Scaling Hidden Markov Models

April 28, 2021

Latent Variables Models in NLP

- NLP benchmarks are dominated by fully observed models
    - Transformers
    - Previously, Recurrent Neural Networks
- We instead explore latent variable models

Latent Variable Models: Motivation

- LVMs posit a generative process involving unseen variables
- Maintain uncertainty over latent representations, rather than just output correlations
- Often improves interpretability and controllability
- Bottlenecked by the computational complexity of inference

Research Question

To what extent is the performance of tractable latent variable models limited by scale and choices in parameterization?

This work: Scale hidden Markov models (HMMs) on language modeling using techniques drawn from recent advances in neural networks

Language Modeling

How now, brown _____

- Given the words seen so far, predict the next word
- Language requires modeling long-range phenomena

Hidden Markov Models in NLP

- Simplest latent variable models for time series data
- Are thought to be very poor language models
- We show they are better than previously thought, once scaled

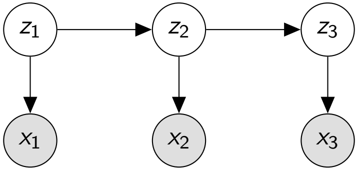
Background: Hidden Markov Models

Hidden Markov Models (HMMs)

- Classical models for unsupervised per-word tag induction
  - Part-of-speech induction
  - Word alignment for translation
- Admits tractable exact inference
  - Strong conditional independence assumptions
  - Finite set of discrete latent states

Hidden Markov Models (HMMs)

For times t, model states $z_t \in [Z]$, and tokens $x_t \in [X]$,



This yields the joint distribution

$$p(x, z) = \prod_t p(x_t \mid z_t) p(z_t \mid z_{t-1})$$

with

start state	$p(z_1)$,
transitions	$p(z_t \mid z_{t-1})$,
and emissions	$p(x_t \mid z_t)$

represented as vectors and matrices

Inference

Given observed $x = (x_1, \dots, x_T)$ We wish to maximize

$$p(x) = \sum_{z_1} \cdots \sum_{z_T} p(x, z) = \alpha_1^\top \Lambda_2 \Lambda_3 \cdots \Lambda_T \mathbf{1},$$

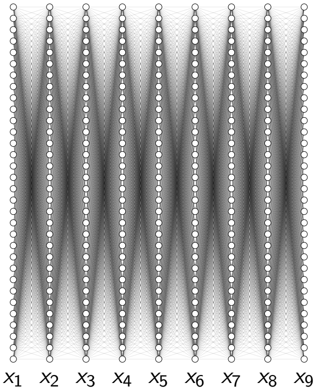
where we have the

$$\begin{aligned} &\text{start,} & [\alpha_1]_{z_1} &= p(x_1 \mid z_1) p(z_1), \\ &\text{and transition operators,} & [\Lambda_t]_{z_{t-1}, z_t} &= p(x_t \mid z_t) p(z_t \mid z_{t-1}) \end{aligned}$$

The result of each matvec has the alphas of the forward algorithm, i.e. $\alpha_3 = \alpha_1 \Lambda_2 \Lambda_3$ has entries corresponding to $p(z_3, x_{1:3})$

Inference

$$p(x) = \alpha_1^\top \Lambda_2 \cdots \Lambda_T \mathbf{1}$$



- Each node corresponds to a state
- Each edge to an entry in the transition operator matrix

Scaling HMMs

Lessons from Large Neural Language Models

Large models perform better but are . . .

1. Slow to train
2. Prone to overfitting

We must overcome these issues when scaling HMMs

3 Techniques for Training Large HMMs

- Compact neural parameterization

  ⬆ Generalization

- State dropout

  ⬆ Speed ⬆ Generalization

- Block-sparse emission constraints

  ⬆ Speed

- Will cover a fourth in the second part of this talk

Technique 1: Neural Parameterization

- The transition and emission matrices have Z^2 and ZX entries
- Causes the number of parameters to explode as the state size increases
- We instead use a low-dimensional decomposition of all conditional distributions that greatly reduces the number of parameters

Neural Parameterization: Softmax

For both the transition and emission matrices, we use a softmax parameterization, which assumes a nonlinear D-dimensional decomposition

$$\boxed{W} \propto \exp\left(\boxed{U} \times \boxed{V^\top}\right)$$

with embeddings $U \in \mathbb{R}^{Z \times D}$, $V \in \mathbb{R}^{Z \times D}$ or $\mathbb{R}^{X \times D}$

- Can further parameterize U or $V = \mathrm{MLP}(E_u)$

Technique 2: State Dropout

- Dropout is a common technique for regularizing neural networks
  - Reduces a network's reliance on a particular neuron
- Extend dropout to the states of an HMM
  - Encourage broad utilization of all states

State Dropout

- At each batch, sample dropout mask $\mathbf{b} \in \{0,1\}^Z$
- Compute distributional parameters by indexing into embeddings U, V

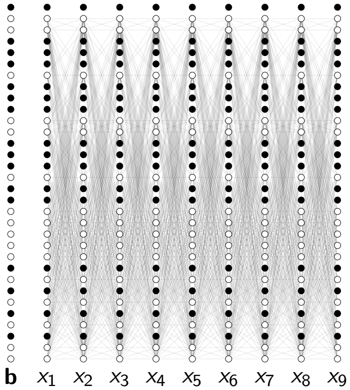
$$\left(\mathbf{b} \circ \boxed{U_{\text{trans}}}\right) \times \left(\mathbf{b} \circ \boxed{V_{\text{trans}}}\right)^\top$$

(a) Unnormalized transition logits

$$\left(\mathbf{b} \circ \boxed{U_{\text{emit}}}\right) \times \boxed{V_{\text{emit}}}^\top$$

(b) Unnormalized emission logits

State Dropout: Inference



- Shaded nodes depict dropped states
- Ignore dropped states during inference

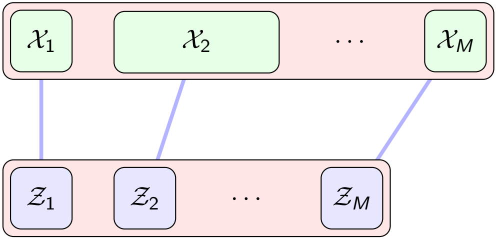
Technique 3: Block-Sparse Emission Constraints

- Reduce cost of marginalization by enforcing structure
- Only allow each word to be emit by a subset of states
- Cost of inference is quadratic in the size of the largest subset due to sparsity

Block-Sparse Emission Constraints: Alignment

Start with a joint partitioning of both states and wards

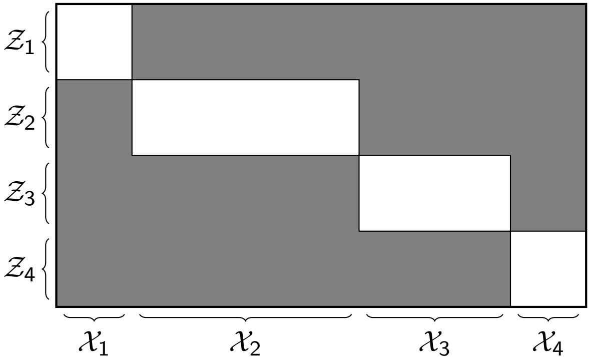
Indices $m \in [M]$ State partitions $\mathcal{Z}_m$ Word partitions $\mathcal{X}_m$



Block-Sparse Emission Constraints

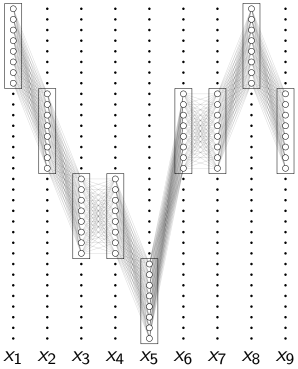
Given the unnormalized emission logits,

- Mask out unaligned state-word entries
- Normalize rows across words in aligned partition

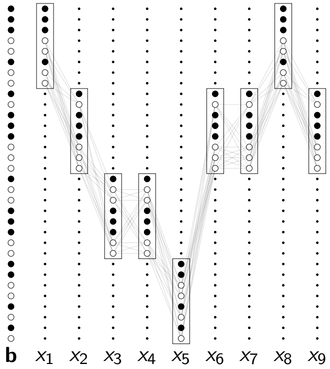


Block-Sparse Emissions: Inference

Given each word x_t, only the states in the correct group can occur



(a) Block-sparse emission



(b) With state dropout

Method Recap

- Compact neural parameterization

  ⬆ Generalization

- State dropout

  ⬆ Speed ⬆ Generalization

- Block-sparse emission constraints

  ⬆ Speed

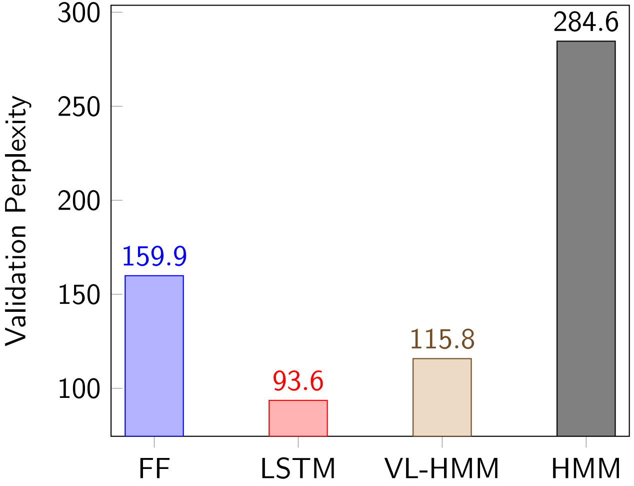
- A fourth after experiments

Experiments

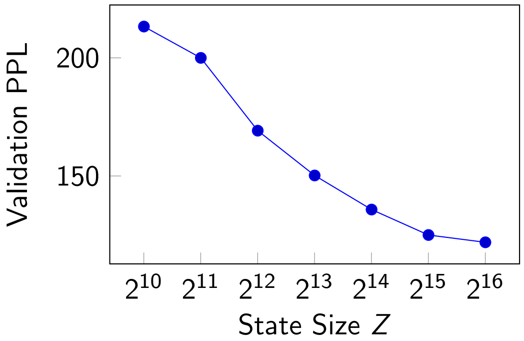
Experiments

- Language modeling on Penn Treebank
- Baselines
  - Feedforward 5-gram model
  - 2-layer LSTM
  - A 900 state HMM (Buys et al 2018)
- Model
  - 2^{15} (32k) state very large HMM (VL-HMM)
  - $M = 128$ groups (256 states per type), obtained via Brown Clustering
  - Dropout rate of 0.5 during training

Results on PTB Validation Data



State Size Ablation



Validation perplexity on PTB by state size ($\lambda = 0.5$ and $M = 128$)

Other Ablations

Model	Param	Train	Val
VL-HMM (2^{14})	7.2M	115	134
- neural param	423M	119	169
- state dropout	7.2M	88	157

Discussion

- Greatly scaled the state size of HMMs
- Performance improved with increasing state size
- Still a large gap between RNNs and HMMs
- Does the emission sparsity constraint improve computation complexity at the price of accuracy?

Speeding up HMMs with Low-Rank Decompositions

Fast Inference with Low-Rank Decompositions

- The previous approach relied a pre-specified emission sparsity constraint
- Can we scale inference with a weaker constraint?
- Exploit structure in the transition matrix to speed up inference

Inference

Start by unpacking inference to reveal the most expensive step

$$p(x) = \alpha_1^\top \Lambda_2 \Lambda_3 \cdots \Lambda_T \mathbf{1}$$

with

start,	$[\alpha_1]_{z_1} = p(x_1 \mid z_1) p(z_1),$
and transition operators,	$[\Lambda_t]_{z_{t-1}, z_t} = p(x_t \mid z_t) p(z_t \mid z_{t-1})$

Inference

Decompose transition operators into transition matrix A and emission matrix O

$$\begin{aligned} p(x) &= \alpha_1^\top \Lambda_2 \cdot \Lambda_T \mathbf{1} \\ &= \alpha_1^\top (A \operatorname{diag}([O]_{\cdot,x_2})) \cdots \Lambda_T \mathbf{1} \\ &= \alpha_1^\top A \operatorname{diag}([O]_{\cdot,x_2}) \cdots A \operatorname{diag}([O]_{\cdot,x_T}) \mathbf{1} \end{aligned}$$

where the most expensive steps are the matrix-vector products $\alpha_t^\top A$, which take $O(Z^2)$ computation

Fast Matrix-Vector Products

- Goal is to reduce the naive matvec complexity of $O(Z^2)$
- Various methods
  - Sparsity (nnz entries)
  - Fast Fourier Transform ($Z \log Z$)
  - Low-Rank decomposition (ZR)
- We utilize low-rank decompositions

Low-Rank Factorization

Factor transition matrix A into product of skinny matrices

$$\boxed{\alpha^\top} \times \boxed{A} = \boxed{\alpha^\top} \times \boxed{U} \times \boxed{V^\top}$$

resulting in two matrix-vector products of cost $O(ZR)$ each

- Constraint: Entries of A must be nonnegative
- Solution: Use a nonnegative matrix factorization (NMF)

$$A = \phi(U)\phi(V)^\top,$$

with $\phi : \mathbb{R}^D \to \mathbb{R}_+$

Method Recap

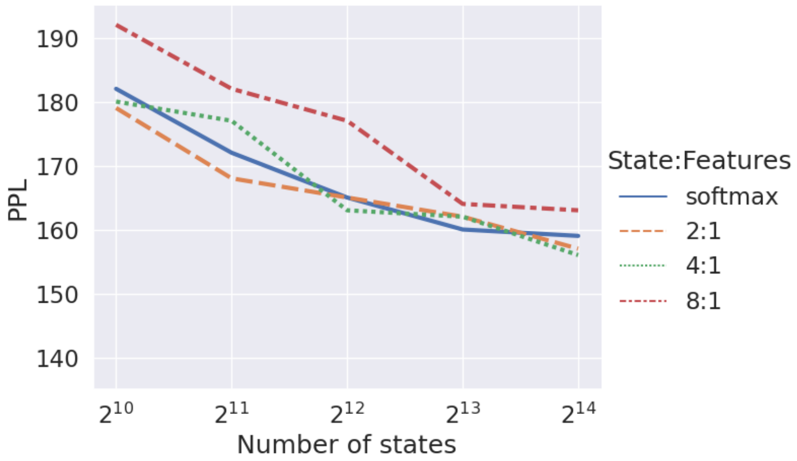
- Target key $O(Z^2)$ matvec step in inference
- Use NMF to reduce cost to $O(ZR)$
- How small can R be relative to Z without sacrificing accuracy?

Experiments

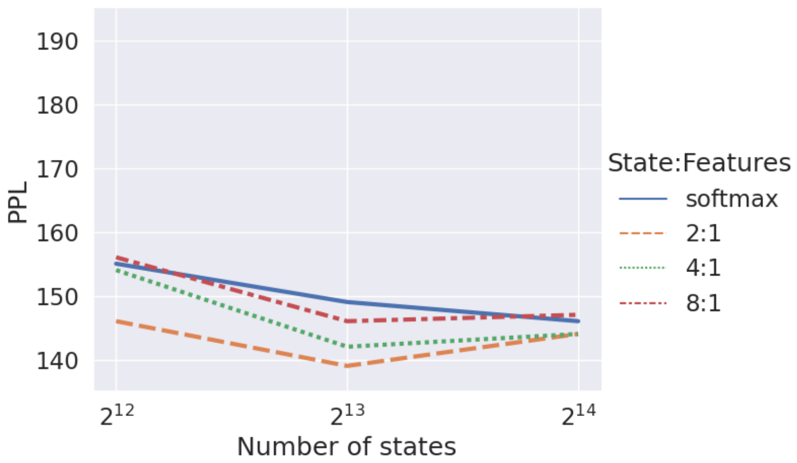
Experiments

- Language modeling on PTB
- Feature map $\phi(\mathbf{x}) = \exp(W\mathbf{x})$, with learned $W \in \mathbb{R}^{D \times R}$
- No sparsity constraints

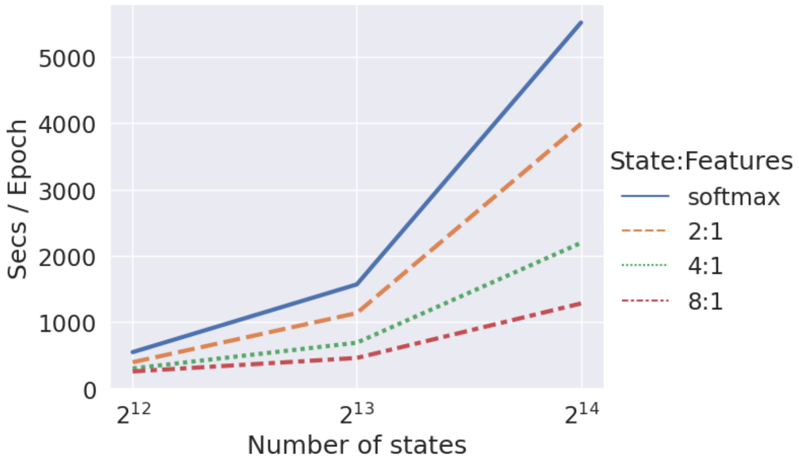
Scaling on PTB (Validation)



Further Scaling on PTB with Dropout (Validation)



Speed Comparison



Conclusion (TODO)

- Hopeful that HMMs can be competitive language models
- Introduced 4 techniques for tackling speed and overfitting
- Future work will extend to other discrete latent variable models

EOS

Citations

Generalized Softmax

- Softmax

$$p(z_t \mid z_{t-1}) = \frac{\exp(\mathbf{u}_{z_{t-1}}^\top \mathbf{v}_{z_t})}{\sum_z \exp(\mathbf{u}_{z_{t-1}}^\top \mathbf{v}_z)}$$

- Generalized Softmax

$$p(z_t \mid z_{t-1}) = \frac{K(\mathbf{u}, \mathbf{v})}{\sum_z K(\mathbf{u}, \mathbf{v}_z)} = \frac{\phi(\mathbf{u})^\top \phi(\mathbf{v})}{\sum_z \phi(\mathbf{u})^\top \phi(\mathbf{v}_z)},$$

for positive kernel $K : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}_+$ and feature map $\phi : \mathbb{R}^D \to \mathbb{R}^R$

Generalized Softmax: Inference

- The key $O(Z^2)$ step in the forward algorithm:

$$p(z_t \mid x_{<t}) = \sum_{z_{t-1}} p(z_t \mid z_{t-1}) p(z_{t-1} \mid x_{<t})$$

- In matrix form,

$$\boldsymbol{\gamma}_t = \underbrace{\boldsymbol{\alpha}_{t-1}}_{\mathbb{R}^Z} \underbrace{\Lambda}_{\mathbb{R}^{Z \times Z}},$$

  where we have the probability of the

current state,	$[\boldsymbol{\gamma}_t]_{z_t} = p(z_t \mid x_{<t})$,
last state,	$[\boldsymbol{\alpha}_{t-1}]_{z_{t-1}} = p(z_{t-1} \mid x_{<t})$,
transition probability,	$[\Lambda]_{z_{t-1}, z_t} = p(z_t \mid z_{t-1})$

Generalized Softmax: Inference

- Use generalized softmax in transition distribution

$$[\Lambda]_{z_{t-1},z_t} = p(z_t \mid z_{t-1}) \propto \phi(\mathbf{u}_{z_{t-1}})^\top \phi(\mathbf{v}_{z_t})$$

- Allows us to apply associative property of matrix multiplication

$$\begin{aligned}
\boldsymbol{\gamma}_t &= \boldsymbol{\alpha}_{t-1}\Lambda \\
&= \boldsymbol{\alpha}_{t-1}(\mathrm{diag}(d)\phi(U)\phi(V)^\top) \\
&= \underbrace{(\boldsymbol{\alpha}_{t-1} \circ d)}_{\mathbb{R}^Z}\underbrace{\phi(U)}_{\mathbb{R}^{Z\times f}}\underbrace{\phi(V)^\top}_{\mathbb{R}^{f\times Z}},
\end{aligned}$$

  with stacked embeddings $\phi(U), \phi(V) = [\phi(\mathbf{v}_1), \ldots, \phi(\mathbf{v}_Z)]$ and normalizing constants d

- Takes $O(Zf)$ time from left to right!