

# Scaling Hidden Markov Language Models

March 2, 2021

## Abstract

Modern methods for language models based purely on neural networks are opaque and difficult to analyze, while alternative classical methods based on probabilistic graphical models are interpretable but not performant. We hypothesize that classical methods are not as performant because they have neither been scaled to similar sizes as their modern neural counterparts nor taken advantage of modern parallel hardware. A significant barrier to scaling in classical methods is inference, which typically scales at best polynomially in the size of the model. In order to make classical methods performant while maintaining interpretability, we present techniques for circumventing the heavy computational costs of inference. We first focus on scaling one of the simplest models, hidden Markov models, then show our techniques generalize to more complex models, such as probabilistic context-free grammars.

## 1 Introduction

Neural network-based generative models have led to progress in difficult tasks such as language modeling and machine translation. However, this progress comes at the cost of interpretability. Neural networks are flexible function approximations, but that flexibility results in opaque models that are difficult to analyze. Rather than post-hoc analysis [8], we instead propose to explore the space of models that are defined with interpretability in mind. In particular, we focus on probabilistic graphical models, which allow for the execution of arbitrary probabilistic queries, i.e. the calculation of (functions of) probability distributions via operations such as conditioning and marginalization [3] (simplify). We seek to answer the question of whether we can have models that are both performant and interpretable.

We use language modeling as a benchmark task, as the complex and sometimes long-range phenomena in natural language provides a good testbed. Additionally, language modeling captures import scientific and linguistic questions (more on this later). Language model benchmarks are currently dominated by autoregressive neural models, which makes it difficult to analyze what properties of language the models are actually capturing. For example, it is difficult to analyze the inner representations used in syntax-inspired neural language models [1] (this is my own take on that paper, need to find a better citation). This is an issue if the goal is to learn from models in order to answer scientific questions.

In this work, we investigate scaling probabilistic graphical models which explicitly reason about latent variables. We focus on the one of the simplest latent variable models, the hidden Markov model (HMM). HMMs posit a simple generative process, that first generates a sequence of latent states then the emissions. Additionally, HMMs make very strong conditional independence assumptions. Despite the simplicity and constraints of HMMs, they are still computationally expensive to scale due to the cost of marginalization.

In order to scale HMMs, we propose a series of choices in parameterization that result in computational speedups for inference in HMMs: sparse emission constraint, state dropout, and softmax kernel feature map approximation.

Before diving into these contributions, we first review HMMs.

## 2 Hidden Markov Models for Language Modeling

Hidden Markov models (HMMs) have a rich history in natural language processing. They have been used for a series of tasks, including speech recognition [5], part of speech tagging [4], and word alignment in machine translation [9]. They have also been used as components in neural models, such as in attention [7]. We focus on applying HMMs to the task of language modeling, where the goal is to model the tokens in a sentence  $x = (x_1, \dots, x_T)$ .

HMMs are one of the simplest latent variable models for language modeling with a per-token latent variable. The simplicity is a result of the very strong conditional independence assumption: Every timestep has a single discrete state latent variable for representing context, and HMMs emit each token given only the corresponding latent state. Additionally, the next state is a function of only the previous state. These independence assumptions result in a bottleneck where all information must flow through the single state variable, causing performance to be limited by the number of states while also giving the model interpretability by limiting dependencies.

Formally, HMMs have the following generative process: for each  $t \in [T]$  timestep, first choose a latent state  $z_t \in \mathcal{Z}$ , where  $|\mathcal{Z}|$  is the number of latent states, then choose a token to emit  $x_t \in \mathcal{X}$ . This defines the joint distribution:

$$p(x, z) = \prod_t p(x_t | z_t) p(z_t | z_{t-1}), \quad (1)$$

with transitions  $p(z_t | z_{t-1})$ , emissions  $p(x_t | z_t)$ , and a distinguished start state  $z_0 = S$ .

Training an HMM requires marginalizing over the unobserved  $z = (z_1, \dots, z_T)$  in order to obtain the evidence  $p(x) = \sum_z p(x, z)$ , accomplished via the forward algorithm. The forward algorithm can be written as a sequence of matrix-vector multiplications: Let the transition operators  $\Lambda_t \in \mathbb{R}^{|\mathcal{Z}| \times |\mathcal{Z}|}$  for  $t \in [2, \dots, T]$ , with entries

$$[\Lambda_t]_{z_t, z_{t-1}} = p(x_t | z_t) p(z_t | z_{t-1}),$$

with the first operator given by

$$[\Lambda_1]_{z_1, z_0} = \begin{cases} p(x_1 | z_1) p(z_1 | z_0 = S) & z_0 = S \\ 0 & \text{otherwise.} \end{cases}$$

The evidence is then given by

$$p(x) = \mathbf{1}^\top \Lambda_1 \Lambda_2 \cdots \Lambda_T \mathbf{1}, \quad (2)$$

where  $\mathbf{1} \in \mathbb{R}^{|\mathcal{Z}| \times 1}$  is the column vector of all ones.

On a serial machine, the cost of computing each matrix-vector product in the above equation takes time  $O(|\mathcal{Z}|^2)$ , resulting in a total running time of  $O(T|\mathcal{Z}|^2)$  for the forward algorithm. The quadratic dependence on the number of states precludes scaling to extremely large state spaces.

gradient  
computa-  
tion requires  
 $O(T|\mathcal{Z}|^2)$   
space

### 3 Kernel Parameterization

Recent work in latent variable modeling with neural components has shown that the parameterization of conditional distributions can have a large effect on generalization [2]. In particular, one is able to encode different inductive biases through choices in parameterization.

A common choice when parameterizing conditional distributions with neural nets is the use of softmax, which has origins in smooth approximations of argmax as well as the conditional max entropy problem (citations needed). In a conditional distribution  $p(x | z)$ , the softmax parameterization is as follows:

$$p(x | z) = \frac{\exp(\mathbf{u}_z^\top \mathbf{v}_x)}{\sum_{x'} \exp(\mathbf{u}_z^\top \mathbf{v}_{x'})},$$

with  $\mathbf{u}_z, \mathbf{v}_x \in \mathbb{R}^d$  vector-space embeddings of values of  $x, z$ . The numerator,  $\exp(\mathbf{u}_z^\top \mathbf{v}_x)$ , is an instance of a kernel, referred to as the exponential kernel [6].

Kernel methods have a rich history in machine learning, having been applied to nonparametric regression, support vector machines, gaussian processes, among other models, to a large degree of success. In those settings, kernels are intuitively used to measure the similarity between two data points. In our setting, kernels parameterize conditional distributions, measuring the propensity of one random variable taking a particular value given the value of another.

Generalizing from the exponential kernel, a kernel is a function  $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ . Although applications that rely on the kernel trick, such as support vector machines, only require positive symmetric definite kernels, our use of kernels to define probability mass functions additionally requires kernels to be nonnegative, with  $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}_+$ . We can then replace the exponential kernel in softmax with an arbitrary nonnegative kernel:

$$p(x | z) = \frac{K(\mathbf{u}_z, \mathbf{v}_x)}{\sum_{x'} K(\mathbf{u}_z, \mathbf{v}_{x'})},$$

where we again have embeddings  $\mathbf{u}_z, \mathbf{v}_x \in \mathbb{R}^d$ . A nonnegative kernel must have a positive semidefinite kernel matrix, which implies that

Consider the emission distribution  $p(x \mid z; \theta)$ , with discrete random variables  $x, z$  and model parameters  $\theta$ .<sup>12</sup> The full rank parameterization defines the model parameters  $\theta \in \mathbb{R}^{|\mathcal{Z}| \times |\mathcal{X}|}$ , giving a one-to-one correspondence between the distribution parameters and model parameters:

$$p(x = j \mid z = i) = [\theta]_{i,j}.$$

A kernel parameterization instead defines model parameters  $\theta = \{U, V\}$ , with  $U \in \mathbb{R}^{|\mathcal{Z}| \times d}$ ,  $V \in \mathbb{R}^{|\mathcal{X}| \times d}$ ,<sup>3</sup> and obtains the distributional parameters via

$$p(x = j \mid z = i) = \frac{K(u_i, v_j)}{\sum_{j'} K(u_i, v_{j'})},$$

where  $u_i$  and  $v_j$  are the  $i$ th and  $j$ th rows of  $U$  and  $V$ , and  $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  is a kernel. There are a variety of valid kernels  $K$  one could employ; however, we focus on the exponential kernel, given by  $K(u, v) = \exp(u^\top v)$ , as it is widely used in practice as part of the softmax function.

(turn into footnote, or experimental detail) Finally, a neural parameterization takes a further step in abstraction by parameterizing the embedding matrices  $U, V$  as functions of neural networks, allowing for more parameter sharing. For the embedding matrix  $U \in \mathbb{R}^{|\mathcal{Z}| \times d}$ , this entails having another underlying embedding matrix  $U' \in \mathbb{R}^{|\mathcal{Z}| \times d'}$  generating the rows  $u_i = f_U(u'_i)$ , where the same  $f_U$  is used for all  $i \in \mathcal{Z}$ . In this work, we use the following parameterization for  $f$ : (resnet thing).

Pros and cons of each, implications? Probably make this a subsection too.

Add summarizing paragraph about the different levels of abstraction, with a picture. Talk about going up and down the stack? This way when we talk about each technique we can point to exactly what abstraction it operates in.

## 4 Kernels

In the previous section, we discussed three types of parameterizations: full rank, kernel, and neural. Both the kernel and neural parameterizations hinged on the use of a kernel to model the interaction between the values of random variables. In this section, we will discuss some background on kernels, as well as their use in obtaining an efficient inference algorithm for HMMs.<sup>4</sup>

A kernel is positive definite symmetric (PDS) if .

PDS kernel to RKHS, reproducing property?

### 4.1 Exponential Kernels and Softmax

Kernels, and in particular exponential kernels, are widely used to parameterize conditional distributions in neural networks.

Why is softmax / exponential kernel special? Maximum entropy distributions, exponential family. Slides on information projections Power series representation?

## 5 Kernel Approximations

Taylor approximation, generating functions  
limitations?

<sup>1</sup> The dimension of the model parameters  $\theta$  will vary depending on the choice of parameterization.

<sup>2</sup> The following parameterizations extend to the transition distribution as well, and any conditional distribution.

<sup>3</sup> Typically  $U$  and  $V$  are referred to as embedding matrices, as the embeddings of the elements of  $\mathcal{Z}$  and  $\mathcal{X}$  in  $\mathbb{R}^d$  are given by the rows of  $U$  and  $V$ .

<sup>4</sup> As we will see, the inference procedure extends to more complex models as well.

## 5.1 Bochner Theorem

## 5.2 Nystrom

# 6 Kernelized Inference

# 7 Generalization: Kernelized Belief Propagation

# 8 Spectral Methods?

## References

- [1] Wenyu Du, Zhouhan Lin, Yikang Shen, Timothy J. O’Donnell, Yoshua Bengio, and Yue Zhang. Exploiting syntactic structure for better language modeling: A syntactic distance approach, 2020.
- [2] Yoon Kim, Chris Dyer, and Alexander Rush. Compound probabilistic context-free grammars for grammar induction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2369–2385, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1228. URL <https://www.aclweb.org/anthology/P19-1228>.
- [3] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. Adaptive computation and machine learning. MIT Press, 2009. ISBN 9780262013192. URL <https://books.google.co.in/books?id=7dzpHCHzNQ4C>.
- [4] Bernard Merialdo. Tagging English text with a probabilistic model. *Computational Linguistics*, 20(2):155–171, 1994. URL <https://www.aclweb.org/anthology/J94-2001>.
- [5] Lawrence R. Rabiner. *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, page 267–296. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990. ISBN 1558601244.
- [6] Ankit Singh Rawat, Jiecao Chen, Felix Xinnan X Yu, Ananda Theertha Suresh, and Sanjiv Kumar. Sampled softmax with random fourier features. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 13857–13867. Curran Associates, Inc., 2019. URL <https://proceedings.neurips.cc/paper/2019/file/e43739bba7cdb577e9e3e4e42447f5a5-Paper.pdf>.
- [7] Shiv Shankar and Sunita Sarawagi. Posterior attention models for sequence to sequence learning. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=BkltNhC9FX>.
- [8] Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. In *Association for Computational Linguistics*, 2019. URL <https://arxiv.org/abs/1905.05950>.
- [9] Stephan Vogel, Hermann Ney, and Christoph Tillmann. Hmm-based word alignment in statistical translation. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 2, COLING ’96*, page 836–841, USA, 1996. Association for Computational Linguistics. doi: 10.3115/993268.993313. URL <https://doi.org/10.3115/993268.993313>.

## A Gradient Estimator Implementation

In the log-semiring, addition is given by  $\oplus = \text{L}\Sigma\text{E}$  and multiplication by  $\otimes = +$ . Consider the linear chain CRF,  $\otimes_t \psi(x_{t-1}, x_t)$ , with  $\psi(x_{t-1}, t) = f(x_t) \oplus g(x_{t-1}, x_t)$ . We would like to compute the gradient of the log partition function,  $A = \otimes_x \otimes_t \psi(x_{t-1}, x_t)$ . Recall the gradient identities

$$\begin{aligned}\nabla_a a \oplus b &= \frac{\exp(a)}{\exp(a \oplus b)} \\ \nabla_a a \otimes b &= 1.\end{aligned}\tag{3}$$

We then have

$$\begin{aligned}\nabla_{\psi(x_a, x_b)} \otimes_t \psi(x_{t-1}, x_t) \\ = \nabla_{\psi(x_a, x_b)} \otimes_t \psi(x_{t-1}, x_t)\end{aligned}\tag{4}$$