsymbol{y}^n \in \Sigma^{n-1}$. MLE models are prone to overfitting, especially when training data is sparse or if history lengths are long, thus demonstrating the need for smoothing.

A.2 Add-λ smoothing (Add-λ)

Add-λ smoothing is one of the simplest methods of obtaining a smoothed n-gram model from the MLE estimate q_{MLE}^n. Add-λ smoothing adds a pseudo-count of λ to the counts of all possible n-grams, including those not seen in the training dataset. Mathematically,

$$\#_{\text{AS}}(\boldsymbol{y}^n y) \overset{\text{def}}{=} \#(\boldsymbol{y}^n y) + \lambda \tag{7}$$

and

$$\tilde{q}_{\text{AS}}^n(y \mid \boldsymbol{y}^n) \overset{\text{def}}{=} \frac{\#(\boldsymbol{y}^n y) + \lambda}{\#(\boldsymbol{y}^n) + |\Sigma + 1|\lambda} \tag{8}$$

for $y \in \overline{\Sigma}$ and $\boldsymbol{y}^n \in \Sigma^{n-1}$. In the special case of $\lambda = 1$, we have **Laplace Smoothing**.

A.3 Absolute Discounting (AD) (1994)

Absolute Discounting (AD) involves the interpolation of higher and lower-order n-gram models. Though higher-order n-grams capture more context, they often suffer from zero probabilities and overfitting due to limited training data. AD makes higher-order distributions by subtracting a fixed discount $\delta \leqslant 1$ from each nonzero count and recursively interpolates this with n-gram of lower degree. That is:

$$\tilde{q}_{\text{AD}}^n(y \mid \boldsymbol{y}^n) = \frac{\max\{\#(\boldsymbol{y}^n y) - \delta, 0\}}{\#(\boldsymbol{y}^n)} + (1 - \lambda_n)\tilde{q}_{\text{AD}}^{n-1}(y \mid \boldsymbol{y}^{n-1}) \tag{9}$$

for $y \in \overline{\Sigma}$ and $\boldsymbol{y}^n \in \Sigma^{n-1}$. To make the probabilities sum to 1, we let

$$1 - \lambda_n = \frac{\delta \#_{\text{T}}^n(\boldsymbol{y}^n, \bullet)}{\#(\boldsymbol{y}^n)}, \tag{10}$$

where $\#_{\text{T}}^n$ is the *type counts* function of order n, formally defined as

$$\#_{\text{T}}^n(\boldsymbol{y}^n, y) \overset{\text{def}}{=} \begin{cases} 1 & \textbf{if } \#(\boldsymbol{y}^n y) > 0 \\ 0 & \textbf{otherwise} \end{cases} \tag{11}$$

and $\#_{\text{T}}^n(\bullet, y)$, $\#_{\text{T}}^n(\boldsymbol{y}^n, \bullet)$ and $\#_{\text{T}}^n(\bullet, \bullet)$ are type count functions with bulleted arguments summed out.

A.4 Witten–Bell Smoothing (WB) (1991)

Though interpolating higher-order n-gram models with lower-order n-gram models helps handle unseen n-grams, it can sometimes produce undesirable results, especially if an n-gram occurs only in a specific context. A common illustrative example in the literature (Chen and Goodman, 1999) is the bigram *San Francisco*. If the bigram *San Francisco* shows up frequently in the data set, the word *Francisco* will have a high unigram count, thus assigning a higher probability of continuation to *Francisco* when interpolating higher-order n-grams with lower order count functions. This, however, may be undesirable, as the term *Francisco* will rarely follow any context other than the word *San*. Witten–Bell smoothing addresses this by

9859

considering the number of unique continuations a context has observed, based on the intuition that backing off to lower-order n-gram statistics is more accurate only when there are more distinct continuations of a given context. The interpolation works as follows:

$$\tilde{q}_{\mathrm{WB}}^n(y \mid \boldsymbol{y}^n) = \lambda_n q_{\mathrm{MLE}}^n(y \mid \boldsymbol{y}^n) + (1 - \lambda_n)\tilde{q}_{\mathrm{WB}}^{n-1}(y \mid \boldsymbol{y}^{n-1}) \tag{12}$$

for $y \in \overline{\Sigma}$ and $\boldsymbol{y}^n \in \Sigma^{n-1}$ where

$$1 - \lambda_n = \frac{\#_{\mathrm{T}}^n(\boldsymbol{y}^n, \bullet)}{\#_{\mathrm{T}}^n(\boldsymbol{y}^n, \bullet) + \#(\boldsymbol{y}^n)}. \tag{13}$$

Substituting, we rewrite the interpolation as

$$\tilde{q}_{\mathrm{WB}}^n(y \mid \boldsymbol{y}^n) = \frac{\#(\boldsymbol{y}^n y) + \#_{\mathrm{T}}^n(\boldsymbol{y}^n, \bullet)\tilde{q}_{\mathrm{WB}}^{n-1}}{\#_{\mathrm{T}}^n(\boldsymbol{y}^n, \bullet) + \#(\boldsymbol{y}^n)}. \tag{14}$$

A.5 A Log-Linear Model

As a simple parameter-sharing representation-based baseline, we consider a log-linear model that represents the history $\boldsymbol{y}^n$ as a concatenation of the one-hot encodings of the symbols in $\boldsymbol{y}^n$.[8] Concretely, the model represents the history $\boldsymbol{y}^n = y_1 \cdots y_{n-1}$ as

$$\boldsymbol{h}(y_1 \cdots y_{n-1}) \stackrel{\text{def}}{=} \begin{pmatrix} [\![y_1]\!] \\ \vdots \\ [\![y_{n-1}]\!] \end{pmatrix} \in \{0, 1\}^{(n-1)|\Sigma|}. \tag{15}$$

Its parameter is an output matrix $\mathbf{E} \in \mathbb{R}^{|\Sigma| \times (n-1)|\Sigma|}$, which determines the logits of the conditional distribution $p_n(y \mid \boldsymbol{y}^n)$ as

$$p_n(y \mid \boldsymbol{y}^n) \stackrel{\text{def}}{=} \mathrm{softmax}(\mathbf{E}\,\boldsymbol{h}(\boldsymbol{y}^n))_y. \tag{16}$$

The model is trained by minimizing the cross-entropy loss between the true conditional distributions p_n and the predicted distributions p_n.

A.6 A Neural n-gram LM

The neural n-gram LM is a classic neural LM popularized by Bengio et al. (2000, 2003, 2006) and modernized by Sun and Iyyer (2021). It learns D'-dimensional word2vec-style static representations of the symbols and concatenates them before feeding them through an MLP $\boldsymbol{f}$.[9] The MLP thus produces the final representation of the history, which is used to define the next-symbol probabilities together with the learned output matrix $\mathbf{E}$ as

$$p(y \mid \boldsymbol{y}^n) \stackrel{\text{def}}{=} \mathrm{softmax}(\mathbf{E}\,\boldsymbol{h}(\boldsymbol{y}_{<t}))_y \tag{17a}$$

$$= \mathrm{softmax}\left(\mathbf{E}\,\boldsymbol{f}\left(\begin{pmatrix} [\![y_1]\!] \\ \vdots \\ [\![y_{n-1}]\!] \end{pmatrix} \right) \right)_y. \tag{17b}$$

The size of the symbol representations as well as the complexity of the MLP $\boldsymbol{f}$ (its depth and width) are hyperparameters of the model.

---

[8]This exactly matches the representations used to *generate* sparse representation-based n-gram LMs described in App. B.1.1 and thus provides a strong inductive bias for those models, making the log-linear model a strong baseline.

[9]The neural model studied by Sun and Iyyer (2021) adds a pooling operation over the entire preceding string. However, to keep the model truly n-gram-based, we do not include this pooling operation and ignore the symbols further than $n-1$ steps in the past.

B Experimental Details

B.1 Data Generation

B.1.1 Generating n-gram LMs

We generate the training and test datasets by randomly generating n-gram LMs and sampling strings from them. We construct three types of n-gram LMs: 1. General n-gram LMs whose conditional distributions $p_n(y \mid \boldsymbol{y}^n)$ are sampled independently of each other. 2. Sparse representation-based n-gram LMs. 3. Dense representation-based n-gram LMs.

General n-gram LMs. We sample general n-gram LMs by sampling, for each possible context $\boldsymbol{y}^n$ (including the contexts padded with different numbers of BOS symbols), a conditional distribution $p(y \mid \boldsymbol{y}^n)$ for $y \in \Sigma$. The conditional distributions are sampled from a Dirichlet distribution with concentration parameter α, where we set $\alpha = 0.1$ to encourage concentrated distributions. We control the expected length of the string $\mathbb{E}[|\boldsymbol{y}|]$ generated by the n-gram LM by hard-coding the probability $p_n(\text{EOS} \mid \boldsymbol{y}^n)$ to be $\frac{1}{\mathbb{E}[|\boldsymbol{y}|]}$ for all $\boldsymbol{y}^n$, where we set $\mathbb{E}[|\boldsymbol{y}|] = 40$. Due to the requirement of storing all the $|\Sigma|^{n-1}$ conditional distributions, we limit ourselves to the case of $n \in \{2, 4, 6\}$ and $|\Sigma| \in \{8, 12, 16\}$ for the general n-gram LM case. This procedure is described with pseudocode in Alg. 1.

Algorithm 1 The generation of a random general n-gram LM.

1. **def** GENERATE GENERAL n-GRAM LM($n, \Sigma, \alpha, \mathbb{E}[|\boldsymbol{y}|]$):
2.   **for** $\boldsymbol{y}^n \in \bigcup_{j=0}^{n-1} \{\text{BOS}\}^j \times \Sigma^{n-1-j}$:
3.   ▷ *Iterate through all possible contexts, including* BOS-*padded contexts.*
4. $p_n(y \mid \boldsymbol{y}^n) \sim \text{DIRICHLET}(\alpha \mathbf{1}_{|\Sigma|})$ for $y \in \Sigma$
5. $p_n(\text{EOS} \mid \boldsymbol{y}^n) \leftarrow \frac{1}{\mathbb{E}[|\boldsymbol{y}|]}$
6.   **Renormalize** $p_n(\overline{y} \mid \boldsymbol{y}^n)$ for $\overline{y} \in \overline{\Sigma}$
7.   **return** p_n

Representation-based n-gram LMs. Representation-based n-gram LMs are generated by defining the conditional distributions $p_n(y \mid \boldsymbol{y}^n)$ in terms of an **output matrix** $\mathbf{E}$ which transforms the vectorial **representations** $\boldsymbol{h}(\boldsymbol{y}^n) \in \mathbb{R}^D$ of the history $\boldsymbol{y}^n$ into the (logits of the) conditional distribution $p_n(y \mid \boldsymbol{y}^n)$. More concretely, let $\boldsymbol{h} \colon \Sigma^{n-1} \to \mathbb{R}^D$ be an representation function that maps the history $\boldsymbol{y}^n$ to a vector in $\mathbb{R}^D$. Then, we define the conditional distribution $p_n(y \mid \boldsymbol{y}^n)$ for $y \in \Sigma$ as

$$p_n(y \mid \boldsymbol{y}^n) \overset{\text{def}}{=} \text{softmax}(\mathbf{E}\,\boldsymbol{h}(\boldsymbol{y}^n))_y, \tag{18}$$

where $\mathbf{E} \in \mathbb{R}^{|\Sigma| \times D}$ is an output matrix. Once again, we hard-code the probability of the end-of-string symbol EOS to be $\frac{1}{\mathbb{E}[|\boldsymbol{y}|]} = \frac{1}{40}$ for all $\boldsymbol{y}^n$ by post-hoc renormalizing the conditional distributions at every time step.

We consider two representation functions $\boldsymbol{h}$:

1. **Sparse**: We define the sparse representation function $\boldsymbol{h} \colon \Sigma^{n-1} \to \{0, 1\}^{(n-1)|\Sigma|}$ as the function mapping the history $y_1 \cdots y_{n-1}$ to the concatenation of the one-hot encodings of its symbols. In symbols, this mean

$$\boldsymbol{h}(y_1 \cdots y_{n-1}) \overset{\text{def}}{=} \begin{pmatrix} [\![y_1]\!] \\ \vdots \\ [\![y_{n-1}]\!] \end{pmatrix} \in \{0, 1\}^{(n-1)|\Sigma|} \tag{19}$$

where $[\![y]\!] \in \{0, 1\}^{|\Sigma|}$ is the one-hot encoding of the symbol y. Notice that in this case, the size of the representations grows with the size of the alphabet.

2. **Dense**: We define a dense representation function $\boldsymbol{h} \colon \Sigma^{n-1} \to \mathbb{R}^{(n-1)D'}$ as the function mapping the history $y_1 \cdots y_{n-1}$ to the concatenation of *dense* embeddings $\mathbf{r}(y) \in \mathbb{R}^{D'}$ of its symbols. We

(a) Distribution of the entropy of the datasets from representation-based n-gram LMs.

(b) Distribution of the entropy of the datasets from the general n-gram LMs.
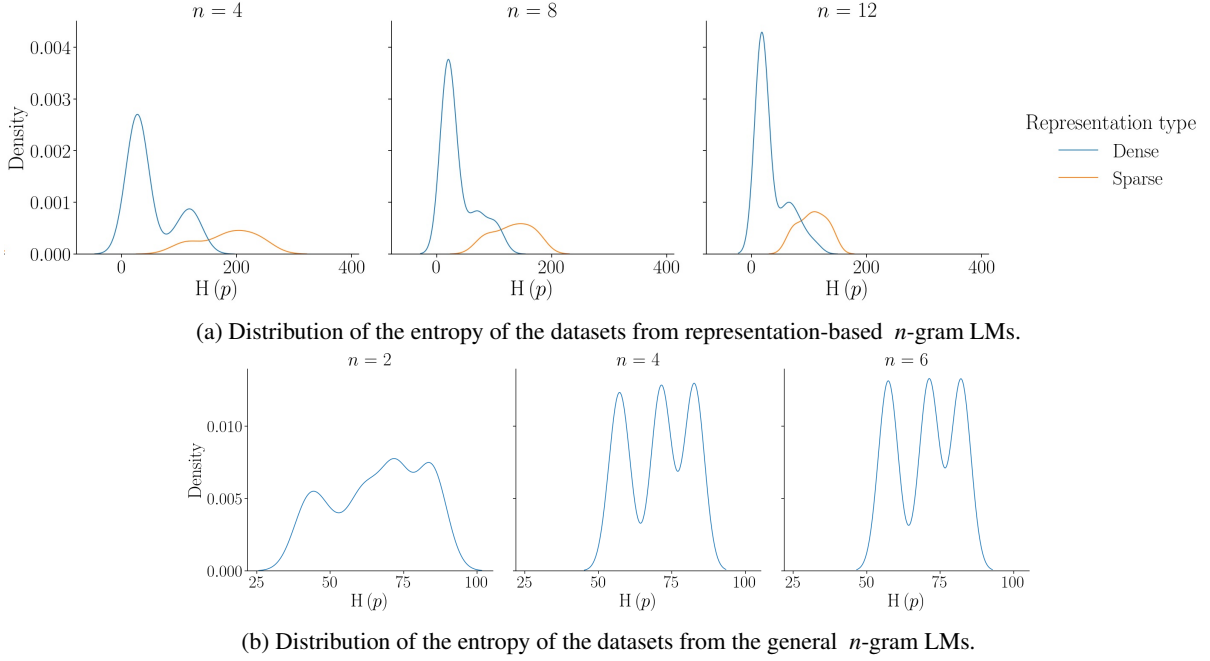
Figure 1: Distributions of the mean string lengths and the LM entropies for the 45 generated datasets.

generate the symbol embeddings randomly from a standard normal distribution. We use $D' = 16$ for the dense representation of symbols. Furthermore, since the *rank* of the output matrix $\mathbf{E}$ was found to be a significant predictor of the learnability of general regular LMs (Borenstein et al., 2024), we control for the rank of the output matrix generating $\mathbf{E}$ by constructing $\mathbf{E} = \mathbf{E}_1 \mathbf{E}_2$ the product of two random matrices of $\mathbf{E}_1 \in \mathbb{R}^{|\Sigma| \times R}$ and $\mathbf{E}_2 \in \mathbb{R}^{R \times D}$, resulting in a rank-$R$ matrix of size $|\Sigma| \times D$, where we vary the rank as $R \in \{2, 8, 16\}$. This results in a fixed-size representation of the history, regardless of the size of the alphabet.

The randomly-generated matrices ($\mathbf{E}$ in the sparse case and $\mathbf{E}_1$ and $\mathbf{E}_2$ in the dense case) are generated by sampling their entries independently from a standard normal distribution.

The main difference between sparse and dense representation functions is therefore the degree of parameter sharing between the different symbols in the history. Since, unlike in the general n-gram LM case, we do not need to store the conditional distributions for all possible histories, we can consider larger values of n and $|\Sigma|$. In particular, we generate datasets with $n \in \{4, 8, 12\}$ and $|\Sigma| \in \{64, 128, 256\}$.

B.1.2 Generating Train and Test Datasets

The training and test datasets are generated by sampling strings from the n-gram LMs. Let p_n be an n-gram LM. We sample *disjoint* training and test datasets from p_n in a multi-step process:

(1.) Sample a large set of strings $\mathcal{D}'$ from p_n (where strings are not repeated).

(2.) Divide $\mathcal{D}' = \mathcal{D}'_{\text{Train}} \sqcup \mathcal{D}'_{\text{Train}}$ into the set of strings $\mathcal{D}'_{\text{Train}}$ that are allowed to appear in the training dataset and a set of strings $\mathcal{D}'_{\text{Train}}$ that are allowed to appear in the test dataset.

(3.) Sample two large (multi-)sets of strings $\mathcal{D}_{\text{Train}}$ and $\mathcal{D}_{\text{Test}}$ from p_n.

(4.) Remove all strings from $\mathcal{D}_{\text{Train}}$ that are in $\mathcal{D}'_{\text{Test}}$ and all strings from $\mathcal{D}_{\text{Test}}$ that are in $\mathcal{D}'_{\text{Train}}$.

(5.) Retain N_{Train} strings from $\mathcal{D}_{\text{Train}}$ and N_{Test} strings from $\mathcal{D}_{\text{Test}}$.

This procedure ensures that the training and test datasets are disjoint and that the test dataset is not seen during training.

Fig. 1 shows the distributions of the entropies for the generated datasets.

Smoothing Technique	Hyperparameter	Values
Add-λ smoothing (Add-λ)	λ	$0.01, 0.1, 1$
Absolute Discounting (AD)	δ	$0.6, 0.8, 0.95$

Table 5: Hyperparameters for the smoothing techniques.

B.2 Models

B.2.1 Transformer Models

We use the GPT-2 model architecture (Radford et al., 2019) of varying sizes. We sample the number of heads H the number of layers L from $\{1, n\}$, where n is the learnt n-gram's order. We also control for the type of attention activation function, using either the standard softmax or its variant sparsemax. Moreover, we use an embedding size of 256, a hidden representation size of 512, and an output size (the dimensionality of the final linear layer) of 128. The number of trainable parameters of the models ranges between 500k and 4M. We train each model for 10 epochs with an input context length of 256 using a batch size of 128 and a learning rate of 0.0001, an AdamW (Loshchilov and Hutter, 2018) optimizer with default settings, and a standard cross-entropy loss. We did not tie the weights of the word embeddings. Training a single instance took, on average, 30 minutes, with some variations depending on the size of the dataset and exact architecture size.

B.2.2 Smoothing Techniques

We train a number of smoothing techniques on the generated datasets as a baseline for the transformers. We use the smoothing techniques described in App. A. Each of them allows us to control the order of the learned n-gram LM, $\hat{n}$. For each setting, we test out three scenarios: An under-parametrized model ($\hat{n} = n - 2$), a well-parametrized model ($\hat{n} = n$), and an over-parametrized model ($\hat{n} = 2n$ for $n \in \{4, 8\}$ and $\hat{n} = 20$ for $n = 12$).[10] This results in models between with a large number of parameters—the largest model on $|\Sigma| = 256$ symbols of order $\hat{n} = 20$ defines in the order of 10^48 parameters—but note that only the parameters corresponding to observed n-grams are memorized, making the training feasible. Some of the estimation techniques—MLE and Witten–Bell—do not have any hyperparameters and are trained with the default settings. For the other techniques (Add-λ smoothing and Absolute discounting), we train models with the hyperparameters listed in Tab. 5 and report the performance of the best-performing models. We use the standard nltk implementations (Bird et al., 2009) for these estimation techniques.

B.2.3 Log-Linear Model and Neural n-gram LM

We implemented the log-linear model and the neural n-gram LM in PyTorch (Paszke et al., 2019). The log-linear model only learns the output matrix $\mathbf{E}$ while our implementation of the neural LM closely follows the model defined by Bengio et al. (2000, 2003, 2006) with the modern improvements studied by Sun and Iyyer (2021).[11] As with the smoothing techniques, we study the effects of the order $\hat{n}$ of the trained LM on the performance of the log-linear model and the neural n-gram LM. Thus, we fit models with $\hat{n} \in \{n - 2, n, \min(2n, 20)\}$, resulting with models with between 4k and 1.25M parameters. The models are trained by minimizing the cross-entropy loss between the empirical distribution of the training dataset and the predicted distributions p. For the log-linear model, we use the Adam optimizer (Kingma and Ba, 2017) with a learning rate of 0.1 and a batch size of 1024. We train the models for 16 epochs and halve the learning rate after every five epochs. The neural n-gram LMs were implemented as a shallow neural network with three learned layers. The first is an embedding later with a dimensionality of 128. The second is a fully connected layer with an output dimensionality of 512, ReLU activation function, and dropout probability of 0.5. The third layer is a softmax-normalized linear transformation for predicting next-symbol probabilities, i.e., the matrix $\mathbf{E}$. The number of trainable parameters of the models ranges between 400k and 1.5M. They were trained for 20 epochs with early stopping on a development set that

---

[10]While transformer models do not have an analogous hyperparameter to make them fit the ground-truth n-gram LM perfectly, we vary their sizes (number of heads and layers) to find the best-performing model, as motivated by theoretical work.

[11]Since we are interested in n-gram LMs, we do not implement the pooling of the longer context from Sun and Iyyer (2021).

was sampled from the training set (80%–20% split) using a batch size of 128 and, learning rate of $5e^{-5}$, and Adam optimizer with default parameters. Training of a single instance took, on average, 5 minutes, with some variations depending on the size of the dataset and model.

B.3 Evaluation

As described in §3.1, we evaluate the learned LMs by computing the KL divergence between the learned LM p and the ground-truth n-gram LM p_n. Concretely, we can compute $D_{\mathrm{KL}}\left(p_n \parallel p\right)$ through its decomposition as

$$D_{\mathrm{KL}}\left(p_n \parallel p\right) = \mathrm{H}(p_n, p) - \mathrm{H}(p_n). \tag{20}$$

While $\mathrm{H}\left(p_n\right)$ can be computed analytically with a dynamic program (Eisner, 2002; Zmigrod et al., 2021), the runtime complexity of $\mathcal{O}\left(\left(\left|\overline{\Sigma}\right|^{n-1}\right)^3\right)$ makes exact computations infeasible for even moderately large $\left|\overline{\Sigma}\right|$ and n. We thus rely on empirical estimates of both terms in Eq. (20) on the test dataset. Concretely, we compute

$$\widehat{\mathrm{H}}\left(p_n\right) = -\frac{1}{N_{\mathrm{Test}}} \sum_{\boldsymbol{y} \in \mathcal{D}_{\mathrm{Test}}} \log p_n\left(\boldsymbol{y}\right) \tag{21a}$$

$$\widehat{\mathrm{H}}\left(p_n, p\right) = -\frac{1}{N_{\mathrm{Test}}} \sum_{\boldsymbol{y} \in \mathcal{D}_{\mathrm{Test}}} \log p\left(\boldsymbol{y}\right), \tag{21b}$$

which allows us to compute the empirical approximation of $D_{\mathrm{KL}}\left(p_n \parallel p\right)$,

$$\widehat{D_{\mathrm{KL}}}\left(p_n \parallel p\right) \stackrel{\text{def}}{=} \widehat{\mathrm{H}}(p_n, p) - \widehat{\mathrm{H}}(p_n). \tag{22}$$

B.4 Statistical Analysis

To assess the influence of the predictors specified in Tab. 1 on the learnability of the n-gram LMs (i.e, the empirical KL divergence on the test dataset), we implement a linear regression model predicting the KL divergence based on the predictors. Before fitting the model, we standardize each parameter with a z-score transformation for an interpretable comparison of the estimated coefficients.

The linear regression model provides insights into the influence of each predictor on the dependent variable (KL divergence) by assigning each parameter three key values: the coefficient of the linear model $\hat{\beta}_i$, the standard deviation of the coefficient, and a p-value. The magnitude of the coefficient $\hat{\beta}_i$ indicates the strength of the predictor's effect on the dependent variable. The coefficient's *sign* reveals whether this effect is positive (an increase in the predictor is expected to increase the value of the parameter; in our case, this indicates that the increase of the parameter is associated with a *worse* performance of the model and vice-versa) or negative (an increase in the predictor is expected to decrease the value of the parameter; in our case, this indicates that the increase of the parameter is associated with a *better* performance of the model and vice-versa). The standard deviation measures the variability or uncertainty of the $\hat{\beta}_i$ coefficient, providing a sense of the reliability of this estimate. The p-value quantifies the statistical significance of the effect, indicating the likelihood that the observed relationship occurred by chance. It thus provides a measure of the reliability of the effect, with lower p-values indicating a more reliable effect.

C Additional Results

C.1 The Effect of Parameter Sharing

In §4.1 (particularly, in Tab. 2), we compared the performance of the models of interest across general and dense representation-based n-gram LMs. As expected, neural LMs fare much better with representation-based n-gram LMs, whereas the difference in the performance of counting-based methods is smaller. Tabs. 6a and 6b provide results for additional model orders and alphabet sizes, showing the same trends as Tab. 2—all learning methods perform better on representation-based n-gram LMs across all settings.

The next natural question is whether the degree to which the parameters are shared affects the performance as well. To determine that, Tab. 7a shows the performance on one-hot representation-based

n	2			4			6		
$|\Sigma|$	8	12	16	8	12	16	8	12	16
Classic	−0.08 (±0.94)	−0.35 (±0.78)	0.18 (±1.49)	−1.00 (±0.39)	0.23 (±1.41)	1.86 (±1.96)	3.04 (±1.46)	17.34 (±1.01)	50.35 (±1.87)
LL	35.31 (±7.55)	50.57 (±12.27)	59.76 (±4.96)	86.03 (±2.05)	103.36 (±2.95)	111.95 (±5.41)	101.42 (±3.46)	109.60 (±2.06)	117.83 (±2.37)
Neural	−0.19 (±1.04)	−0.46 (±0.81)	0.07 (±1.57)	0.91 (±0.51)	16.29 (±1.74)	40.00 (±2.78)	60.87 (±2.87)	79.77 (±1.63)	90.01 (±1.18)
TF	0.07 (±1.09)	−0.48 (±0.82)	0.18 (±1.50)	0.18 (±0.53)	4.68 (±1.61)	10.98 (±2.35)	67.06 (±2.91)	77.95 (±1.74)	86.38 (±1.88)

(a) Learnability of general n-gram LMs.

n	2			4			6		
$|\Sigma|$	8	12	16	8	12	16	8	12	16
Classic	−0.25 (±0.53)	−0.35 (±1.47)	0.55 (±1.00)	0.72 (±0.66)	0.89 (±0.69)	1.28 (±0.90)	2.36 (±0.23)	2.45 (±0.96)	3.11 (±0.42)
LL	25.23 (±9.12)	26.83 (±9.12)	29.10 (±7.82)	19.67 (±4.28)	24.03 (±5.47)	26.55 (±6.62)	21.96 (±3.38)	27.10 (±4.95)	28.37 (±3.53)
Neural	−0.39 (±0.60)	−0.52 (±1.73)	0.26 (±0.41)	0.61 (±0.65)	0.35 (±0.66)	1.15 (±0.72)	1.50 (±0.20)	1.43 (±0.78)	1.63 (±0.47)
TF	1.45 (±2.34)	1.58 (±3.83)	1.34 (±0.35)	4.55 (±3.02)	1.82 (±0.49)	3.46 (±1.15)	4.63 (±2.00)	2.80 (±0.72)	3.68 (±1.33)

(b) Learnability of small representation-based n-gram LMs.

Table 6: Learnability of general and (small) representation based n-gram LMs.

n-gram LMs. For an easier comparison to the results on dense representation-based n-gram LMs, the results from Tab. 3 are reproduced in Tab. 7b. Transformers perform better on dense representation-based n-gram LMs across all settings, as expected, since those more closely fit their modeling assumptions. In fact, transformers do not perform much better than classical smoothing techniques on some of the configurations. While dense-representation-based models are also better learned by smoothing methods, the log-linear model interestingly performs *better* on the sparse n-gram LMs (apart from $n = 4$). This is in line with the specification of the log linear model: The model *a priori* assumes a sparse-representation-based n-gram LM, and thus fits the model specifications well, which again shows the effect of correct model specification.

C.2 The Effect of the Rank

In §4.2, we investigated the trends in the performance with respect to the order of the n-gram model and the size of the alphabet (cf. Tab. 3). Here, we also consider the rank R of $\mathbf{E}$. The results for varying ranks are presented in Tab. 8. They again follow the intuitions from theory. The performance smoothing methods and the log-linear model, which can by design implement any-rank dense-representation-based n-gram LM, is unaffected by R. The performance of neural n-grams and transformers (which in our experiments all have rank at most 128—the output dimension), however, is negatively correlated with the rank. This is confirmed by the analysis of the predictors of the transformer performance (cf. Tab. 4), although the effect is not as significant and strong as for the other predictors.

C.3 The Effect of Sparse Attention

Tab. 4 suggests a significant impact of the use of sparse attention on the ability to learn n-gram LMs. This is confirmed by looking at the difference in the performance of soft- and sparse-attention transformers in Tab. 9.

n	4			8			12		
$\mid\Sigma\mid$	64	128	256	64	128	256	64	128	256
Classic	16.78 (±3.99)	28.67 (±2.20)	47.18 (±7.87)	76.08 (±8.97)	94.45 (±7.19)	112.85 (±8.70)	104.57 (±5.87)	130.03 (±6.96)	141.89 (±6.28)
LL	40.36 (±5.40)	42.44 (±2.34)	56.95 (±8.99)	36.33 (±8.23)	40.95 (±5.66)	53.01 (±7.57)	32.43 (±3.13)	40.01 (±3.97)	51.14 (±4.73)
Neural	−1.14 (±3.81)	**0.84** (±1.91)	**17.82** (±7.08)	**5.27** (±6.44)	**13.08** (±5.27)	**35.84** (±6.68)	**7.61** (±2.91)	**22.48** (±3.76)	**45.11** (±4.15)
TF	0.63 (±3.73)	6.90 (±1.93)	35.03 (±7.47)	10.36 (±6.95)	40.33 (±9.09)	98.79 (±10.55)	14.38 (±3.77)	70.50 (±14.64)	118.36 (±5.80)

(a) Learnability of sparse representation-based n-gram LMs with $R = 16$.

n	4			8			12		
$\mid\Sigma\mid$	64	128	256	64	128	256	64	128	256
Classic	2.11 (±0.39)	3.40 (±0.81)	5.00 (±0.72)	25.72 (±8.48)	54.95 (±4.17)	67.64 (±5.25)	58.17 (±9.36)	74.09 (±6.78)	89.90 (±6.96)
LL	32.05 (±3.14)	37.10 (±9.53)	40.61 (±3.93)	40.45 (±11.01)	58.26 (±3.64)	62.00 (±5.23)	43.40 (±5.46)	49.75 (±4.12)	61.60 (±5.45)
Neural	**1.14** (±0.64)	**1.98** (±0.91)	**2.17** (±1.46)	**5.96** (±1.86)	**9.13** (±0.88)	**9.06** (±0.68)	**7.71** (±0.75)	**9.87** (±1.28)	**11.20** (±1.09)
TF	2.43 (±0.51)	5.04 (±1.75)	5.63 (±1.80)	9.52 (±1.98)	14.72 (±0.95)	16.89 (±1.33)	14.73 (±1.70)	22.79 (±6.42)	33.99 (±9.06)

(b) Learnability of dense representation-based n-gram LMs with $R = 16$.

Table 7: Learnability of sparse and dense representation-based n-gram LMs.

$\mid\Sigma\mid$	64			128			256		
R	2	8	16	2	8	16	2	8	16
Classic	75.82 (±3.52)	72.86 (±5.39)	49.41 (±8.95)	79.71 (±9.06)	82.67 (±4.94)	68.35 (±5.64)	79.21 (±6.90)	94.94 (±8.97)	86.94 (±6.74)
LL	45.82 (±3.52)	47.98 (±4.49)	43.40 (±5.46)	48.94 (±4.10)	54.77 (±4.84)	49.75 (±4.12)	61.66 (±9.63)	60.69 (±5.13)	61.60 (±5.45)
Neural	2.28 (±2.48)	4.54 (±1.02)	7.71 (±0.75)	2.16 (±4.03)	5.08 (±1.04)	9.87 (±1.28)	0.31 (±4.79)	7.34 (±1.55)	11.20 (±1.09)
TF	5.49 (±24.78)	9.06 (±3.57)	13.96 (±1.93)	5.90 (±11.60)	11.05 (±1.61)	19.01 (±2.64)	4.34 (±30.98)	14.30 (±2.70)	25.15 (±2.33)

Table 8: The effect of the rank R on the performance of the best performing models for $n = 12$.

C.4 The Effect of Over-parametrization

The baselines used in the experiments in the main part of the paper (classic smoothing techniques, the log-linear model, and the neural n-gram model) are particularly well-specified for the LMs they approximate—in particular, they were trained with the correct order of the ground-truth n-gram LM, n. Apart from the intuitions offered by the theoretical constructions, such an appropriate parametrization is more difficult to specify for transformers, whose parameters do not match the n-gram definition as closely. In this section, we investigate how much possible misspecification of the baselines impacts their performance and compare it to the performance of the largest and smallest transformers trained.

To test the effect of *over-parametrization*, that is, assuming a too-large order $\widehat{n}$, Tab. 10a shows the performance of the best-performing baseline models with $\widehat{n} = 2n$ for $n \in \{4, 8\}$ and $\widehat{n} = 20$ for $n = 12$. Again, Tab. 7b serves as reference for the performance of the best models. Tab. 10a also shows the performance of the largest transformer models for each of the settings (that is, one where both the number of layers and heads are largest).[12] Over-parameterization is particularly harmful to models whose

---

[12]According to the theoretical constructions by Svete and Cotterell (2024), such transformers are over-parametrized—only

n	4			8			12		
$|\Sigma|$	64	128	256	64	128	256	64	128	256
softmax	2.58 (±0.52)	5.45 (±1.50)	6.52 (±1.73)	14.47 (±5.67)	22.37 (±7.95)	23.51 (±8.49)	23.74 (±8.82)	37.52 (±7.51)	47.79 (±7.64)
sparsemax	**2.43** (±0.51)	**5.04** (±1.75)	**5.63** (±1.80)	**9.52** (±1.98)	**14.72** (±0.95)	**16.89** (±1.33)	**14.73** (±1.70)	**22.79** (±6.42)	**33.99** (±9.06)

Table 9: Performance of soft- and sparse-attention transformers on dense representation n-gram LMs with $R = 16$.

number of parameters increases most—the classical smoothing techniques. Other models are less affected. In particular, the neural n-gram model performs as well as the optimally-parametrized variant, and transformers remain close to their best performance, with the exception of the largest model.

We contrast this to the effects of *under-parameterization*, where we set $\widehat{n} = n - 2$ and constrain the transformers to a single head and layer. The results of these runs are presented in Tab. 10b. Under-parametrization noticeably degrades the performance of most models and is most noticeable with the smaller orders n. Due to the fast growth of the number of parameters in the classical models, under-parametrization actually outperforms over-parameterized models, likely due to the parameter-sharing nature of the ground-truth n-gram LMs, possibly making them more easily approximatable with lower-order models. Single-head and single-layer transformers, in contrast, perform noticeably worse than their well- or over-parametrized variants, again in line with the intuitions from the theoretical constructions.

n	4			8			12		
$|\Sigma|$	64	128	256	64	128	256	64	128	256
Classic	3.49 (±0.30)	7.30 (±1.55)	10.21 (±0.75)	29.98 (±9.82)	61.97 (±4.51)	73.69 (±5.46)	50.22 (±9.03)	68.50 (±5.64)	87.01 (±6.85)
LL	33.69 (±3.14)	40.24 (±9.53)	43.49 (±4.84)	63.38 (±14.44)	83.96 (±9.37)	91.11 (±7.78)	66.45 (±9.08)	70.05 (±11.32)	93.89 (±17.32)
Neural	1.28 (±0.49)	2.27 (±0.98)	2.36 (±1.43)	6.26 (±1.87)	9.31 (±1.00)	9.14 (±0.77)	7.76 (±0.81)	9.91 (±1.29)	11.26 (±1.08)
TF	2.42 (±0.52)	5.02 (±2.24)	5.51 (±1.83)	10.33 (±1.85)	14.59 (±1.01)	17.39 (±1.13)	15.55 (±1.54)	26.52 (±6.95)	42.61 (±7.57)

(a) Results of the over-parametrized models on representation-based n-gram LMs.

n	4			8			12		
$|\Sigma|$	64	128	256	64	128	256	64	128	256
Classic	76.53 (±24.55)	84.94 (±42.59)	87.56 (±9.99)	22.41 (±8.48)	54.95 (±4.17)	67.64 (±5.25)	49.41 (±8.95)	68.36 (±5.67)	86.94 (±6.74)
LL	97.80 (±23.05)	105.58 (±45.45)	114.81 (±11.48)	47.93 (±16.66)	72.46 (±5.35)	73.03 (±6.23)	46.98 (±5.79)	52.33 (±4.12)	62.85 (±5.51)
Neural	76.61 (±22.81)	85.12 (±39.41)	87.71 (±9.32)	17.78 (±7.90)	36.78 (±2.82)	38.58 (±4.74)	18.08 (±3.03)	23.48 (±2.94)	28.10 (±2.16)
TF	3.17 (±0.62)	6.39 (±1.39)	8.21 (±1.84)	28.03 (±13.64)	65.44 (±6.39)	69.86 (±7.50)	50.79 (±10.58)	65.81 (±6.71)	81.96 (±7.43)

(b) Results of the under-parametrized models on representation-based n-gram LMs.

one of the number of heads or layers needs to increase.