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

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). However, Buys et al. (2018) reported HMM performance much worse than an n-gram model for language modeling, and (Krakovna and Doshi-Velez, 2016) experimented with 20 hidden states. We demonstrate that HMMs, once given enough states, outperform n-gram models and are comparable to recurrent neural networks in performance.

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. Other work has neuralized classic models, such as dependency models with valence (Han et al., 2017), hidden semi-Markov models (Wiseman et al., 2018), and context free grammars (Kim et al., 2019). All 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.

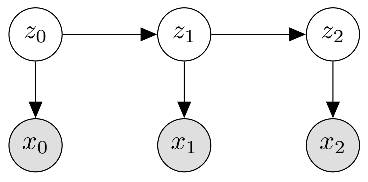
$$\begin{tikzcd} z_0 \arrow[r] \arrow[d] & z_1 \arrow[r] \arrow[d] & z_2 \arrow[d] \\ x_0 & x_1 & x_2 \end{tikzcd}$$

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

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 more complicated split-merge procedures is an avenue for future work, as we focus on fixed-size state spaces for simplicity.

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 $\mathbf{x} = \langle x_1, \dots, x_T \rangle$, 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 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 $t \in \{0, \dots, T\}$, 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]

HMMs and RNNs Recurrent neural networks (RNNs) are one of the main alternatives to HMMs, as they also model data with sequential structure and achieve strong performance doing so. However, HMMs do maintain advantages over RNNs in certain cases. As a fully observed model, an RNN does not decouple the latent dynamics from the observed. This rarely matches the true generative process of the data we wish to model. For example, HMMs are widely used in the case of known unknowns, such as acoustic modeling in speech recognition.

Additionally, there are computational benefits to HMMs that are not available in RNNs. Computing the likelihood of a sequence in an RNN is linear in the length of a sequence and quadratic in the size of the hidden state. For an HMM, the same is true with a naive approach: the cost is linear sequence length but quadratic in the size of the state space. However, inference in the HMM can be sped up to be logarithmic in sequence length on a parallel machine via the prefix-sum trick. 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.

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 Static Parameterization

A neural parameterization allows us to precisely control the represention of distributional parameters, affording a degree of flexibility not present in a scalar parameterization. The scalar parameterization of a very large HMM would take millions of parameters due to the transition matrix parameters $\psi \in \mathbb{R}^{|\mathcal{Z}|\times|\mathcal{Z}|}$. We use the intuition that introducing a parameter to model the attraction between each state is unnecessary, and instead use embeddings which allow the number of parameters to scale linearly with the size of the state space.

We parameterize the transition and emission distributions using a residual network with LayerNorm. Let $E \in \mathbb{R}^{v\times h}$ be an embedding matrix, where v is the number of embeddings and h is the hidden dimension of our neural network. We use the following neural network for $i \in \{\phi, \psi\}$

$$\begin{aligned} f_i(E) &= g_i(\text{ReLU}(EW_{i1})) \\ g_i(D) &= \text{LayerNorm}(\text{ReLU}(DW_{i2}) + D) \end{aligned} \tag{3}$$

[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.

to define the distributional parameters, where D has the same dimension as E and $W_{i1}, W_{i2} \in \mathbb{R}^{h \times h}$. Additionally, LayerNorm acts on the last dimension of its argument. Let $E_x \in \mathbb{R}^{|\mathcal{X}| \times h}$ be the token embeddings and $E_{\text{previous}}, E_{\text{current}}, E_{\text{preterminal}} \in \mathbb{R}^{|\mathcal{Z}| \times h}$ the state embeddings. The distributional parameters are given by

$$\begin{aligned} \phi &= E_x f_\phi(E_{\text{preterminal}})^\top \\ \psi &= E_{\text{current}} f_\psi(E_{\text{previous}})^\top \end{aligned} \tag{4}$$

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 $O(|\mathcal{Z}|^2 + |\mathcal{X}||\mathcal{Z}|)$ parameters, 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.

Given the parameters, we can then perform marginal inference by first constructing a trellis via indexing into ϕ and ψ (illustrated in Fig. 2), then computing marginals with dynamic programming. The clean separation between the computation of the distributional parameters and marginal inference allows the distributional parameters to be computed and then cached for use in inference. In practice, we compute the emission and transition distributions once per batch. This contrasts with recurrent neural networks and transformers, which must compute emission probabilities for every observation due to a theoretically infinite number of hidden states.

4.2 Inducing Blocked Transitions

Marginal inference for HMMs is quadratic in the size of 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

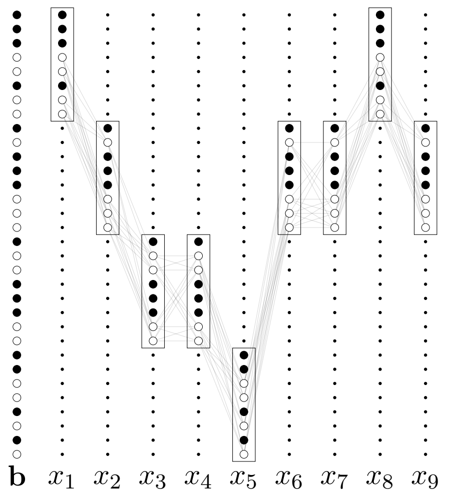

Figure 2: A depiction of the trellis for marginal inference after applying emission sparsity constraints and the state dropout method. The leftmost column depicts the state dropout mask **b**, which is used to prevent states from being considered during inference. For every token x_t, the only incoming edges with nonzero mass are those whose source was not dropped out, i.e. $b_{z_t} = 1$, and is within the previous partition, i.e. $z_{t-1} \in \mathcal{C}_{x_{t-1}}$.

each word, we call this set $\mathcal{C}_x \subset \mathcal{Z}$. This yields the following constrained emission distribution:

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

Exact marginal inference can be performed as

$$\begin{aligned} &p(\mathbf{x}) \\ &= \sum_{z_0 \in \mathcal{C}_{x_0}} p(z_0) p(x_0 \mid z_0) \sum_{z_1 \in \mathcal{C}_{x_1}} p(z_1 \mid z_0) p(x_1 \mid z_1) \\ &\quad \cdots \sum_{z_T \in \mathcal{C}_{x_T}} p(z_T \mid z_{T-1}) p(x_T \mid z_T) \end{aligned} \tag{6}$$

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.

4.3 Dropout as State Reduction

Although a neural parameterization has empirically been shown to find good optima, it requires that the distributional parameters be recomputed every time the model parameters are updated. This entails running a neural network $O(|\mathcal{Z}| + |\mathcal{X}|)$ times at every iteration of gradient descent.

Algorithm 1 Pseudocode for computing the likelihood of a corpus

```
Given: constraints $\mathcal{C}_x$ and model parameters
function ComputeLikelihood
    Compute parameters $\phi, \psi$ from model parameters
    for all examples $\mathbf{x}$ do
        Compute log potentials $\Phi = \text{LogPotentials}(\phi, \psi, \mathbf{x}, \mathcal{C}_x)$
        Compute evidence $\log p(\mathbf{x}) = \text{Forward}(\Phi)$
function ComputeLikelihoodDropout
    for all examples $\mathbf{x}$ do
        Sample dropout mask $\mathbf{b}$
        Let $\mathcal{B}_x$ be the remaining states after removing all dropped states in $\mathbf{b}$ from $\mathcal{C}_x, \forall x$
        Compute parameters $\phi_{\mathbf{b}}, \psi_{\mathbf{b}}$ ignoring states where $b_z = 0$
        Compute log potentials $\Phi = \text{LogPotentials}(\phi, \psi, \mathbf{x}, \mathcal{B}_x)$
        Compute evidence $\log p(\mathbf{x}) = \text{Forward}(\Phi)$
```

We introduce a variant of dropout called state dropout that lessens the computational burden of the neural parameterization at training time while also encouraging full use of the state space and reducing the complexity of marginalization.

State dropout prevents a state z from being considered during training. This can be implemented by adding another constraint to the emission matrix. Recall distributional parameters of the emission matrix $\phi \in \mathbb{R}^{|\mathcal{X}|\times|\mathcal{Z}|}$ and the emission sparsity constraints which associate each token x with a set of states $\mathcal{C}_x$. State dropout samples a mask over states $\mathbf{b} \in \{0,1\}^{|\mathcal{Z}|}$. Given the mask and the emission constraints defined in the previous section, we have the following emission distribution:

$$p(x \mid z) \propto b_z 1(z \in \mathcal{C}_x) e^{\phi_{xz}} \tag{7}$$

Performing marginal inference with state dropout requires $O(Tn^2)$ computation, where $n = \max_x |\{z : b_z = 1, z \in \mathcal{C}_x\}|$. Additionally, we do not need to compute the parameters of the transition and emission distributions that have been dropped. This reduces the cost of computing ϕ and ψ from $O(|\mathcal{Z}|^2 + |\mathcal{Z}||\mathcal{X}|)$ to $n^2 + n|\mathcal{X}|$.

(Unsatisfied, seems like there's a big jump from description to understanding trellis. Think this means section 3 is incomplete)

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 $\mathbf{w} = \langle w_1, \dots, w_T \rangle$ in a sentence. We thus learn a model over words and states $p(\mathbf{w}, \mathbf{z}) = \prod_t p(w_t \mid z_t) p(z_t \mid z_{t-1})$.

Parameterization We have two methods for parameterizing the state embeddings $E_j \in \mathbb{R}^{|\mathcal{Z}|\times h}$, where h is the dimension of the neural components and $j \in \{\text{previous}, \text{current}, \text{preterminal}\}$. The first is by introducing one set of embeddings and sharing it for all j. The second is to introduce a set of embeddings $E_{\text{state}} \in \mathbb{R}^{|\mathcal{Z}|\times h/2}$, and another set of embeddings $E_{\text{cluster}} \in \mathbb{R}^{m\times h}$ for each Brown cluster, where m is the number of Brown clusters. We then use a separate residual network f_j (as defined in Eqn. 3) to obtain $E_j = f_j([E_{\text{cluster}}, E_{\text{state}}])$. We refer to this as a factored state embedding. This reduces the number of parameters, as there are many more states than Brown clusters.

Emission constraints 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, every word x in a Brown cluster is assigned the same constraint set $\mathcal{C}_x$. We choose the $\mathcal{C}_x$ such that they are all of size k. This constraint corresponds to learning a refinement of the Brown clusters by splitting each cluster into k states.

We additionally compare the Brown cluster constraint with a uniform constraint that samples each $\mathcal{C}_x$ of size n independently and uniformly from all subsets.

State dropout We apply state dropout such that each partition $\mathcal{C}_x$ has exactly $n = |\mathcal{C}_x|p$ dropped states, where p is the dropout rate. We accomplish this by sampling from the uniform distribution over subsets, obtained by sampling iid Gumbel noise and selecting the top n indices.

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 $\mathbf{y} = \langle y_1, \dots, y_T \rangle$.

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.

We additionally employ a CharCNN word emission parameterization (Kim et al., 2015), as used by Ma and Hovy (2016).

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 (LDC99T42 treebank 3). 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

6.1 Language Modeling

We report perplexities for `Penn Treebank` in Table 1 and for `wikitext2` in Table 2.

The HMMs in all cases outperform the n-gram and feedforward 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`.

Number of states In Tbl. 3 we examine the effect of the number of states on perplexity. We find that performance continuously improves as we increase the size of the state space, up until we reach 65k states. It is possible that at 65k states, we are bottlenecked by other aspects of the model, such as the expressive power or learnability of the neural parameterization. Due to GPU memory constraints, we were unable to further explore larger parameterizations, both in terms of the dimension of the neural components as well as the size of the state space. However, we note that increasing the dimension of the neural component to 512 dimensions with $|\mathcal{Z}| = 32,768$ did not improve

Table 1: Perplexities on the `Penn Treebank` dataset. The top shows results from previous work, while the bottom shows our results for models with comparable computational cost. In particular, we compare models that have the same asymptotic inference cost: linear in the length of a sequence and quadratic in the hidden dimension. This is $h = 256$ for the FF model and LSTM and $|\mathcal{Z}| = 256$ for the HMM.

Model	Num Params	Valid PPL	Test PPL
KN-5 (Mikolov and Zweig, 2012)	2M	-	141.2
KN-5 + cache (Mikolov and Zweig, 2012)	2M	-	125.7
RNN (Mikolov and Zweig, 2012)	2M	-	124.7
Medium LSTM (Zaremba et al., 2014)	20M	86.2	82.7
Large LSTM (Zaremba et al., 2014)	66M	82.2	78.4
AWD-LSTM (Merity et al., 2017)	24M	60.0	57.3
AWD-LSTM + cache (Merity et al., 2017)	24M	53.9	52.8
HMM (Buys et al., 2018)	10M	284.59	-
HMM + sigmoid + RNN emit (Buys et al., 2018)	10M	142.31	-
256 dim FF 4-gram	2.8M	163.02	151.00
2 layer 256 dim LSTM	3.6M	93.55	88.83
32k state HMM	7.7M	125.02	115.82

Table 2: Perplexities on the `Wikitext-2` dataset. We report results from previous work on top, while the bottom shows our results for the same models and hyperparameters applied to `Penn Treebank`.

Model	Num Params	Valid PPL	Test PPL
A	b	c	d
This work FF		209	-
LSTM		125	-
HMM		167	-

Table 3: Perplexities on the `Penn Treebank` dataset. Obviously better as a graph. Increase number of states while holding $\mathcal{C}_x = 256$ and state dropout at 0.5.

m	Val PPL
1024	213.25
2048	199.98
4096	169.18
8192	150.22
16384	135.79
32768	125.02
65536	121.93

performance. Another hypothesis is that the marginal gains in likelihood decreases as we continue to increase the size of the state space.

Emission constraint ablation We next analyze the effect of the emission constraint and dropout on performance. We analyze first a 16k state HMM, keeping the total number of states fixed while varying the emission constraints and dropout. In the top section of Tbl. 4, we find that the performance is insensitive to the number of Brown clusters at 16k states. However, in the middle section, the model appears to be sensitive to the number of Brown clusters at 1k total states. The 1k state HMM with 4 Brown clusters matches the unconstrained 1k state HMM, while the HMM with 8 Brown clusters underperforms. This implies that there may be a loss in performance due to the emission constraints, however it is difficult this insight to extrapolate to larger HMMs. This observation warrants further investigation in future work.

We also experiment with a different style of constraint in the bottom section of Tbl. 4. Models using the uniform constraints as described in Sec. 5 are consistently outperformed by the Brown cluster constraints as measured by validation perplexity. The models with uniform constraints also reached better training

Table 4: Perplexities on the `Penn Treebank` dataset. Ablate number of brown clusters. Ablate Brown cluster constraints against uniform for 16k state model. Ablate Brown cluster emission constraints against small HMM. Let m be the number of clusters.

Constraint	$\|\mathcal{Z}\|$	$\|\mathcal{C}_x\|$	m	Val PPL
Brown	16384	512	32	136.58
Brown	16384	256	64	137.55
Brown	16384	128	128	134.06
Brown	16384	64	256	135.75
None	1024	-	-	180.34
Brown	1024	256	4	182.13
Brown	1024	128	8	194.16
Uniform	8192	128	-	150
Brown	8192	128	64	142
Uniform	16384	128	-	146
Brown	16384	128	128	136

perplexities, implying overfitting. (Need conclusion for this experiment)

Dropout and parameterization ablation We ablate dropout and model parameterization in Tbl. 5. We find that

6.2 Part-of-Speech Tagging

6.3 Error analysis

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.

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.

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.

Yoon Kim, Yacine Jernite, David A. Sontag, and Alexander M. Rush. 2015. Character-aware neural language models. *CoRR*, abs/1508.06615.

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.

Table 5: Perplexities on the `Penn Treebank` dataset. Dropout and parameterization ablation

Model	Num params	Val PPL	Time per epoch (s)
16384 state HMM	5.6M	135.79	159
- dropout	5.6M	145.48	363
- neural parameterization	423M	169.031	520 [2]

Table 6: Tagging accuracies on the Wall Street Journal (WSJ) portion of the `Penn Treebank`.

Model	Num params	Valid Acc	Test Acc
BRNN (Ma and Hovy, 2016)	-	96.56	96.76
BLSTM (Ma and Hovy, 2016)	-	96.88	96.93
BLSTM + CNN (Ma and Hovy, 2016)	-	97.34	97.33
BLSTM + CNN + CRF (Ma and Hovy, 2016)	-	97.46	97.55
HMM	19M	96.25	96.50
HMM + CNN Emit	9M	96.73	96.95

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.

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.

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 Hyperparameters

LSTM

- 2 layers

B Supplemental Material