Scaling Hidden Markov Models

First Author
Affiliation / Address line 1
Affiliation / Address line 2
Affiliation / Address line 3
email@domain

Second Author
Affiliation / Address line 1
Affiliation / Address line 2
Affiliation / Address line 3
email@domain

Abstract

Hidden Markov models (HMMs) are a classic approach in NLP that explicitly separate the hidden state and generative structure. Unfortunately, this clean separation makes them difficult to fit 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 fitting HMMs to large datasets, taking ideas from recent approaches to neural modeling. We propose methods for scaling HMMs to massive state spaces, while maintaining compact parameterization, effective regularization, and efficient exact inference. Experiments show that this approach leads to models that are vastly better than previous HMM and ngram-based methods while nearing the performance of simple NN models.

1 Introduction

Hidden Markov models (HMMs) are a fundamental latent-variable model for sequential data. They are core to time-series problems in bioinformatics, reinforcement learning, and, of course, natural language processing. Historically they have been used extensively in NLP for tasks such as sequence modeling (Rabiner, 1990), alignment (Vogel et al., 1996), and less extensively language modeling (Kuhn et al., 1994; Huang, 2011). Compared to other approaches for sequence models, HMMs are naturally appealing since they fully separate out the process of sequential memory from the process of generation, while allowing for exact posterior inference.

In recent years, most state-of-the-art systems in NLP have moved away utilizing explicit hidden states, particularly for structured models. For instance in the area of language models, systems have moved to model observed words using n-gram models, feedforward neural networks (Bengio et al., 2003), recurrent neural networks (Mikolov et al., 2010; Zaremba et al., 2014; Merity et al., 2017) or transformers (Radford et al., 2019). This progress has led to the common wisdom that latent-variable models like HMMs are not competitive with fully observed models.

We take several lessons from the success of deep neural models for NLP tasks: (a) 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; (b) overparameterization is critical for finding better local optima, e.g. overly large LSTMs Zaremba et al. (2014) show marked improvements in performance; (c) regularization choices are necessary to find good solutions for different model parameterizations, e.g. experiments by Merity et al. (2017) outline a variety of training choices.

In this work, we revisit HMMs for NLP. We posit that for HMMs to be effective they must scale to much large state-spaces, utilizing appropriate parameterization, and employ the right regularization. Taking this into account, we develop a neural parameterization for HMMs that extends them to comparable size and structure of deep learning models, while allowing us to lazily instantiate distributions. We then develop a sparsity constraint that allows us to utilize very large HMMs, while maintaining efficient exact inference. Finally we incorporate a variant of dropout that both improves accuracy and reduces the computational overhead by an order of magnitude during training.

Experiments employ HMMs on two language modeling datasets and a supervised sequence tagging task. We find that our HMM extension significantly outperforms past HMMs as well as n-gram models. It also performs comparably to neural counterparts with a similar number of parameters while maintaining uncertainty over the state dynamics.

2 Related work

Prior work has demonstrated the benefits of neural parameterization of structured generative models. For HMMs, Tran et al. (2016) demonstrated improvements in POS induction with a neural parameterization of an HMM, while Kim et al. (2019) demonstrated improvements in grammar induction with a probabilistic context free grammar. Both of these works used latent variables with relatively small state spaces, as the goal of both was structure induction rather than language modeling itself. We extend the neural parameterization to much larger state space models.

We also draw inspiration from the experiments with cloned HMMs by Dedieu et al. (2019), who proposed to introduce sparsity constraints in scalar emission distribution of HMMs in order to make conditional inference tractable in large state spaces. Dedieu et al. (2019) trained a 30k state HMM on character-level language modeling by constraining every state to emit only a single character type. This particular constraint is problematic for language modeling at the word level, where the vocabulary size is much larger. We build on their work by proposing a sparsity constraint based on Brown clustering (Brown et al., 1992) which allows us to extend their work to vocabularies that are larger than the state space.

Another approach to scaling to larger state spaces is to initialize with a small state space then grow the state space via a split-merge process (Petrov et al., 2006; Huang, 2011). In particular, Huang (2011) learn an HMM for language modeling via this process. Additionally, the cloned HMM (Dedieu et al., 2019) can be seen as an HMM that starts with a single state per word, then splits every state into k states at the start with no subsequent splits or merges. The application of split-merge is an avenue for future work, as we focus on fixed-size state spaces for simplicity.

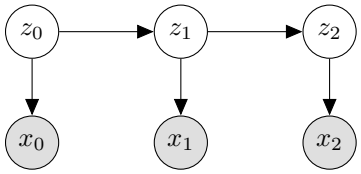

Figure 1: An HMM with tokens x_t and states z_t.

Other investigations into improving the performance of HMMs involved relaxing independence or modeling assumptions (Buys et al., 2018) to resemble a recurrent neural network, or through model combination with a recurrent neural network (Krakovna and Doshi-Velez, 2016). There are also extensions of HMMs, such as factorial HMMs (Ghahramani and Jordan, 1997; Nepal and Yates, 2013) and context free grammars (Kim et al., 2019). We leave scaling more expressive models to large state spaces for future work, and focus on scaling the basic HMM.

3 Background: HMMs

We are interested in learning a distribution over observed tokens $\mathbf{x} = \langle x_1, \ldots, x_T \rangle$, 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 $\mathbf{z} = \langle z_1, \ldots, z_T \rangle$, with each z_t from finite set $\mathcal{Z}$. For notational convenience, we define the starting state $z_0 = \epsilon$. The model is defined by the following generative process (shown in figure 1): For every time step $t \in \{0, \ldots, T\}$, choose a state given the previous state $z_t \mid z_{t-1}$ from the transition distribution $p(z_t \mid z_{t-1})$. Then choose a token given the current state $x_t \mid z_t$ from the emission distribution $p(x_t \mid z_t)$. This yields the joint distribution

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

The distributions are parameterized as follows

$$\begin{aligned} p(z_1 | z_0) &\propto e^{\psi_{z_1}} \\ p(z_t \mid z_{t-1}) &\propto e^{\psi_{z_t z_{t-1}}} \\ p(x_t \mid z_t) &\propto e^{\phi_{x_t z_t}} \end{aligned} \tag{2}$$

where each $\psi_{z_0}, \psi_{z_t z_{t-1}}, \phi_{x_t z_t} \in \mathbb{R}$. Thus the transition distribution $p(z_t \mid z_{t-1})$ is parameterized by $\psi \in \mathbb{R}^{|\mathcal{Z}| \times |\mathcal{Z}|}$, and the emission distribution $p(x_t \mid z_t)$ by $\phi \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{Z}|}$.

We distinguish two types of parameterizations: *scalar* and *neural*. A scalar parameterization simply uses $\theta = \{\phi, \psi\}$ to fit one model parameter for each distributional parameter. This results in $O(|\mathcal{Z}|^2 + |\mathcal{X}||\mathcal{Z}|)$ model parameters, since the transition and emission matrices must be explicitly represented. This parameterization can lead to either over or under parameterization for most NLP tasks. In contrast, a neural parameterization uses θ as the parameters of a neural network that generates ϕ and ψ for the distribution. These may be customized for the task through embedding or other layers, and separate the parameterization size from the hidden state size.

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. As the dynamic program is differentiable, we can optimize the likelihood with gradient ascent. [1]

(Talk about constructing matrices once per batch, decoupling cost of softmax over vocab and batch size, constructing log potentials via indexing, then logarithmic reduction DP.)

HMM / RNNs RNNs also model $p(\mathbf{x})$, but without introducing a latent variable for the state. Whereas RNNs encode the previous observations with a single distributed representation, HMMs encode the past as a distribution over a finite set of states. Given a hidden size of h for the RNN and a cost $v(h)$ for computing a matrix-vector product, the computational complexity of computing $p(x)$ for a single sequence is $O(Tv(h))$ due to the use of matrix-vector products at every timestep. For an HMM with a state space of size $|\mathcal{Z}| = h$ and matrix-vector product cost $v(h)$, the complexity of computing $p(\mathbf{x}) = \sum_z p(\mathbf{x}, \mathbf{z})$ on sequential hardware is also $O(Tv(h))$. However, on parallel device, we are able to compute $p(\mathbf{x})$ for an HMM in time logarithmic in T using the parallel prefix-sum trick. Assuming a cost $m(h)$ for matrix multiplication, this gives cost $O(\log(T)m(h))$ for an HMM on a parallel device. Quasi-RNNs (Bradbury et al., 2016) also have a logarithmic dependency on T by applying the same prefix-sum trick, but the operation applied is less expensive than a matrix multiplication.

HMMs cleanly separate out latent memory from the generative process of language. This separation makes them an important model for isolating observed properties and performing inference. However, this separation also means they require a large hidden state space to capture the underlying properties of language.

4 Method

Our goal is to vastly expand the state space of an HMM model, while maintaining a regularized neural parameterization and efficient exact inference.

4.1 Inducing Blocked Transition

Marginal inference for HMMs is quadratic in the state space $|\mathcal{Z}|$. This limits the size of the state space to the order of 100s, which prevents models from having enough states to capture the full history. However, we can improve the complexity in special cases. In particular, if we know that the probability of emitting a word x_t from a state z_t is 0, i.e. $p(x_t \mid z_t) = 0$ then we can ignore transitions into and from that state during inference.

Inspired by cloned HMMs (Dedieu et al., 2019), we add parameterization constraints to ensure that this property holds. Specifically, we require that for an observed token x a only fixed number of states z have $p(x \mid z) > 0$. For each word, we call this set $\mathcal{C}_x \subset \mathcal{Z}$. This yields the following constrained emission distribution:

$$p(x \mid z) \propto 1(z \in \mathcal{C}_x) e^{\phi_{xz}} \tag{3}$$

[1] An alternative method would be to employ expectation maximization, however there is no closed form solution for the M-step with a neural parameterization. Gradient ascent is applicable to both the scalar and neural parameterizations.

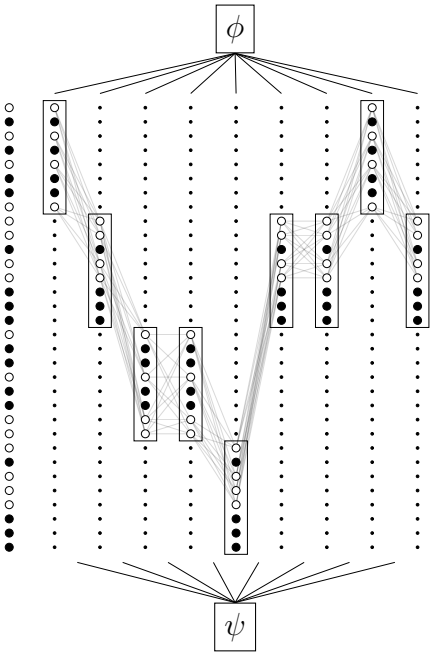

Figure 2: WILL REPLACE WITH TRELLIS A depiction of the emission sparsity constraints.

Exact marginal inference can be performed as

$$\begin{aligned} &p(\mathbf{x}) \\ &= \sum_{z_0 \in \mathcal{C}_{x_0}} p(z_0)p(x_0 \mid z_0) \sum_{z_1 \in \mathcal{C}_{x_1}} p(z_1 \mid z_0)p(x_1 \mid z_1) \\ &\quad \cdots \sum_{z_T \in \mathcal{C}_{x_T}} p(z_T \mid z_{T-1})p(x_T \mid z_T) \end{aligned} \tag{4}$$

This allows us to compute $p(\mathbf{x})$ with T matrix-vector products, each matrix of dimension $|\mathcal{C}_{x_t-1}| \times |\mathcal{C}_{x_t}|$. We choose sets $\mathcal{C}_x$ such that $\forall x, |\mathcal{C}_x| = k$ yielding a computation complexity of $O(Tk^2)$ rather than $O(T|\mathcal{Z}|^2)$. Please refer to Fig. 2 for an illustration of this method. (Describe here)

4.2 Dropout as State Reduction

We introduce a variant of dropout called state dropout that operates on the emission distribution. The goal of state dropout is to encourage usage of the full state space in HMMs as well as to reduce the complexity of marginalization in a manner that compounds with the sparse emission constraints discussed above.

State dropout prevents a token x from always being emit by a specific state z by zeroing out emission probabilities during training. Recall that the HMM has distributional parameters $\phi \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{Z}|}$ for the emission matrix, and the emission sparsity constraints associate each token x with a set of states $\mathcal{C}_x$ that are the only states which emit x with nonzero probability. State dropout samples a mask over states $\mathbf{b}_x \in \{0,1\}^{|\mathcal{C}_x|}$ such that $|\mathbf{b}_x| = n$ for each x. Given the mask, we then computes the following emission distribution:

$$p(x \mid z) \propto b_{xz} 1(z \in \mathcal{C}_x) e^{\phi_{xz}} \tag{5}$$

Performing marginal inference with state dropout requires $O(Tn^2)$ computation, where n is the number of nonzero elements of $\mathbf{b}_{\mathcal{C}}, \forall \mathcal{C}$.

4.3 Static Parameterization

We parameterize the transition and emission distributions using a residual network with LayerNorm. Let $\mathbf{a}, \mathbf{b} \in \mathbb{R}^h$, where h is the hidden dimension of our neural network and the size of our token and state embeddings. We use the following neural network for $i \in \{\phi, \psi\}$

$$\begin{aligned} f_i(A) &= g(\text{ReLU}(W_{i1}A)) \\ g_i(B) &= \text{LayerNorm}(\text{ReLU}(W_{i2}B) + B) \end{aligned} \tag{6}$$

to define the distributional parameters, where $A, B \in \mathbb{R}^{h \times |\mathcal{Z}|}$ and $W_{i1}, W_{i2} \in \mathbb{R}^{h \times h}$. Let $V_x \in \mathbb{R}^{h \times |\mathcal{X}|}$ be the token embeddings and $U_{\text{previous}}, U_{\text{current}}, U_{\text{preterminal}} \in \mathbb{R}^{h \times |\mathcal{Z}|}$ the state embeddings. The distributional parameters are given by

$$\begin{aligned} \phi &= V_x^\top f_\phi(U_{\text{preterminal}}) \\ \psi &= U_{\text{current}}^\top f_\psi(U_{\text{previous}}) \end{aligned} \tag{7}$$

where $\phi \in \mathbb{R}^{|X| \times |Z|}$ and $\psi \in \mathbb{R}^{|Z| \times |Z|}$.

Compared to fully representing ϕ, ψ with a scalar parameterization which would require $O(|\mathcal{Z}|^2 + |\mathcal{X}||\mathcal{Z}|)$, this neural parameterization takes $O(h^2 + h|\mathcal{Z}| + h|\mathcal{X}|)$ parameters where h is in the 100s and $|\mathcal{Z}|, |\mathcal{X}|$ are on the order of 10k.

5 Experiments

In this section we provide further details on the application of the methods described in the previous section to language modeling and sequence tagging.

5.1 Language modeling

In word-level language modeling the tokens $\mathbf{x}$ correspond to the words $\mathbf{w} = \langle w_1, \ldots, w_T \rangle$ in a sentence. We thus learn a model over words and states $p(\mathbf{w}, \mathbf{z}) = \prod_t p(w_t \mid z_t) p(z_t \mid z_{t-1})$.

We use Brown clusters (Brown et al., 1992) to create the emission constraints. Brown clusters are obtained by assigning every token type in $\mathcal{X}$ a state in a HMM, then continually merging states until a desired number of states is reached. The merge at each iteration attempts to merge the two states that results in best likelihood.

To apply the Brown clusters as an emission constraint, each Brown cluster cluster is assigned a disjoint set of k states such that the state space is partitioned. Each state corresponding to a Brown cluster is then allowed to emit only the words in that Brown cluster. This constraint corresponds to learning a refinement of the Brown clusters by splitting each cluster into k states.

Since the emission constraints using Brown clusters partitions the state space, we share state dropout masks for all word types within a Brown cluster.

(Parameterization of embeddings, factoring cluster + word to reduce params)

5.2 Sequence Tagging

We extend the language modeling HMM to part of speech (POS) tagging, where the tokens $\mathbf{x}$ are now the product of words $\mathbf{w}$ as well as tags $\mathbf{y} = \langle y_1, \ldots, y_T \rangle$.

At every timestep, our model emits both a word w_t and tag y_t independently given the states z_t. This yields the following joint distribution

$$p(\mathbf{w}, \mathbf{y}, \mathbf{z}) = \prod_t p(w_t \mid z_t) p(y_t \mid z_t) p(z_t \mid z_{t-1}) \tag{8}$$

We use the same emission constraints and dropout strategy as language modeling, but applied only to the word emission distribution $p(w_t \mid z_t)$. The tag emission distribution is unconstrained.

(Parameterization, CharCNN)

5.3 Datasets

Language modeling We evaluate on the `Penn Treebank` (Marcus et al., 1993) and `wikitext2` (Merity et al., 2016) datasets. `Penn Treebank` contains 929k tokens in the training corpus, with a vocabulary size of 10k. We use the preprocessing from Mikolov et al. (2011), which lowercases all words and substitutes words outside of the vocabulary with unks. `Wikitext2` contains 2M tokens in the training corpus, with a vocabulary size of 33k. Casing is preserved, and all words outside the vocab are unked. Both datasets contain inter-sentence dependencies, due to the use of documents consisting of more than a single sentence.

Part of speech tagging We use the Wall Street Journal portion of the `Penn Treebank` for evaluating POS tagging. We map all numbers to 0, and perform no further preprocessing. We train on sections 0-18, validate on sections 19-21, and test on 22-24 following Ma and Hovy (2016).

6 Results

We report perplexities for `Penn Treebank` in Table 1 and for `wikitext2` in Table **??**.

The HMMs in all cases outperform the feed-forward models (FF), but underperform the recurrent LSTMs. Since the HMMs require parameters linear in the number of hidden states, we find that the performance of the HMMs scales poorly compared to the other models which only require parameters that scale linearly with the vocbaulary size. Although representing and summing over the hidden states allows us to explicitly capture uncertainty in the hidden state, it proves to be a limiting factor in terms of performance.

In the remainder of this section we ablate and analyze the HMMs on `Penn Treebank`.

6.1 Error analysis

Table 1: Perplexities on the `Penn Treebank` dataset. The FF model is a 256-dim 2-layer feedforward neural network with a window of 4 previous tokens with 0.3 dropout. The LSTM is a 256-dim 2-layer recurrent neural network with 0.3 dropout. The HMM is a 64k-state HMM with 0.5 state dropout.

Dataset	Model	Num Params	Valid PPL	Test PPL
`Penn Treebank`	FF	2.8M	164	-
	LSTM	3.6M	93.55	88.83
	HMM	7.7M	125.02	115.82
`Wikitext-2`	FF		209	-
	LSTM		125	-
	HMM		167	-

7 Discussion

Acknowledgments

TBD

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.

Antoine Dedieu, Nishad Gothoskar, Scott Swingle, Wolfgang Lehrach, Miguel Lázaro-Gredilla, and Dileep George. 2019. Learning higher-order sequential structure with cloned hmms.

Zoubin Ghahramani and Michael I. Jordan. 1997. Factorial hidden markov models. *Mach. Learn.*, 29(2–3):245–273.

Zhongqiang Huang. 2011. *Modeling Dependencies in Natural Languages with Latent Variables*. Ph.D. thesis, University of Maryland.

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.

Xuezhe Ma and Eduard H. Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *CoRR*, abs/1603.01354.

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.

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.

Tomas Mikolov, Anoop Deoras, Stefan Kombrink, Lukás Burget, and Jan Cernocký. 2011. Empirical evaluation and combination of advanced language modeling techniques. pages 605–608.

Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. pages 1045–1048.

Anjan Nepal and Alexander Yates. 2013. Factorial hidden markov models for learning representations of natural language. *CoRR*, abs/1312.6168.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. page 433–440.

Lawrence R. Rabiner. 1990. *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, page 267–296. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

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

Ke M. Tran, Yonatan Bisk, Ashish Vaswani, Daniel Marcu, and Kevin Knight. 2016. Unsupervised neural hidden markov models. *CoRR*, abs/1609.09007.

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.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *CoRR*, abs/1409.2329.

A Hyperparameters

LSTM

- 2 layers

B Supplemental Material