

# Scaling Hidden Markov Language Models

Justin T. Chiu and Alexander M. Rush

Department of Computer Science

Cornell Tech

{jtc257, arush}@cornell.edu

## Abstract

The hidden Markov model (HMM) is a fundamental tool for sequence modeling that cleanly separates the hidden state from the emission structure. However, this separation makes it difficult to fit HMMs to large datasets in modern NLP, and they have fallen out of use due to very poor performance compared to fully observed models. This work revisits the challenge of scaling HMMs to language modeling datasets, taking ideas from recent approaches to neural modeling. We propose methods for scaling HMMs to massive state spaces while maintaining efficient exact inference, a compact parameterization, and effective regularization. Experiments show that this approach leads to models that are much more accurate than previous HMMs and n-gram-based methods, making progress towards the performance of state-of-the-art NN models.

## 1 Introduction

Hidden Markov models (HMMs) are a fundamental latent-variable model for sequential data, with a rich history in NLP. They have been used extensively in NLP tasks such as tagging (Merialdo, 1994), alignment (Vogel et al., 1996), and even, in a few cases, language modeling (Kuhn et al., 1994; Huang, 2011). Compared to other sequence models, HMMs are naturally appealing since they fully separate the process of generating hidden states from observations, while allowing for exact posterior inference.

State-of-the-art systems in NLP have moved away from utilizing latent hidden states and toward deterministic deep neural models. We take several lessons from the success of neural models for NLP tasks: (a) model size is critical for finding better local optima, e.g. large LSTMs (Zaremba et al., 2014) show marked improvements in performance;

(b) the right factorization is critically important for representation learning, e.g. a feedforward model (Bengio et al., 2003) can have the same probabilistic structure as an n-gram model while performing significantly better; (c) dropout is key to achieving strong performance (Zaremba et al., 2014; Merity et al., 2017).

We revisit HMMs for language modeling, positing that competitive performance may require very large models. Our contributions are as follows: We demonstrate large improvements in an HMM language model by scaling the state space. Towards that goal, we introduce three techniques: a modeling constraint that allows us to use a large number of states while maintaining efficient exact inference, a neural parameterization that improves generalization while remaining faithful to the probabilistic structure of the HMM, and a variant of dropout that both improves accuracy and halves the computational overhead during training.

Experiments employ HMMs on two language modeling datasets. Our three techniques allow us to train an HMM with tens of thousands of states, significantly outperforming past HMMs as well as n-gram models.

## 2 Related Work

In order to improve the performance of HMMs on language modeling, several recent papers have combined HMMs with neural networks. Buys et al. (2018) develop an approach to relax HMMs, but their models either perform poorly or alters the probabilistic structure to resemble an RNN. Krakovna and Doshi-Velez (2016) utilize model combination with an RNN to connect both approaches in a 20 state model. We instead improve pure HMMs by scaling to many states.

Prior work has considered neural parameteriza-

tions of HMMs. Tran et al. (2016) demonstrate improvements in POS induction with a neural parameterization of an HMM. They consider small state spaces, as the goal was tag induction rather than language modeling.<sup>1</sup>

Prior work has also used HMMs with many states. Dedieu et al. (2019) introduce a sparsity constraint in order to train a 30K state non-neural HMM for character-level language modeling; however, their constraint precludes application to large vocabularies. We overcome this limitation and train models with neural parameterizations on word-level language modeling.

Finally, another approach for scaling state spaces is to grow from small to big via a split-merge process (Petrov et al., 2006; Huang, 2011). In particular, Huang (2011) learn an HMM for language modeling via this process. As fixed-size state spaces are amenable to batching GPU, we leave split-merge procedures for future work.

### 3 Background: HMMs

We are interested in learning a distribution over observed tokens  $\mathbf{x} = \langle x_1, \dots, x_T \rangle$ , with each token  $x_t$  an element of the finite vocabulary  $\mathcal{X}$ . Hidden Markov models (HMMs) specify a joint distribution over observed tokens  $\mathbf{x}$  and discrete latent states  $\mathbf{z} = \langle z_1, \dots, z_T \rangle$ , with each  $z_t$  from the finite set  $\mathcal{Z}$ . For notational convenience, we define the starting state  $z_0 = \epsilon$ . This yields the joint distribution

$$p(\mathbf{x}, \mathbf{z}; \theta) = \prod_{t=1}^T p(x_t | z_t) p(z_t | z_{t-1}) \quad (1)$$

The distributions are parameterized as follows

$$p(z_t | z_{t-1}) \propto e^{\psi_{z_{t-1}z_t}} \quad p(x_t | z_t) \propto e^{\phi_{z_t x_t}} \quad (2)$$

with transitions  $\psi \in \mathbb{R}^{|\mathcal{Z}| \times |\mathcal{Z}|}$  and emissions  $\phi \in \mathbb{R}^{|\mathcal{Z}| \times |\mathcal{X}|}$ . These are the distributional parameters of an HMM. We refer to emissions  $p(x_t | z_t)$  as  $\mathbf{O}$ .

We distinguish between two types of model parameterizations: *scalar* and *neural*, where the model parameters are given by  $\theta$ . A scalar parameterization sets the model parameters equal to the distributional parameters, so that  $\theta = \{\phi, \psi\}$ , resulting in  $O(|\mathcal{Z}|^2 + |\mathcal{Z}||\mathcal{X}|)$  model parameters. A

neural parameterization instead generates the distributional parameters from a neural network (with parameters  $\theta$ ), decoupling the size of  $\theta$  from  $\phi, \psi$ . This decoupling gives us the ability to choose between compact or overparameterized  $\theta$  (relative to  $\phi, \psi$ ). As we will use large state spaces, we take advantage of compact neural parameterizations.

In order to fit an HMM to data  $\mathbf{x}$ , we must marginalize over the latent states to obtain the likelihood  $p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z})$ . This sum can be computed in time  $O(T|\mathcal{Z}|^2)$  via dynamic programming, which becomes prohibitive if the number of latent states  $|\mathcal{Z}|$  is large. We can then optimize the likelihood with gradient ascent (or alternative variants of expectation maximization). See the appendix for more details on inference and optimization in HMMs.

**HMMs and RNNs** Although the forward algorithm resembles that of the forward pass in a recurrent neural network (RNN) (Buys et al., 2018), there are key representational differences. RNNs do not decouple the latent dynamics from the observed. This often leads to improved accuracy, but precludes posterior inference which is useful for interpretability. A further benefit of HMMs over RNNs is that their associative structure allows for parallel inference via the prefix-sum algorithm (Ladner and Fischer, 1980).<sup>2</sup> Finally, HMMs bottleneck information from every timestep through a discrete hidden state. NLP has a long history of utilizing discrete representations, and returning to explore discrete representations may yield interesting results. For example, recent work has found that discrete latent variables work well in low-resource regimes (Jin et al., 2020).

### 4 Scaling HMMs

**Blocked Emissions** Efficiency of marginal inference inherently limits the state space of general HMMs. However, we can improve inference complexity in special cases. As states in an HMM are used to represent context, a reasonable assumption is that not every word should be used in every context. Inspired by cloned HMMs (Dedieu et al., 2019), we constrain our HMMs to have rectangular

<sup>1</sup> Other work has used neural parameterization for structured models, such as dependency models (Han et al., 2017), hidden semi-Markov models (Wiseman et al., 2018), and context free grammars (Kim et al., 2019).

<sup>2</sup> Quasi-RNNs (Bradbury et al., 2016) also have a logarithmic dependency on  $T$  by applying the same prefix-sum trick, but do not model uncertainty over latent dynamics.



Figure 1: The emission matrix as a set of blocks  $\mathbf{O}_1, \dots, \mathbf{O}_4$  with fixed height  $k$ . The width of each block may vary, as there is no constraint on the number of words a state can emit. Each active cell is constructed from word  $\mathbf{E}_x$  and state  $\mathbf{E}_z$  embeddings.

blocked emissions,

$$\mathbf{O} = \begin{bmatrix} \mathbf{O}_1 & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & \mathbf{O}_M \end{bmatrix}$$

where each  $\mathbf{O}_m \in \mathbb{R}^{k \times |\mathcal{X}_m|}$  ( $k = |\mathcal{Z}|/M$ ) is a partition indicating which tokens  $\mathcal{X}_m$  can be emitted by states  $mk$  through  $(m+1)k$ . Conversely, let  $\mathcal{Z}_x \subset \mathcal{Z}$  be the states with non-zero probability of emitting  $x$  (with  $|\mathcal{Z}_x| = k$ ). Exact marginalization can be computed via

$$p(\mathbf{x}) = \sum_{z_1 \in \mathcal{Z}_{x_1}} p(z_1 | z_0) p(x_1 | z_1) \times \dots \sum_{z_T \in \mathcal{Z}_{x_T}} p(z_T | z_{T-1}) p(x_T | z_T) \quad (3)$$

Since there are only  $k$  states with nonzero probability of occurring at every timestep, we only need to consider transitioning from the  $|\mathcal{Z}_{x_t}| = k$  previous states to the next  $|\mathcal{Z}_{x_{t+1}}| = k$  states, resulting in  $O(k^2)$  operations per timestep. This gives a serial complexity of  $O(Tk^2)$ .<sup>3</sup>

**Neural Parameterization** Even with blocked emissions, the scalar parameterization of an HMM grows as  $O(|\mathcal{Z}|^2)$ . We instead employ a neural parameterization. The approach is to embed each state in  $\mathcal{Z}$  ( $\mathbf{E}_z \in \mathbb{R}^{|\mathcal{Z}| \times h}$ ) and each token in  $\mathcal{X}$  ( $\mathbf{E}_x \in \mathbb{R}^{|\mathcal{X}| \times h}$ ). From these we can create representations for leaving and entering a state, and emitting a word:

$$\mathbf{H}_{\text{out}}, \mathbf{H}_{\text{in}}, \mathbf{H}_{\text{emit}} = \text{MLP}(\mathbf{E}_z)$$

<sup>3</sup> This can be sped up on a parallel machine to  $O(\log(T)k^2)$  via a binary reduction.

---

**Algorithm 1** HMM Training (a single batch)

---

Given: block structure and model parameters

Sample block-wise dropout mask  $\mathbf{b}$

Compute  $\phi, \psi$  ignoring  $b_z = 0$

**for all** examples  $\mathbf{x}$  in batch **do**

$\Phi = \text{LOGPOTENTIALS}(\phi, \psi, \mathbf{x}, \mathbf{b})$

$\log p(\mathbf{x}) = \text{FORWARD}(\Phi)$

Update embeddings  $\mathbf{E}_z, \mathbf{E}_x$  and MLP

---

with all  $\mathbf{H}_\bullet \in \mathbb{R}^{|\mathcal{Z}| \times h}$ . The HMM distributional parameters are given by

$$\phi = \mathbf{H}_{\text{emit}} \mathbf{E}_x^\top \quad \psi = \mathbf{H}_{\text{in}} \mathbf{H}_{\text{out}}^\top \quad (4)$$

where  $\phi \in \mathbb{R}^{|\mathcal{Z}| \times |\mathcal{X}|}$  and  $\psi \in \mathbb{R}^{|\mathcal{Z}| \times |\mathcal{Z}|}$ . The MLP architecture follows (Kim et al., 2019), with details in the appendix. This neural parameterization takes  $O(h^2 + h|\mathcal{Z}| + h|\mathcal{X}|)$  parameters (shown in Figure 1).

Note that parameter computation is independent of inference and can be cached completely at test-time. For training, we compute them once per batch (shown in Alg 1), while RNNs and similar models recompute emissions every token.

**Dropout as State Reduction** To encourage full use of the state space, we introduce dropout that prevents the model from favoring specific states. We propose a form of HMM state dropout that removes states from use entirely at each batch, which has the added benefit of speeding up inference.

State dropout acts on each emission block  $\mathbf{O}_1, \dots, \mathbf{O}_M$  independently. For each block (with  $k = |\mathcal{Z}|/M$  rows), we sample a binary dropout mask by sampling  $\lambda k$  dropped row indices uniformly without replacement, where  $\lambda$  is the dropout rate. We concatenate these to a global vector  $\mathbf{b} \in \{0, 1\}^{|\mathcal{Z}|}$ , which, along with the previous constraints, ensures,

$$p(x | z) \propto b_z 1(z \in \mathcal{Z}_x) e^{\phi_{zx}} \quad (5)$$

State dropout gives a large practical speed up for both parameter computation and inference. For  $\lambda = 0.5$  we get a  $4\times$  speed improvement for both, due to the reduction in possible transitions. This structured dropout is also easy to exploit on GPU, as it maintains block structure with fixed-height (as shown in Figure 2).

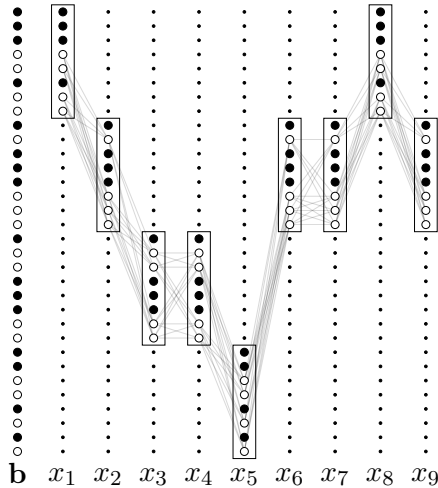


Figure 2: The computation of  $p(\mathbf{x})$  is greatly reduced by blocked emissions and state dropout. In the above trellis, each row corresponds to a latent state. Blocked emissions result in a small subset of all states emitting a given word, as shown by the rectangles, thus only states in the rectangles may occur with nonzero probability after conditioning on observed words. State dropout (leftmost column) allows us to further reduce the number of states we consider, halving the number of (white) states that have nonzero probability in each rectangle.

## 5 Experimental Setup

**Emission Blocks** The model requires partitioning token types into blocks  $\mathcal{X}_m$ . While there are many partitioning methods, a natural choice is Brown clusters (Brown et al., 1992; Liang, 2005) which are also based on HMMs. Brown clusters are obtained by assigning every token type in  $\mathcal{X}$  a state in an HMM, then merging states until a desired number of partitions  $M$  is reached. We construct the Brown clusters on the training portions of the datasets and assume the vocabulary remains identical at test time (OOV words mapped to unk). We include more background on Brown Clusters in the appendix.

**State Dropout** We use a dropout rate of  $\lambda = 0.5$  at training time. For each block of size  $|\mathcal{X}_m|$ , we sample  $\lambda|\mathcal{X}_m|$  states to use in that block each batch. We draw states from each block from a multivariate hypergeometric distribution using the Gumbel Top-k trick for sampling without replacement (Vieira, 2014). At test time we use the full state distribution without dropout.

**Datasets** We evaluate on the Penn Treebank (Marcus et al., 1993) (929k train tokens, 10k vocab) and Wikitext2 (Merity et al., 2016) (2M train tokens, 33k vocab) datasets. For Penn Treebank

Model	Param	Val	Test
Penn Treebank			
KN 5-gram	2M	-	141.2
AWD-LSTM	24M	60.0	57.3
256 FF 5-gram	2.9M	159.9	152.0
2x256 dim LSTM	3.6M	93.6	88.8
HMM+RNN	10M	142.3	-
HMM $ \mathcal{Z}  = 900$	10M	284.6	-
VLNHMM $ \mathcal{Z}  = 2^{15}$	11.4M	125.0	116.0
Wikitext			
KN 5-gram	5.7M	248.7	234.3
AWD-LSTM	33M	68.6	65.8
256 FF 5-gram	8.8M	210.9	195.0
2x256 LSTM	9.6M	124.5	117.5
VLNHMM $ \mathcal{Z}  = 2^{15}$	17.3M	166.6	158.2

Table 1: Perplexities on PTB / Wikitext-2.

we use the preprocessing from Mikolov et al. (2011), which lowercases all words and substitutes OOV words with unks. We insert EOS tokens after each sentence. For Wikitext2 casing is preserved, and all OOV words are unked. We insert EOS tokens after each paragraph. In both datasets OOV words were included in the perplexity (as unks), and EOS was included in the perplexity as well (Merity et al., 2017).

**Baselines** Baselines include AWD-LSTM (Merity et al., 2017); a 900-state scalar HMM and HMM+RNN extension, which discards the HMM assumptions (Buys et al., 2018); a KN 5-gram model (Mikolov and Zweig, 2012; Heafield et al., 2013), a 256 dimension FF model, and a 2-layer 256 dimension LSTM. We compare these with our very large neural HMM (VLNHMM) which has  $|\mathcal{Z}| = 2^{15}$  total states but only considers 256 states at every timestep at test time.<sup>4</sup> See the appendix for all hyperparameters.

## 6 Results

Table 1 gives the main results. On PTB, the VLNHMM is able to achieve 125.0 perplexity on the valid set, outperforming a FF baseline (159.9) and vastly outperforming the 900-state HMM from

<sup>4</sup> The 256 dim FF, LSTM, and VLNHMM in particular have comparable computational complexity:  $O(256^2T)$ .



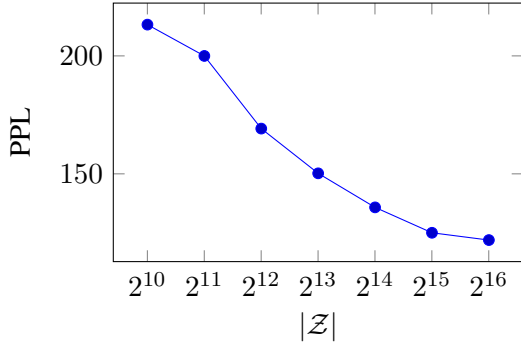


Figure 3: Perplexity on PTB by state size  $|Z|$  ( $\lambda = 0.5$  and  $M = 128$ ).

Model	Param	Train	Val	Time
VLNHMM ( $2^{14}$ )	7.2M	115	134	40
- neural param	423M	119	169	14
- dropout	7.2M	88	157	100

Table 2: Ablations on PTB ( $\lambda = 0.5$  and  $M = 128$ ). Time is ms per eval batch (Run on RTX 2080).

Buy et al. (2018) (284.6).<sup>5</sup> The VLNHMM also outperforms the HMM+RNN extension of Buy et al. (2018) (142.3). These results indicate that HMMs are a much stronger model on this benchmark than previously claimed. However, the VLNHMM is outperformed by LSTMs. This trend persists in Wikitext-2, with the VLNHMM outperforming the FF model but underperforming an LSTM.

Fig. 3 examines the effect of state size: We find that performance continuously improves significantly as we grow to  $2^{16}$  states.

Table 2 considers other ablations: Although neural and scalar parameterizations reach similar training perplexity, the neural model generalizes better on validation with almost 100x fewer parameters. We find that state dropout results in both an improvement in perplexity and a large improvement in computational speed. See the appendix for emission sparsity constraint ablations, as well as experiments on further reducing the number of parameters.

## 7 Conclusion

This work demonstrates that scaling HMMs to large states spaces results in performance gains.

<sup>5</sup> Buy et al. (2018) only report validation perplexity for the HMM and HMM+RNN models, so we compare accordingly.

In order to scale, we introduce three techniques: a blocked emission constraint, a neural parameterization, and state dropout, which lead to an HMM that outperforms n-gram models and prior HMMs. Once scaled up to take advantage of modern hardware, very large HMMs demonstrate meaningful improvements over smaller HMMs and are worthy of consideration for NLP tasks.

## Acknowledgements

We are grateful to Yuntian Deng, Yoon Kim, and Sam Wiseman for insightful conversations and suggestions.

## References

- Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null):1137–1155.
- James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2016. [Quasi-recurrent neural networks](#). *CoRR*, abs/1611.01576.
- Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992. Class-based n-gram models of natural language. *Comput. Linguist.*, 18(4):467–479.
- Jan Buys, Yonatan Bisk, and Yejin Choi. 2018. Bridging hmms and rnns through architectural transformations.
- Tianqi Chen, Thierry Moreau, Ziheng Jiang, Haichen Shen, Eddie Q. Yan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. [TVM: end-to-end optimization stack for deep learning](#). *CoRR*, abs/1802.04799.
- Antoine Dedieu, Nishad Gothoskar, Scott Swingle, Wolfgang Lehrach, Miguel Lázaro-Gredilla, and Dileep George. 2019. [Learning higher-order sequential structure with cloned hmms](#).
- Wenjuan Han, Yong Jiang, and Kewei Tu. 2017. [Dependency grammar induction with neural lexicalization and big training data](#). In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1683–1688, Copenhagen, Denmark. Association for Computational Linguistics.
- Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. 2013. [Scalable modified Kneser-Ney language model estimation](#). In *Proceedings of the 51st Annual Meeting of the Association*

- for *Computational Linguistics (Volume 2: Short Papers)*, pages 690–696, Sofia, Bulgaria. Association for Computational Linguistics.
- Zhongqiang Huang. 2011. *Modeling Dependencies in Natural Languages with Latent Variables*. Ph.D. thesis, University of Maryland.
- Shuning Jin, Sam Wiseman, Karl Stratos, and Karen Livescu. 2020. *Discrete latent variable representations for low-resource text classification*. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4831–4842, Online. Association for Computational Linguistics.
- Yoon Kim, Chris Dyer, and Alexander M. Rush. 2019. *Compound probabilistic context-free grammars for grammar induction*. *CoRR*, abs/1906.10225.
- Viktoriya Krakovna and Finale Doshi-Velez. 2016. *Increasing the interpretability of recurrent neural networks using hidden markov models*.
- Thomas Kuhn, Heinrich Niemann, and Ernst Günter Schukat-Talamazzini. 1994. *Ergodic hidden markov models and polygrams for language modeling*, pages 357–360.
- Richard E. Ladner and Michael J. Fischer. 1980. *Parallel prefix computation*. *J. ACM*, 27(4):831–838.
- Percy Liang. 2005. *Semi-supervised learning for natural language*. In *MASTER’S THESIS, MIT*.
- Ilya Loshchilov and Frank Hutter. 2017. *Fixing weight decay regularization in adam*. *CoRR*, abs/1711.05101.
- Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. *Building a large annotated corpus of English: The Penn Treebank*. *Computational Linguistics*, 19(2):313–330.
- Bernard Merialdo. 1994. *Tagging English text with a probabilistic model*. *Computational Linguistics*, 20(2):155–171.
- Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. *Regularizing and optimizing LSTM language models*. *CoRR*, abs/1708.02182.
- Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. *Pointer sentinel mixture models*. *CoRR*, abs/1609.07843.
- T. Mikolov and G. Zweig. 2012. *Context dependent recurrent neural network language model*. In *2012 IEEE Spoken Language Technology Workshop (SLT)*, pages 234–239.
- Tomas Mikolov, Anoop Deoras, Stefan Kombrink, Lukás Burget, and Jan Cernocký. 2011. *Empirical evaluation and combination of advanced language modeling techniques*, pages 605–608.
- Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. *Learning accurate, compact, and interpretable tree annotation*, page 433–440.
- Ke M. Tran, Yonatan Bisk, Ashish Vaswani, Daniel Marcu, and Kevin Knight. 2016. *Unsupervised neural hidden markov models*. *CoRR*, abs/1609.09007.
- Tim Vieira. 2014. *Gumbel-max trick and weighted reservoir sampling*.
- Stephan Vogel, Hermann Ney, and Christoph Tillmann. 1996. *Hmm-based word alignment in statistical translation*. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 2, COLING ’96*, page 836–841, USA. Association for Computational Linguistics.
- Sam Wiseman, Stuart M. Shieber, and Alexander M. Rush. 2018. *Learning neural templates for text generation*. *CoRR*, abs/1808.10122.
- Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. *Recurrent neural network regularization*. *CoRR*, abs/1409.2329.

## A Appendices

### A.1 Inference and Learning in HMMs

In this section we review the computation of the evidence of observed words  $p(\mathbf{x})$  in an HMM. Recall that the evidence is given by

$$\begin{aligned}
 p(\mathbf{x}) &= \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) \\
 &= \sum_{\mathbf{z}} \prod_{t=1}^T p(z_t | z_{t-1}) p(x_t | z_t) \\
 &= \sum_{z_1} \cdots \sum_{z_T} p(z_1 | z_0) p(x_1 | z_1) \\
 &\quad \cdots p(z_T | z_{T-1}) p(x_T | z_T)
 \end{aligned} \tag{6}$$

where  $p(z_0 = \epsilon) = 1$  is a dummy state for ease of notation. Now consider matrices  $\mathbf{F}_t$  with elements  $F_{ij} = p(z_t = j | z_{t-1} = i) p(x_t | z_t = j)$ . Note that  $\mathbf{f}_1 \in \mathbb{R}^{|\mathcal{Z}|}$  since there is only a single starting state  $z_0 = \epsilon$ , while  $\mathbf{F}_t \in \mathbb{R}^{|\mathcal{Z}| \times |\mathcal{Z}|}$  for  $t \in [2, \dots, T]$ . Repeated application of the distributive rule allows us to express the above in matrix notation

$$p(\mathbf{x}) = \mathbf{f}_1^T \mathbf{F}_2 \cdots \mathbf{F}_{T-1} \mathbf{F}_T \mathbf{1} \tag{7}$$

where  $\mathbf{1}$  is a column vector with all entries equal to 1. In this form the runtime is explicit: there are  $O(T)$  matrix-vector products resulting

in  $O(T|\mathcal{Z}|^2)$  computation.<sup>6</sup> Additionally, computing gradients with respect to each  $\mathbf{F}_t$  can be done with any automatic differentiation library, as all operations in Eqn. 7 are differentiable.

The block-sparse emissions and state dropout serve to zero out entries of each  $\mathbf{F}_t$ , allowing us to only consider dense submatrices of each matrix  $\mathbf{F}_t$  in Eqn. 7.

## A.2 Brown Clustering

TODO: finish

## A.3 Hyperparameters

For Penn Treebank and Wikitext-2, we trained the following baselines: a two layer FF 256-dim 5-gram model and a two layer 256-dim LSTM. The FF model is given by the following:

$$p(w_t | \mathbf{w}_{<t}) = W_x \text{ReLU}(\text{Conv}(\mathbf{E}_w(\mathbf{w}_{t-4:t-1}))) \quad (8)$$

where  $\mathbf{E}_w$  gives the word embeddings and  $W_x \in \mathbb{R}^{|\mathcal{X}| \times h}$  is weight-tied to the embeddings.

For the (5-gram) FF model we use a batch size of 128 and a bptt length of 64, as we found the model needed a larger batch size to achieve decent performance. For the LSTM, we use a batch size of 16 and a BPTT length of 32. For both baseline models we use AdamW (Loshchilov and Hutter, 2017) with a learning rate of 1e-3 and a dropout rate of 0.3 on the activations in the model. Both models use a hidden dimension of 256 throughout. These same hyperparameters were applied on both Penn Treebank and Wikitext-2.

For the HMMs we use a batch size of 16 and a BPTT length of 32. We use state dropout with rate  $\lambda = 0.5$ . We use AdamW (Loshchilov and Hutter, 2017) with a learning rate of 1e-2 for Penn Treebank, and a learning rate of 1e-3 for Wikitext-2.

All weights are initialized with the Kaiming uniform initialization. The FF model was trained for 100 epochs, while all other models were trained for 50. Validation likelihood was checked 4 times per epoch, and learning rates were decayed by a factor of 4 if the validation performance did not improve after 8 checks.

<sup>6</sup> One can apply the associative property of matrix multiplication to perform Eqn. 7 as a binary tree reduction, resulting in  $O(\log(T)|\mathcal{Z}|^3)$  runtime on a parallel machine.

Hyperparameter search was performed manually, using the best validation perplexity achieved in a run. Bounds:

1. Learning rate  $\in \{0.0001, 0.001, 0.01, 0.1\}$
2. Dropout  $\lambda \in \{0, 0.25, 0.5, 0.75\}$
3. Hidden dimension  $h \in \{128, 256, 512\}$
4. Batch size  $\in \{16, 32, 64, 128\}$

Experiments were run on RTX 2080 GPUs.

On PTB the FF model takes 3s per epoch, the LSTM 23s, and the VLHMM 2<sup>15</sup> 433s. The inference for VLHMM was not heavily optimized, and uses a kernel produced by TVM (Chen et al., 2018) for computing gradients through marginal inference.

## A.4 HMM Parameterization

Let  $\mathbf{E}, \mathbf{D} \in \mathbb{R}^{v \times h}$  be an embedding matrix and a matrix of the same size, where  $v$  is the size of the vocab and  $h$  the hidden dimension. We use the following residual network as our MLP:

$$\begin{aligned} f_i(\mathbf{E}) &= g_i(\text{ReLU}(\mathbf{E}W_{i1})) \\ g_i(\mathbf{D}) &= \text{LayerNorm}(\text{ReLU}(\mathbf{D}W_{i2}) + \mathbf{D}) \end{aligned} \quad (9)$$

with  $i \in \{\text{out}, \text{in}, \text{emit}\}$ ,  $W_{i1}, W_{i2} \in \mathbb{R}^{h \times h}$ . The state embeddings are then obtained by

$$\begin{aligned} \mathbf{H}_{\text{out}} &= f_{\text{out}}(\mathbf{E}_z) \\ \mathbf{H}_{\text{in}} &= f_{\text{in}}(\mathbf{E}_z) \\ \mathbf{H}_{\text{emit}} &= f_{\text{emit}}(\mathbf{E}_m, \mathbf{E}_z) \end{aligned} \quad (10)$$

We additionally experiment with factored state embeddings. TODO: finish. We introduce residual networks  $f_j, j \in \{o, i, e\}$  to compose block and state embeddings, yielding

$$\begin{aligned} \mathbf{H}_{\text{out}} &= f_{\text{out}}(f_o([\mathbf{E}_m, \mathbf{E}_z])) \\ \mathbf{H}_{\text{in}} &= f_{\text{in}}(f_i([\mathbf{E}_m, \mathbf{E}_z])) \\ \mathbf{H}_{\text{emit}} &= f_{\text{emit}}(f_e([\mathbf{E}_m, \mathbf{E}_z])) \end{aligned} \quad (11)$$

## A.5 Emission Constraint Ablation

Tbl. 3 shows the results from emission constraint ablations. For the ablations in this section, we do not use the factored state embedding and instead directly learn embeddings  $\mathbf{E}_z \in \mathbb{R}^{|\mathcal{Z}| \times h}$ . We examine the effect of factored state embeddings in the next section.

Constraint	$ \mathcal{Z} $	$ \mathcal{C}_x $	$m$	Val PPL
Brown	16384	512	32	137
Brown	16384	256	64	138
Brown	16384	128	128	134
Brown	16384	64	256	136
None	1024	-	-	180
Brown	1024	256	4	182
Brown	1024	128	8	194
Uniform	8192	128	-	150
Brown	8192	128	64	142
Uniform	16384	128	-	146
Brown	16384	128	128	136

Table 3: Emission constraint ablations on Penn Treebank.

With a VLNHMM that has  $|\mathcal{Z}| = 2^{14}$  states, the model is insensitive to the number of blocks  $M$ . However, with fewer states  $|\mathcal{Z}| = 2^{10}$  where we are able to use fewer blocks to examine whether the block-sparsity of the emission results in a performance loss. With  $M = 4$  blocks, the block-sparse HMM matches an unconstrained HMM with the same number of states. When  $M = 8$ , the block-sparse model underperforms, implying there may be room for improvement with the larger HMMs that use  $M > 8$  blocks.

We additionally compare the blocks induced by Brown clustering with a uniform constraint that samples subsets of states of size  $n$  independently and uniformly from  $\mathcal{Z}$ . This does not admit a partitioning, which makes it difficult to apply state dropout. We therefore zero out half of the logits of the transition matrix randomly before normalization. In the bottom of Tbl. 3, we find that models with uniform constraints are consistently outperformed by models with Brown cluster constraints as measured by validation perplexity. The models with uniform constraints also had poor validation performance despite better training performance, a symptom of overfitting.

These ablations demonstrate that the constraints based on Brown clusters used in this work may not be optimal, motivating future work that learns sparsity structure.

## A.6 Factored State Representation Ablation

We examine the effect of factoring state representations into block embeddings and independent state embeddings. Recall that factored state representations were parameterized via

$$\mathbf{H}_{\text{out}}, \mathbf{H}_{\text{in}}, \mathbf{H}_{\text{emit}} = \text{MLP}(\mathbf{E}_m, \mathbf{E}_z)$$

with the block and state embeddings  $\mathbf{E}_m, \mathbf{E}_z \in \mathbb{R}^{|\mathcal{Z}| \times h/2}$ . We ablate this by comparing to the following independent state representation:

$$\mathbf{H}_{\text{out}}, \mathbf{H}_{\text{in}}, \mathbf{H}_{\text{emit}} = \mathbf{E}'_z$$

with  $\mathbf{E}'_z \in \mathbb{R}^{|\mathcal{Z}| \times h}$ . The results of the factored state ablation are in Fig. 4,

We find that the performance of independent state embeddings with is similar to a model with factored embeddings, until the number of blocks is  $\leq 64$ .

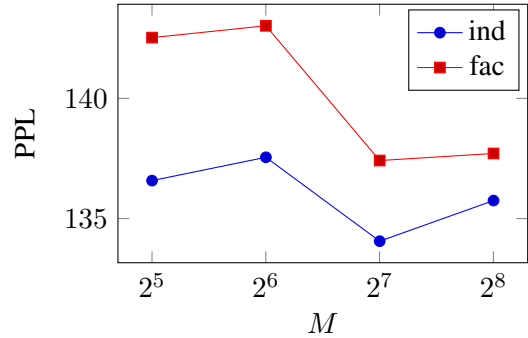


Figure 4: Perplexity on PTB by number of blocks  $M$  ( $\lambda = 0.5$  and  $|\mathcal{Z}| = 2^{14}$ ).

## A.7 Computational Considerations

Model	Size	Train	Val	Time
VLNHMM ( $2^{14}$ )	7.2M	115	134	40
- neural param	423M	119	169	14
- dropout	7.2M	88	157	100
+ block emb	5.6M	122	136	48

Table 4: Ablations on PTB ( $\lambda = 0.5$  and  $M = 128$ ). Time is ms per eval batch (Run on RTX 2080).

We reproduce the technique ablation table here in Tbl. 4 for reference. As we remove neural components, the number of parameters increases but the time of the forward pass decreases. This is because generating parameters from a neural network



takes strictly more time than having those parameters available, similar to a hierarchical Bayesian model.

When block embeddings are removed and the state representations are directly parameterized, the model is faster due to not needing to recompute state representations. This contrast is even more pronounced when removing neural components altogether, with an almost 3x speedup. However, we note that the drop in generalization is large, indicating the neural parameterization helps with learning. Therefore we use the neural parameterization at training time, then cache the resulting HMM parameters (the transition and emission matrices) for use during inference.