Scaling Hidden Markov Models

Anonymous EMNLP submission

Abstract

The hidden Markov model (HMM) is a fundamental tool for sequence modeling that cleanly separates the hidden state from the emission structure. Unfortunately, this clean separation makes HMMs 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 scaling HMMs to language modelings 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 language models that are much more accurate than previous HMMs and ngram-based methods while nearing the performance of 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 even, in a few cases, to 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 from utilizing explicit hidden states, particularly for structured models. For instance in the area of language modeling, systems primarily 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, positing that competitive performance requires very large HMM (VL-HMMs). To scale to large state-spaces, we need shared parameterizations, efficient inference, and effective regularization. We develop a neural parameterization for HMMs that extends them to comparable size and structure of deep models. We combine this parameterization with a modeling constraint that allows us to utilize HMMs with very large state spaces, 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 VL-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

Several recent papers have combined HMMs with neural networks with moderate success. Buys et al. (2018) develop an approach to relax the independence and modeling assumptions to resemble a recurrent neural network. However, their results either report HMM performance much worse than an n-gram model for language modeling or require altering the fundametal probabilistic structure of the model. Krakovna and Doshi-Velez (2016) utilize model combination with a recurrent neural network to connect both approaches in a 20 state model. We demonstrate how to scale to orders of magnitude more states and show performance surpassing n-gram models.

Prior work has demonstrated the benefits of neural parameterization of structured generative models. For HMMs, Tran et al. (2016) demonstrate improvements in POS induction with a neural parameterization of an HMM. Other work has used neural parameterization for 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 use 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 with the aim of supervised modeling.

We also draw inspiration from the experiments with cloned HMMs by Dedieu et al. (2019), who propose to introduce sparsity constraints in scalar emission distribution of HMMs in order to make conditional inference tractable in large state spaces. They train 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.

Finally, 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

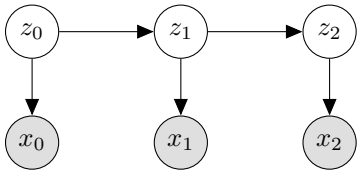

Figure 1: An HMM with tokens x_t and states z_t.

(Petrov et al., 2006; Huang, 2011). In particular, Huang (2011) learn an HMM for language modeling via this process. The application of more complicated split-merge procedures is an avenue for future work, as we focus on fixed-size state spaces.

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

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

The distributions are parameterized as follows

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

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. We can optimize the likelihood with gradient ascent. [1]

HMMs and RNNs Recurrent neural networks (RNNs) also model data with sequential structure and often achieve stronger performance. HMMs though are a useful alternative approach. 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. Inference in an HMM can be sped up to be logarithmic in sequence length on a parallel machine via the prefix-sum algorithm (Ladner and Fischer, 1980). 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 Scaling HMMs

The hypothesis of this work is that by expanding the state space of HMMs while while maintaining a regularized neural parameterization and efficient exact inference, we can build models that are competitive with RNNs. We propose three techniques for scaling these models: a factored neural parameterization, induced block transitions, and state dropout.

[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.

Our approach starts by defining two partitions: one for the hidden states $\pi : \mathcal{Z} \to [m]$, where m is the number of partitions and each $|\pi(z)| = k, \forall z$; and one for the observations $\sigma : \mathcal{X} \to [m]$. Given the structure defined by π and σ, we build models with large capacity that admit tractable inference.

4.1 Factored Neural 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 grows quadratically due to the transition matrix parameters $\psi \in \mathbb{R}^{|\mathcal{Z}|\times|\mathcal{Z}|}$. We use the intuition that introducing a parameter for each state pair is unnecessary, and instead employ state embeddings which allow the number of parameters to scale linearly with the size of the state space.

Let $E_x \in \mathbb{R}^{|\mathcal{X}|\times h}$ be the token embeddings and $E_{\text{out}}, E_{\text{in}}, E_{\text{emit}} \in \mathbb{R}^{|\mathcal{Z}|\times h}$ be embeddings for leaving, entering, and emitting from a state respectively. The distributional parameters are given by

$$\phi = E_x f_\phi(E_{\text{emit}})^\top \quad \psi = E_{\text{out}} f_\psi(E_{\text{in}})^\top \tag{3}$$

where $\phi \in \mathbb{R}^{|X|\times|Z|}$ and $\psi \in \mathbb{R}^{|Z|\times|Z|}$, and f is an MLP. [2]. 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.

We further parameterize the embeddings $E_{in}, E_{out}, E_{emit}$ by factoring them into partition and state embeddings. We introduce m partition embeddings $e_p \in \mathbb{R}^{h/2}$, and construct the partition embeddings $E_\pi = [e_{\pi(1)}, e_{\pi(2)}, \ldots, e_{\pi(|\mathcal{Z}|)}] \in \mathbb{R}^{|\mathcal{Z}|\times h/2}$. Given state embeddings $E_{\mathcal{Z}} \in \mathbb{R}^{|\mathcal{Z}|\times h/2}$, we define

$$E_{\text{in}} = f_{\text{in}}([E_\pi, E_{\mathcal{Z}}]) \tag{5}$$
$$E_{\text{out}} = f_{\text{out}}([E_\pi, E_{\mathcal{Z}}]) \tag{6}$$
$$E_{\text{emit}} = f_{\text{emit}}([E_\pi, E_{\mathcal{Z}}]) \tag{7}$$

This parameterization further reduces the number of model parameters to $O\left(h^2 + \frac{h}{2}|\mathcal{Z}| + \frac{h}{2}m + h|\mathcal{X}|\right)$.

[2] For the MLP we use the following neural network

$$\begin{aligned} f_i(E) &= g_i(\text{ReLU}(EW_{i1})) \\ g_i(D) &= \text{LayerNorm}(\text{ReLU}(DW_{i2}) + D) \end{aligned} \tag{4}$$

where D has the dimensions of E and $W_{i1}, W_{i2} \in \mathbb{R}^{h\times h}$.

Given the distributional parameters, we can then perform marginal inference separately. 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. For training, we can compute the emission and transition distributions once per batch. This contrasts with RNNs and other auto-regressive neural networks, which must compute emission probabilities for every observation.

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. For instance, 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 each observed token x only has a fixed number of states z with $p(x \mid z) > 0$. For each word, we call this support set $\mathcal{C}_x \subset \mathcal{Z}$. This yields the following constrained emission distribution:

$$p(x \mid z) \propto \mathbf{1}(z \in \mathcal{C}_x) e^{\phi_{xz}} \tag{8}$$

Exact marginalization can be performed by computing

$$\begin{aligned} &p(\mathbf{x}) \\ &= \sum_{z_0 \in \mathcal{C}_{x_0}} p(z_0) p(x_0 \mid z_0) \sum_{z_1 \in \mathcal{C}_{x_1}} p(z_1 \mid z_0) p(x_1 \mid z_1) \\ &\quad \cdots \sum_{z_T \in \mathcal{C}_{x_T}} p(z_T \mid z_{T-1}) p(x_T \mid z_T) \end{aligned} \tag{9}$$

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 the sets $\mathcal{C}_x$ by using the partitions of the states π and and tokens ρ. We then have $\mathcal{C}_x = \{z : \pi(z) = \rho(x)\}$, yielding a computation complexity of $O(Tk^2)$ rather than $O(T|\mathcal{Z}|^2)$ as each partition $|\pi(z)| = k, \forall z$.

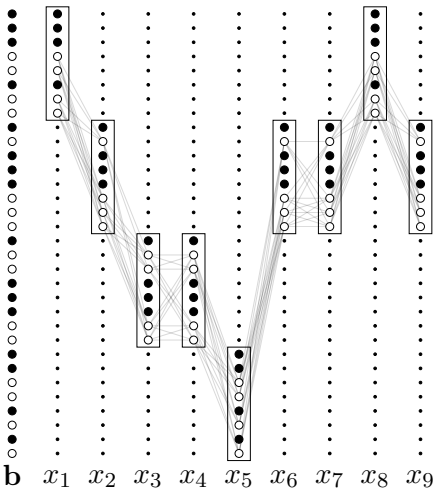

Figure 2: HMM search space after applying emission sparsity constraints and the state dropout method. The leftmost column depicts the state dropout mask $\mathbf{b}$, which is used to prevent states from being considered during inference. For every token x_t, the only incoming edges with nonzero mass are those whose source was not dropped out, i.e. $b_{z_t} = 1$, and is within the previous partition, i.e. $z_{t-1} \in \mathcal{C}_{x_{t-1}}$. The states are partitioned into groups of 8, with the observation partitions $\rho(x_1) = 1, \rho(x_2) = 2$, and so on.

4.3 Dropout as State Reduction

Even with a factored parameterization, the very large state space of the model makes it possible for the model to use states to memorize training patterns. To encourage generalization through distributed state usage, we introduce dropout to the model.

We propose a form of HMM state dropout that removes states from use entirely for a batch. This forces the model to use different states while also leading to efficiency improvements for computing parameterization and inference.

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:

$$p(x \mid z) \propto b_z \mathbf{1}(z \in \mathcal{C}_x) e^{\phi_{xz}} \tag{10}$$

Performing marginal inference with state dropout allows us to skip states that are removed. We utilize the constraint sets induced by π and ρ to ensure that exactly n states remain in each constraint set $\mathcal{C}_x$ after applying dropout. This is

Algorithm 1 Pseudocode for computing the likelihood of a corpus with state dropout

Given: constraints $\mathcal{C}_x$ and model parameters
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 $\log p(\mathbf{x}) = \text{FORWARD}(\Phi)$

accomplished by subsampling n states from each partition in the preimage of π, which preserves the block structure observed in Fig. 2 and allows the number of outgoing edges at each timestep to remain equal to n.

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 $n^2 + n|\mathcal{X}|$.

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.

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

State and word partitions We use Brown clusters (Brown et al., 1992) to construct the state and word partitions. 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 the best likelihood.

Importantly, every word is emit by only a single state, yielding the word partition ρ where the number of partitions m equal to the number of Brown clusters. We then create the state partition π by assigning each of the states in the Brown model k states in the resulting HMM.

We additionally compare the partition induced by Brown clustering with a uniform constraint that samples each $\mathcal{C}_x$ of size n independently and uniformly from all subsets of $\mathcal{C}$. This foregoes a partitioning, which makes it difficult to apply state dropout. We therefore apply a version of dropout that does not have block structure and zeroes out elements of the transition matrix randomly.

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

$$p(\mathbf{w}, \mathbf{y}, \mathbf{z}) = \prod_t p(w_t \mid z_t) p(y_t \mid z_t) p(z_t \mid z_{t-1}) \tag{11}$$

We use the same partitions, 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).

Note that at test time, the computing the condition distribution $p(\mathbf{y} \mid \mathbf{x})$ is intractable, as marginalizing over $\mathbf{z}$ in $\sum_{\mathbf{z}} p(\mathbf{y}, \mathbf{z} \mid \mathbf{x})$ without clamping a $\mathbf{y}$ is exponential in T. At training time, we are able to condition on the gold $\mathbf{y}$ and inference is tractable. We therefore perform approximate inference at test

time through the following two-step procedure: 1) ignore the tag emission distributions and obtain the distributions $\hat{p}(z_t \mid \mathbf{x})$ for each position t, then 2) predict the tag using $\sum_{z_t} p(y_t \mid z_t)\hat{p}(z_t \mid \mathbf{x})$. This method limits interaction between tags.

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.

On `Penn Treebank`, we see in Tbl. 1 that a 32k state HMM is able to outperform the n-gram models as reported by Mikolov and Zweig (2012), achieving a test perplexity of 115.8 versus 141.2 respectively. However, we find that the HMM underperforms RNN-based models. (need longer-term dependency experiment, measure trend as num states is increased).

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

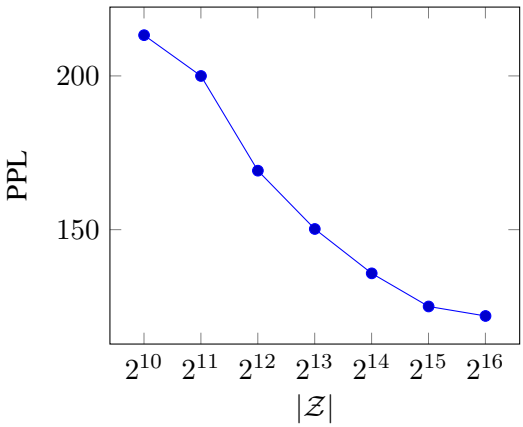

Figure 3: Perplexities on the `Penn Treebank` dataset as a function of the state size $|\mathcal{Z}|$. We hold the emission constraints fixed using 128 Brown clusters, and state dropout at 0.5.

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. However, we note that increasing the dimension of the neural component from 256 to 512 dimensions with $|\mathcal{Z}| = 32,768$ total states did not improve 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 the performance of a 16k state HMM, keeping the total number of states fixed while varying the emission constraints and dropout. In the top section of Tbl. 2, 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.

We also experiment with the uniform constraints as described in Sec. 5. In the bottom section of Tbl. 2, 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 perplexities despite better training perplexities, a symptom of overfitting.

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	284.6	-
HMM + sigmoid + RNN emit (Buys et al., 2018)	10M	142.3	-
256 dim FF 4-gram	2.8M	163.0	151.0
2 layer 256 dim LSTM	3.6M	93.6	88.8
32k state HMM	7.7M	125.0	115.8

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.

Constraint	$\lvert\mathcal{Z}\rvert$	$\lvert\mathcal{C}_x\rvert$	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 2: Perplexities on the `Penn Treebank` dataset. We ablate the effect of the number of Brown clusters, examine whether there may be a drop in performance due to the emission sparsity constraint, and compare the Brown cluster constraint to a uniform baseline. All models have 0.5 state dropout, except for the 1k state HMMs, which have no dropout. We use m to indicate the number of clusters.

In conclusion, we find the Brown cluster emission constraints to achieve reasonable performance in HMMs with large state spaces. We also observe that model performance is sensitive to the emission constraints, motivating future work towards exploring learning emission constraints while keeping inference tractable.

Dropout and parameterization ablation We ablate state dropout and model parameterization in Tbl. 3. We find that state dropout results in both an improvement in perplexity and a large improvement in time per epoch. The train-val gap for the model without state dropout was much larger than the model with dropout, highlighting the effectiveness of state dropout as regularization.

The scalar parameterization, which was run with 0.5 state dropout, has a massive number of model parameters at 423M, compared to the neural parameterization with 5.6M parameters. Although the neural and scalar parameterizations reach a similar training perplexity, the neural model generalizes better on validation.

We additionally ablate the factored state embeddings (add to table), and find that the performance of the factored embeddings drops when the number of clusters is too small. (Makes sense that) (Indicates number of parameters not necessarily the best comparison)

6.2 Part-of-Speech Tagging

We find in Tbl. 4 that the HMM outperforms the BRNN and BLSTM baselines of (Ma and Hovy, 2016), but underperforms the BLSTM with CNN and CRF variants. Additionally, we found that incorporating pretrained embeddings in the HMMs did not improve performance. NEED ERROR ANALYSIS

Model	Num params	Train PPL	Val PPL	Time per epoch (s)
16384 state HMM	5.6M	122	136	159
- dropout	5.6M	89	145	363
- neural parameterization	423M	119	169	520 [3]

Table 3: We report perplexities on the `Penn Treebank` dataset for a 16k state HMM with 0.5 state dropout, and ablate dropout and the neural parameterization one at a time.

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

Table 4: Tagging accuracies on the Wall Street Journal (WSJ) portion of the `Penn Treebank`. We use a HMM with 32k states and 0.5 state dropout.

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.*, 3:1137–1155.

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.

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.

Richard E. Ladner and Michael J. Fischer. 1980. Parallel prefix computation. *J. ACM*, 27(4):831–838.

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.

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.

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