Scaling Hidden Markov Language Models

Who Me
Affiliation / Address line 1
Affiliation / Address line 2
Affiliation / Address line 3
email@domain

No You
Affiliation / Address line 1
Affiliation / Address line 2
Affiliation / Address line 3
email@domain

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 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 supervised 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 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. 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 latent hidden states and toward deterministic deep neural 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 language modeling, 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. 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. Buys et al. (2018) develop an approach to relax the HMM with results that either have HMM performance much worse than an n-gram model or require altering the probabilistic structure. 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.

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 (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 $\mathbf{x} = \langle x_1, \dots, x_T \rangle$, 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 $z_0 = \epsilon$. 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

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

with transitions $\psi \in \mathbb{R}^{|\mathcal{Z}| \times |\mathcal{Z}|}$ and emissions $\phi \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{Z}|}$. We refer to $p(z_t \mid x_t)$ as $\mathbf{O}$ and $p(x_t \mid x_{t-1})$ as $\mathbf{T}$.

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 ($O(|\mathcal{Z}|^2 + |\mathcal{X}||\mathcal{Z}|)$ model parameters). A neural parameterization uses θ as the parameters of a neural network that generates ϕ and ψ for the distribution, which allow for task based factorization.

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).

HMMs and RNNs Recurrent neural networks (RNNs) do not attempt to decouple the latent dynamics from the observed. This often leads to improved accuracy, but does not allow for posterior inference or for directly incorporating additional state information. A further benefits of HMMs is that, unlike RNNs, their commutative structure allow for parallel inference via the prefix-sum algorithm (Ladner and Fischer, 1980).[1]

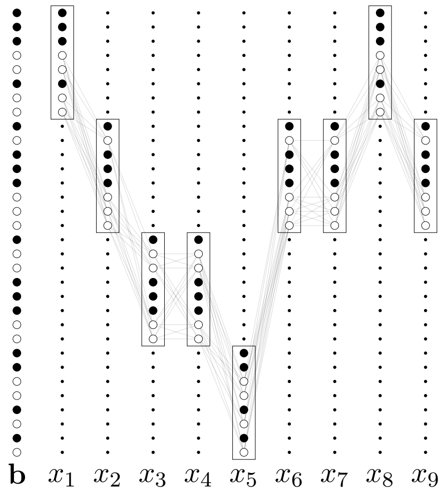

Figure 1: HMM search space after applying emission sparsity constraints and the state dropout method.

4 Scaling HMMs

Blocked Emissions Marginal inference for HMMs is quadratic in the size of the state space $|\mathcal{Z}|$, which inherently limits their representational capacity. However, we can improve the complexity in special cases. For instance, inspired by cloned

[1]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.

HMMs (Dedieu et al., 2019) 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.

We enforce a stronger constraint that the HMM have rectangular fixed-width blocked emissions,

$$\mathbf{O} = \begin{bmatrix} \mathbf{O}^1 & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & \mathbf{O}^M \end{bmatrix}$$

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

Our parameterization is chosen to ensure this property holds.

Exact marginalization can be computed as,

$$\begin{aligned} p(\mathbf{x}) = \sum_{z_1 \in \mathcal{Z}_{x_1}} & p(z_1 \mid z_0) p(x_1 \mid z_1) \times \\ \cdots \sum_{z_T \in \mathcal{Z}_{x_T}} & p(z_T \mid z_{T-1}) p(x_T \mid z_T) \end{aligned} \tag{3}$$

This gives a serial complexity of $O(T(|Z|/M)^2)$

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

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

The HMM distributional parameters are given by,

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

where $\phi \in \mathbb{R}^{|X| \times |Z|}$ and $\psi \in \mathbb{R}^{|Z| \times |Z|}$. MLP architecture follows (Kim et al., 2019). details are in supplementary materials. This neural parameterization takes $O(h^2 + h|\mathcal{Z}| + h|\mathcal{X}|)$ parameters.

Note that parameter computation is independent of inference and can be cache at test-time. For training we compute them once per batch. For RNNs and similar models, emission probabilities must be recomputed for each token.

Algorithm 1 VL-HMM Training

```
Given: block structure and model parameters
Sample block-wise dropout mask $\mathbf{b}$
Compute $\phi, \psi$ ignoring $b_z = 0$
for all batch examples $\mathbf{x}$ do
    $\Phi = \text{LogPotentials}(\phi, \psi, \mathbf{x}, \mathbf{b})$
    $\log p(\mathbf{x}) = \text{Forward}(\Phi)$
Update embeddings $\mathbf{E_z}, \mathbf{E_x}, \mathbf{E}_\pi$
```

Dropout as State Reduction 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, which has the added benefit of speeding up inference.

State dropout acts on each emission block $\mathbf{O}_1 \dots \mathbf{O}_M$ independently. Recall each block has $|\mathcal{Z}|/M$ columns. For each, we sample a binary dropout mask by sampling $\lambda_{\text{drop}} \times (|\mathcal{Z}|/M)$ dropped row indices uniformly without replacement. We concatenate these to a global vector $\mathbf{b}$, which, along with the previous constraints, ensures,

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

State dropout gives a larger practical speed up for both parameter computation and inference. For a dropout factor of 0.5 we get a $4\times$ speed improvement for both, due to reduction of possible transitions. This structured dropout is also easier to exploit on GPU, since it maintains block structure with fixed-height.

5 Experiments: Language modeling

In this section we provide further details on the application of the methods described in the previous section to word-level language modeling, where 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 $p(\mathbf{w}, \mathbf{z}) = \prod_t p(w_t \mid z_t) p(z_t \mid z_{t-1})$.

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.

5.1 Datasets

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.

We construct the Brown clusters on the training portions of both datasets.

6 Results

6.1 Language Modeling

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

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`.

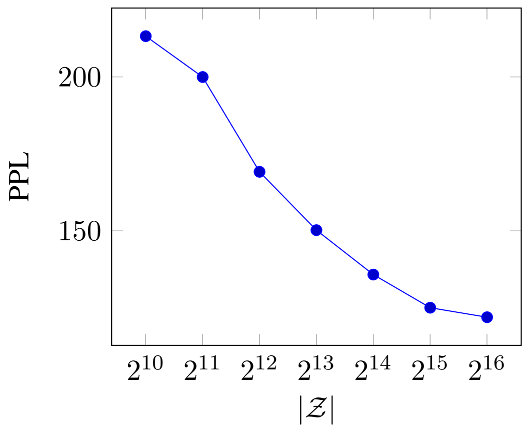

Figure 2: Perplexities on the `Penn Treebank` dataset as a function of the state size $|\mathcal{Z}|$. We hold the emission constraints fixed using 128 Brown clusters, and state dropout at 0.5.

Number of states In Tbl. 2 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. 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. It is possible that

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 val-

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	-
KN-5	2.2M	161.3	157.2
256 dim FF 5-gram	2.9M	159.9	152.0
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.

idation perplexity. The models with uniform constraints also had poor validation perplexities despite better training perplexities, a symptom of overfitting.

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, and find that performance of independent state embeddings is similar to a model with factored embeddings but with fewer parameters. We additionally found that if the number of clusters is too small (i.e. 64 or 32) while keeping the total number of hidden states fixed to 16384, performance on validation drops to 143.

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.

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
- state emb factorization	7.2M	115	134	142
- neural parameterization	423M	119	169	520 [2]

Table 3: We report perplexities on the `Penn Treebank` dataset for a 16k state HMM with 0.5 state dropout and 128 Brown clusters, 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.

6.2 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

6.3 Error analysis

7 Conclusion

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.

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.

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.

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.

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