Linearized Structured Models

Anonymous Author(s)
Affiliation
Address
email

Abstract

Structured distributions, i.e. distributions over combinatorial spaces, are commonly used to learn latent probabilistic representations from observed data. However, scaling these models is bottlenecked by the high computational and memory complexity with respect to the size of the latent representations. Common models such as Hidden Markov Models (HMMs) and Probabilistic Context-Free Grammars (PCFGs) require time and space quadratic and cubic in the number of hidden states respectively. This work demonstrates a simple approach to reduce the computational and memory complexity of a large class of structured models. We show that by viewing the central inference step as a matrix-vector product and using a kernel linearization constraint, we can reduce the complexity of inference by a factor of the hidden state size, critical when scaling to large models. This method provides exact inference for a model approximation of standard structured distributions. Experiments with neural parameterized structured models for language modeling, polyphonic music modeling, unsupervised grammar induction, and video modeling show that our approach matches the accuracy of standard models at large state spaces while providing practical speedups.

1 Introduction

When modeling complex sequential spaces, such as sentences, musical scores, or video frames, a key choice is the internal structural representations of the model. A common choice in recent years is to use neural representations [Bengio et al., 2003, Mikolov et al., 2011, Brown et al., 2020, Boulanger-Lewandowski et al., 2012, Huang et al., 2018, Weissenborn et al., 2020] to store a deterministic history. These models yield strong predictive accuracy but their deterministic, continuous forms provide little insight into the intermediate decisions of the model.

Latent structured models provide an alternative approach where complex modeling decisions are broken down into a series of probabilistic steps. Structured models provide a principled framework for reasoning about the probabilistic dependencies between decisions and for computing posterior probabilities. The structure of the decision processes and the ability to answer queries through probabilistic inference afford interpretability and controllability that are lacking in neural models [Koller and Friedman, 2009, Levine, 2018].

Despite the benefits of structured models, the computational complexity of training scales asymptotically much worse than for neural models, as inference, and therefore training, requires marginalizing over all possible latent structures. For standard general-purpose models like Hidden Markov Models (HMM) and Probabilistic Context-Free Grammars (PCFG), the runtime of inference scales quadratically and cubically in the number of states respectively, which limits the ability to reach a massive scale. Promisingly, recent work has shown that in specific situations these models can be scaled, and that the increased scale results in commensurate improvements in accuracy – without sacrificing the ability to perform exact inference [Dedieu et al., 2019, Chiu and Rush, 2020, Yang et al., 2021].

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

In this work, we propose an approach for improving the runtime of a large class of structured latent models by introducing a linearization constraint. We target the family of models where inference can be formulated through a labeled directed hypergraph, which describes a broad class of dynamic-programming based inference [Klein and Manning, 2004, Huang and Chiang, 2005, Zhou et al., 2006, Javidian et al., 2020, Chiang and Riley, 2020]. We show how under specific kernel decomposition constraints these models allow for more efficient inference. Imposing a model constraint of linearizable relationships between variables allows for a key step of inference to be rewritten as a fast matrix-vector product. This approach is inspired by recent advances in computationally efficient neural attention attention [Katharopoulos et al., 2020, Peng et al., 2021, Choromanski et al., 2020], a significantly different task and formulation, that also uses linearization of (approximate) kernels to rewrite matrix-vector products as low-rank products.

We evaluate this approach by learning linearized structured models for tasks of language modeling, polyphonic music modeling, unsupervised grammar induction, and video modeling. For these tasks we use a variety of models including HMMs, PCFGs, and Hidden Semi-Markov Models (HSMMs). The application of linearization is nontrivial in high-dimensional structured models, and we demonstrate effective techniques for overcoming several practical challenges of linearized parameterizations. We find that our approach achieves very similar results to unconstrained models at large state sizes, while the decomposition allows us to greatly increase the speed of inference. Results on HMMs show that we can scale to more than 16,000 states; results on PCFGs achieve a significant perplexity reduction from much larger state spaces compared to past work [Kim et al., 2019]; and results on HSMMs show that our formulation enables scaling to much larger state spaces for continuous emissions [Fried et al., 2020].

2 Latent Structure and Hypergraphs

We consider the problem of modeling a sequence of observations $p(x) = p(x_1, \ldots, x_T)$. These observations can range in complexity from the words in a sentence to a series of co-occurring musical notes to features of video frames, and may be discrete or continuous. We assume these observations are generated by an unobserved (latent) structured representation z, and model the joint $p(x, z)$. The structure may be sequential or hierarchical, such as latent trees, and the set of structures $\mathcal{Z}$ is combinatorial, i.e. exponential in size with respect to the input sentence. In order to train these models on observations, we must optimize the evidence $p(x) = \sum_z p(x, z)$ by marginalizing over z. Scaling this marginalization will be the focus of this work.

Hypergraphs are a graphical model formalism for structured distributions that admit tractable inference through dynamic programming [Klein and Manning, 2004, Huang and Chiang, 2005, Zhou et al., 2006, Javidian et al., 2020, Chiang and Riley, 2020].[1] Define a labeled, directed, acyclic hypergraph as a set of nodes $\mathcal{V}$, a set of hyperedges $\mathcal{E}$, and a designated root node $S \in \mathcal{V}$. Each node $v \in \mathcal{V}$ has a collection of labels $\mathcal{L}_v$. Each hyperedge $e \in \mathcal{E}$ consists of a head node u and tuple of tail nodes, $v = (v_1, \ldots, v_{|e|})$, where $|e|$ is the number of tail nodes. For simplicity, we will assume *at most* 2 tail nodes v_1, v_2, and unless noted, a fixed label set $\mathcal{L}$ throughout. Each hyperedge e is associated with a score matrix $\Psi_e \in \mathbb{R}^{\mathcal{L} \times \mathcal{L}^{|e|}}$ with a score for all head and tail labels.[2] We use the notation $[\Psi_e]_{z_u,(z_1,z_2)}$ to indicate the score for head label z_u and tail labels z_1 and z_2. Finally, we assume we have a topological ordering over the edges.

A hypergraph is used to aggregate marginals bottom-up, which is done through a dynamic programming (belief propagation) algorithm. Algorithm 1(left) shows the algorithm. It works by filling in a table vector $\alpha_v \in \mathbb{R}^{\mathcal{L}}$ for each node v in order, and is initialized to 1 at the leaf nodes.[3] It returns the sum over latent structures, i.e. $p(x)$. Counting loops, the worst-case runtime complexity is $O(|\mathcal{E}| \times L^{|e^*|+1})$ where $L = |\mathcal{L}|$ is the size of the label set and $|e^*|$ the max hyperedge tail size. Algorithm 1(right) shows the same algorithm in matrix form by introducing joined tail vectors

[1] While the formalism is similar to undirected factor graphs, it allows us to represent more complex distributions: notably dependency structures with unknown topologies, such as latent trees.

[2] This formalism can represent inference in both locally and globally normalized models, although we focus on local normalization in this work.

[3] The accumulation of scores is denoted by $\overset{+}{\leftarrow}$. Multiple hyperedges can have the same head node, whose scores must be added together.

Algorithm 1 Hypergraph marginalization

[*Scalar Form*]
for $u \leftarrow v_1, v_2$ hyperedge e topologically **do**
  for $z_u \in \mathcal{L}_u$ **do**
    $[\alpha_u]_{z_u} \overset{+}{\leftarrow} \sum_{z_1,z_2} [\Psi_e]_{z_u,(z_1,z_2)} \cdot [\alpha_{v_1}]_{z_1} \, [\alpha_{v_2}]_{z_2}$
return $\sum_z [\alpha_S]_z$

[*Matrix Form*]
for $u \leftarrow v$ hyperedge e topologically **do**
  $\alpha_u \overset{+}{\leftarrow} \Psi_e \beta_v$
return $\alpha_S^\top \mathbf{1}$

Algorithm 2 Hypergraph marginalization for HMMs and PCFGs

[*HMM - Backward*]
for $t \leftarrow (t+1)$ in right-to-left order **do**
  for $z_{t+1} \in \mathcal{L}$ **do**
    $[\beta_{t+1}]_{z_{t+1}} = [\alpha_{t+1}]_{z_{t+1}}$
  $\alpha_t \overset{+}{\leftarrow} \Psi_t \beta_{t+1}$
return $\alpha_0^\top \mathbf{1}$

[*PCFG - CKY*]
for $(i,k) \leftarrow (i,j),(j,k)$ in span-size order **do**
  for $z_1, z_2 \in \mathcal{L}_{i,j} \times \mathcal{L}_{j,k}$ **do**
    $[\beta_{i,j,k}]_{(z_1,z_2)} = [\alpha_{i,j}]_{z_1} [\alpha_{j,k}]_{z_2}$
  $\alpha_{i,k} \overset{+}{\leftarrow} \Psi \beta_{i,j,k}$
return $\alpha_{1,T}^\top \mathbf{1}$

$\beta_v \in \mathbb{R}^{\mathcal{L}^{|e|}}$ for each group of nodes v. Letting $z_v = (z_1, z_2)$, the joined tail vector contains entries $[\beta]_{z_v} = [\alpha_{v_1}]_{z_1} [\alpha_{v_2}]_{z_2}$.

To make this formalism more concrete, we show how hypergraphs can be used for inference in several structured generative models: hidden Markov models, probabilistic context-free grammars, and hidden semi-Markov models. These examples are instances of the hypergraph algorithm.

Example: Hidden Markov Models HMMs are discrete latent sequence models defined by the following generative process: first, a sequence of discrete latent states $z = (z_1, \ldots, z_T)$ with state size L are sampled as a Markov chain. Then each state z_t independently emits an observation x_t, i.e.

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

where $p(z_t \mid z_{t-1})$ is the transition distribution, $p(x_t \mid z_t)$ the emission distribution, and $p(z_1 \mid z_0)$ is the initial distribution with distinguished start symbol z_0.

Given a sequence of observations $x = (x_1, \ldots, x_n)$ we can compute $p(x) = \sum_z p(x, z)$ using a labeled directed hypergraph, with single-tailed edges, nodes corresponding to state positions, labels corresponding to states, and emissions probabilities incorporated into the scoring matrices Ψ. There are T scoring matrices, $\Psi_t \in \mathbb{R}^{\mathcal{L} \times \mathcal{L}}$, with entries $[\Psi_{T-t}]_{z_t, z_{t+1}} = p(z_{t+1}, x_t \mid z_t)$ corresponding to transitions. Algorithm 2 (left) shows the approach. This requires time $O(TL^2)$ and is identical to the backward algorithm for HMMs.[4]

Example: Context-Free Grammars Context-free grammars (CFGs) are a structured model defined by the 5-tuple $\mathcal{G} = (S, \mathcal{N}, \mathcal{P}, \mathcal{X}, \mathcal{R})$, where S is the distinguished start symbol, $\mathcal{N}$ is a set of nonterminals, $\mathcal{P}$ is a set of preterminals, $\mathcal{X}$ is the token types in the vocabulary, and $\mathcal{R}$ is a set of grammar rules. Production rules for start, nonterminals, and preterminals take the following forms:[5]

$$S \to A, \quad A \in \mathcal{N}; \quad A \to B\,C, \quad B, C \in \mathcal{N} \cup \mathcal{P}; \quad D \to x, \quad D \in \mathcal{P}, x \in \mathcal{X}. \tag{2}$$

A probabilistic context-free grammar (PCFG) additionally has a probability measure on the set of rules. To compute $p(x_1, \ldots, x_T)$ with a hypergraph, we create one node for each contiguous subspan $[i, k)$ in the sentence. Nodes with $i + 1 < k$ have a nonterminal label set $\mathcal{L} = \mathcal{N}$. Nodes with $i + 1 = k$ have a preterminal label set $\mathcal{L}_{i,i+1} = \mathcal{P}$. The main scoring matrix is $\Psi \in \mathbb{R}^{\mathcal{L} \times \mathcal{L}^2}$, with entries $[\Psi]_{z_u,(z_1,z_2)} = p(z_1, z_2 \mid z_u)$.[6] Algorithm 2 (right) shows how for every hyperedge we join the scores from the two tail nodes in $\alpha_{i,j}$ and $\alpha_{j,k}$ into joined tail vector $\beta_{i,j,k} \in \mathbb{R}^{\mathcal{L}^2}$. As there are $O(T^3)$ hyperedges and the largest $\mathcal{L}$ is of size $|\mathcal{N}|$, the runtime of the algorithm is $O(T^3 |\mathcal{N}|^3)$. This approach is identical to the CKY algorithm.

[4] In the case of HMMs, the table vectors α_t correspond to the backward algorithm's β values.

[5] We restrict our attention to grammars in Chomsky normal form.

[6] We have a separate matrix for terminal production on x which we elide for simplicity.

Example: Hidden Semi-Markov Models HSMMs are extensions of HMMs that allow for generating a variable-length sequence of observations per state. It defines the following generate process: first, we sample a sequence of discrete latent states $z = (z_1, \cdots, z_K)$ with a first-order Markov model. We then use them to generate the length of observations per state. For our experiments we generate independent continuous emissions x_t with a Gaussian distribution for $p(x_i \mid z_k)$. Full details of the inference procedure are given in Appendix B.

3 Linearizing Structured Models

For these structured distributions, hypergraphs provide a general method for inference (and therefore training parameterized versions). However, the underlying algorithms scale poorly with the size of the label sets (quadratic for HMM and HSMM, cubic for CFG). This complexity makes it challenging to scale these models and train versions with very large numbers of states.

In this section, we consider an approach for improving the scalability of these models by reducing the dependence of the computational complexity of inference on the label set size. The basic idea is to speed up the matrix-vector product step in inference by using a low-rank decomposition of Ψ. This is accomplished by imposing a constraint on the scoring functions which determine the entries of Ψ. In the next section we show that this constraint can easily be incorporated into parameterized versions of these models.

3.1 Kernelizing Hyperedges

The main bottleneck for inference speed is the matrix-vector product $\alpha_u = \Psi_e \beta_v$ that must be computed for every edge in the hypergraph. As we saw in Algorithm 1 (left), this step takes time $L^{|e|+1}$ to compute. However, the matrix form Algorithm 1 (right) hints that there is an alternative way to view this computation as a kernel.

Recall that a kernel is a symmetric function $K : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}_+$ for dimensionality D, and that the linearization [Aizerman et al., 1964, Jäkel et al., 2007] of a kernel is its representation as an inner product $K(a, b) = \phi(a)^\top \phi(b)$ for a feature map $\phi : \mathbb{R}^D \to \mathbb{R}^N$, where N is the feature dimension determined by the kernel. All kernels admit a linearization, but with potentially infinite N.

We note the following elementary property of matrix-vector products. If the entries of Ψ can be represented as a kernel, then it can be linearized based on head and tail labels, i.e.

$$[\Psi]_{z_u,(z_1,z_2)} = c_{z_u} c_{z_1,z_2} \phi(f(z_u))^\top \phi(g(z_1, z_2)),$$

where $f : \mathcal{L}_u \to \mathbb{R}^D$ and $g : \mathcal{L}_{v_1} \times \mathcal{L}_{v_2} \to \mathbb{R}^D$ are representation functions, and the c's are scalars (e.g. normalizing constants). This implies Ψ can be decomposed into the head factor $U \in \mathbb{R}^{\mathcal{L} \times N}$ and tail factor $V \in \mathbb{R}^{\mathcal{L}^{|e|} \times N}$. This leads to the following matrix-vector product expression for each hyperedge e with $u \leftarrow v$:

$$\Psi_e \beta_v = \left(U_e V_e^\top\right) \beta_v = U_e \left(V_e^\top \beta_v\right), \tag{3}$$

where $[U_e]_{z_u,n} = c_{z_u}[\phi(f(x_u))]_n$ and $[V_e]_{z_v,n} = c_v[\phi(g(z_1, z_2))]_n$, with α_u and β_v.

This rearranging allows for shared computation and removes a factor of the head labels from the computation entirely. This optimization leads to a runtime complexity of $O(|\mathcal{E}| \times L^{|e^*|} \times N)$.Algorithm 3 shows this explicitly. In practice, this can be a major change. For instance, HMMs and HSMMs go from scaling quadratically in the number of hidden states to linearly (with a new factor of the feature dimension).

Unfortunately, we cannot turn an arbitrary hypergraph into a convenient kernelized form. For one, there is no guarantee that scores will be symmetric. Furthermore, even if Ψ forms a kernel, many common kernels do not admit a small N-dimensional feature mapping, and therefore preclude a low-rank factorization for speeding up inference. To achieve a decomposition for fast inference, we will need to modify the parameterization of the model.

For example, the most common parameterization of a conditional probability distribution, i.e. $p(z_1, z_2 \mid z_u)$, is a softmax parameterization. Softmax assumes that local transitions can be computed using logistic regression or more complex neural networks.

Algorithm 3 Linearized marginalization

for $u \leftarrow v_1, v_2$ hyperedge e topologically **do**
  for $n \in 1, \ldots, N$ **do**
    $[\gamma]_n = \sum_{z_v} c_v \, [\phi(g(z_1, z_2))]_n \, [\beta_v]_{z_v}$ $\quad \triangleright \, O(L^{|e|})$
  $\alpha_u \overset{+}{\leftarrow} U_e \gamma$ $\quad \triangleright \, O(LN)$
return $\alpha_S^\top \mathbf{1}$

A common symmetric form of softmax parameterization is to utilize label embeddings. We can construct local conditional distributions by taking the softmax of the D-dimensional dot-product between the vector representation (embeddings) of the given label z_u and the generated labels z_1, z_2 [He et al., 2017]:

$$p(z_1, z_2 \mid z_u) \propto \exp\left(f(z_u)^\top g(z_1, z_2)\right) = K_{\mathrm{SM}}(f(z_u), g(z_1, z_2)).$$

This gives a kernel (symmetric positive), which can be used to form Ψ, known as a normalized exponential kernel [Tsai et al., 2019]. However, the exponential kernel does not afford speedups as it does not admit a finite dimension N linearization [Steinwart et al., 2006].[7]

3.2 Linearized Parameterization

Given the challenges of kernelizing softmax, we instead consider explicitly constructing classes of finite-dimensional linearizable kernels K with associated feature mapping $\phi : \mathbb{R}^D \to \mathbb{R}^N$:

$$p(z_1, z_2 \mid z_u) \propto K(f(z_u), g(z_1, z_2)) = \phi(f(z_u))^\top \phi(g(z_1, z_2)).$$

Since we use kernels to construct probability mass functions, we restrict our attention to kernels that have both strictly positive and low-dimensional feature mappings $\phi : \mathbb{R}^D \to \mathbb{R}_+^N$ [Choromanski et al., 2020]. This immediately ensures that the resulting scoring matrix Ψ admits a low-rank decomposition, as in Eqn. 3.

In practice, we find the choice of kernel has a large effect on performance. As enforcing a low-rank factorization in every conditional distribution limits the expressivity of a model, we explicitly target distributions that are involved in computational bottlenecks. For these key distributions, we use an efficiently linearizable kernel, denoted by K, allowing factorization of the scoring matrix. For other computations, we utilize a standard softmax kernel K_{SM} and do not factorize the resulting scoring matrix. We refer to this as a mixed parameterization.

Hidden Markov Models For Linearized HMMs (LHMMs) we use the following mixed kernel parameterization, which specifically targets the state-state transition bottleneck:

$$p(z_1 \mid z_0) \propto K(\mathbf{u}_{z_0}, \mathbf{v}_{z_1}), \quad p(z_t \mid z_{t-1}) \propto K(\mathbf{u}_{z_{t-1}}, \mathbf{v}_{z_t}), \quad p(x_t \mid z_t) \propto K_{\mathrm{SM}}(\mathbf{u}_{z_t}, \mathbf{v}_{x_t}), \tag{4}$$

where $\mathbf{u}_z$ is the embedding of z when z is used as head, $\mathbf{v}_z$ its embedding when used as tail. This allows Ψ to be decomposed as the $p(x_t \mid z_t)$ terms become constants. These embeddings come from neural network parameterizations detailed in Appendix C.

This allows inference to be run in time $O(TLN)$, where T is the length of a sequence, L the size of the label space, and N the feature dimension.

Hidden Semi-Markov Models Linearized HSMM (LHSMM) uses a similar kernel parameterization:

$$p(z_1 \mid z_0) \propto K(\mathbf{u}_{z_0}, \mathbf{v}_{z_1}), \quad p(z_k \mid z_{k-1}) \propto K(\mathbf{u}_{z_{k-1}}, \mathbf{v}_{z_k}), \quad p(x_t \mid z_k) \propto K_{\text{Gauss}}(\mathbf{u}_{z_k}, \mathbf{x}_t), \tag{5}$$

where $K_{\text{Gauss}}(\cdot, \cdot)$ is the Gaussian kernel used to model continuous $\mathbf{x}_t$. The total inference complexity is $O(TLMN)$, where M is the maximum length of the observation sequence under any state.

[7] While the exponential kernel does not afford an exact linearization, the kernel can be approximated deterministically [Cotter et al., 2011] or stochastically [Rahimi and Recht, 2007]. The stochastic approximation is a key component of recent work on efficient attention [Choromanski et al., 2020, Peng et al., 2021].

Context-Free Grammars The PCFG uses a similar mixed parameterization to the HMM. These probabilities correspond to start ($S \to A$), preterminal ($D \to x$), and standard productions ($A \to B\ C$) respectively.

$$p(z_{1,N} \mid S) \propto K_{\mathrm{SM}}(\mathbf{u}_S, \mathbf{u}_{z_{1,N}}), \quad p(z_{i,j}, z_{j,k} \mid z_{i,k}) \propto \begin{cases} K_{\mathrm{SM}}(\mathbf{u}_{z_{i,k}}, \mathbf{v}_{z_{i,j}\ z_{j,k}}) & \substack{i+1=j \vee \\ j+1=k} \\ K(\mathbf{u}'_{z_{i,k}}, \mathbf{v}_{z_{i,j}\ z_{j,k}}) & \text{o.w.} \end{cases} \tag{6}$$
$$p(x_i \mid z_i) \propto K_{\mathrm{SM}}(\mathbf{u}_{z_i}, \mathbf{v}_{x_i}),$$

where $\mathbf{u}_z$/$\mathbf{u}'_z$ is the embedding of z when z is used as head,$\mathbf{v}_x$/$\mathbf{v}_{z_1,z_2}$ is the embedding of x/(z_1, z_2) when they are used as tail. See Appendix C for the full parameterization, drawn from Kim et al. [2019]. Note that we limit the use of linearization to nonterminal to nonterminal productions. These productions dominate the runtime as they are applied at $O(T^3)$ hyperedges.

This allows inference to be run in time $O(T^3 L^2 N)$, where T is the length of a sequence, L the size of the label space, and N the feature dimension.

4 Experimental Setup

We evaluate the linearization approach with four experiments: sequential language modeling with HMMs, polyphonic music modeling with a large observation space, hierarchical language modelings with PCFGs, and video modeling with HSMMs.

Data Our first set of experiments evaluate sequential models on PENN TREEBANK dataset (PTB) [Marcus et al., 1993] for the task of word-level language modeling. We use the preprocessing from Mikolov et al. [2011]. The second set of experiments is on polyphonic music modeling [Boulanger-Lewandowski et al., 2012]. We evaluate on four music datasets: Nottingham (Nott), Piano, MuseData (Muse), and JSB chorales (JSB). Each timestep consists of an 88-dimensional binary vector indicating whether a particular note is played. Since multiple notes may be played at the same time, the effective vocabulary size is extremely large. The third set of experiments use PCFGs for language modeling, we also use PTB, but with the splits and preprocessing used in unsupervised constituency parsing [Shen et al., 2018, 2019, Kim et al., 2019]. The last set of experiments use HSMMs for video modeling, where we use CROSSTASK [Zhukov et al., 2019] with 10% of the training data for validation. We follow the preprocessing steps in Fried et al. [2020] and apply PCA to project features to vectors of size 200. For the full data details, please see Appendix A.

Models and Hyperparameters For language modeling with HMMs, we experiment with a range of state sizes, $|\mathcal{L}| = L \in \{2^{10}, 2^{11}, 2^{12}, 2^{13}, 2^{14}\}$, and rank $N \in \{L/2, L/4, L/8\}$. For polyphonic music modeling with HMMs, we experiment with states sizes $L \in \{2^7, 2^8, 2^9, 2^{10}, 2^{11}\}$. For language modeling with PCFGs, we use a set of nonterminals of size $|\mathcal{N}| \in \{30, 60, 100\}$ and preterminals of twice the number of nonterminals $|\mathcal{P}| = 2|\mathcal{N}|$. Our smallest setting ($|\mathcal{N}| = 30$, $|\mathcal{P}| = 60$) is the one used in Kim et al. [2019]. For video modeling with HSMMs, we use the same model setting as Fried et al. [2020], but we don't constrain states to the predefined states per task, and we experiment with state sizes $L \in \{2^6, 2^7, 2^8, 2^9, 2^{10}\}$ and rank $N \in \{2^4, 2^5, 2^6, 2^7\}$.

We utilize the feature map $\phi(x) = \exp(Wx)$ for the linearized HMM and HSMM, and $\phi(x) = \exp(Wx - \|x\|_2^2/2)$ for the PCFG. We initialize the parameters of feature maps using orthogonal feature projections [Choromanski et al., 2020], and update it alongside the model parameters. For the full hyperparameter and optimization details, see Appendix D.

Baselines and Evaluation The language modeling experiments evaluate using perplexity. Baselines are neurally parameterized HMM with a standard softmax transition. We also compare to VL-HMM, which makes a strong structural sparsity assumption on the emission distribution [Chiu and Rush, 2020]. We include for reference a state-of-the-art language model, the AWD-LSTM [Merity et al., 2017]. For polyphonic music modeling, we compare our LHMM against RNN-NADE [Boulanger-Lewandowski et al., 2012] which models the full joint distribution of notes as well as temporal dependencies; as well as autoregressive neural models such as the R-Transformer [Wang et al., 2019] (as reported by Song et al. [2019]) and an LSTM (as reported by Ziegler and Rush [2019]); models with latent continuous dynamics such as the LV-RNN [Gu et al., 2015] and SRNN [Fraccaro et al., 2016]; and finally comparable models with latent discrete dynamics, the TSBN [Gan et al., 2015] and

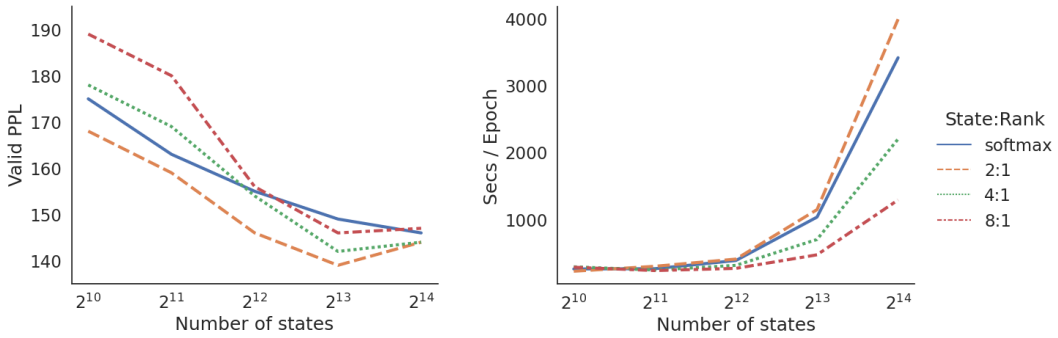


Figure 1: Validation perplexities on PTB versus model scale, as well as speed in seconds per epoch.

Model	Val	Test
AWD-LSTM	60.0	57.3
VL-HMM	128.6	119.5
HMM	144.3	136.8
LHMM	141.4	131.8

Model	$L : N$	Train	Val
HMM	-	95.9	144.3
LHMM	8	97.5	141.4
LHMM+band	8	101.1	143.8
LHMM	16	110.6	146.3
LHMM+band	16	96.9	138.8
LHMM	32	108.4	153.7
LHMM+band	32	110.7	145.0

Table 1: Model perplexities on PTB. All HMM variants have $L = 2^{14}$ states. (Left): Validation and test perplexities. The LHMM has a state-to-rank ratio $8 : 1$. (Right): Further experiments with extending the low-rank structure of LHMMs with a banded transition structure.

the baseline HMM. We evaluate perplexities of our Linearized PCFG (LPCFG) against a softmax PCFG (PCFG) [Kim et al., 2019]. For video modeling, we evaluate the negative log likelihoods on the test set and compare linearized HSMMs to vanilla HSMMs.

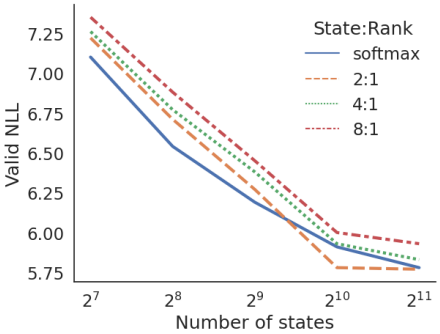
5 Results

Hidden Markov Models for Language Modeling Our main experimental result is that the linearized models produce similar results as our baselines. Fig. 1 shows that perplexity improves as we increase the scale of the HMM, and that the performance of our LHMM also improves at the same rate. At small sizes, the linearization slightly hinders performance; however once the size is large enough, i.e. larger than 2^{12}, LHMMs with 8:1 state-to-rank ratios perform comparably. [8]

Fig. 1 also contains speed comparisons between HMMs and LHMMs. A state-to-rank ratio of 8:1 matches the performance of softmax HMMs at larger state sizes and also gives an empirical speedup of more than 3x at $L = 2^{14}$. As expected, we only see a speedup when the state-to-rank ratio exceeds 2:1, as we replaced the $O(L^2)$ operation with two $O(LN)$ ones. This implies that linearization is most effective with scale, where we observe large computational gains at no cost in perplexity.

All our HMMs are outperformed by neural models, and also by VL-HMMs [Chiu and Rush, 2020] which offer similar modeling advantages, shown in Tbl. 1 (left). This indicates that some aspects of performance are not strictly tied to scale. We posit this is due to the problem-specific block-sparse emission constraint in VL-HMMs. While very effective for language modeling, the VL-HMM relies on a hard clustering of states for emission. This is difficult to apply to problems with richer emission models (as in music and video modeling).

[8] One explanation is that while the rank constraint results in underfitting at smaller state sizes, at larger state sizes the transition matrix of the softmax HMMs is already lower rank due to implicit rank regularization [Gunasekar et al., 2017, Arora et al., 2019, Razin and Cohen, 2020], resulting in no loss of performance with a rank constraint in the corresponding LHMM. See Appendix E for an analysis of the ranks of HMMs/LHMMs.



Model	Nott	Piano	Muse	JSB
RNN-NADE	2.31	**7.05**	**5.6**	5.19
R-Transformer	**2.24**	7.44	7.00	8.26
LSTM	3.43	7.77	7.23	8.17
LV-RNN	2.72	7.61	6.89	**3.99**
SRNN	2.94	8.20	6.28	4.74
TSBN	3.67	7.89	6.81	7.48
HMM	2.43	8.51	7.34	5.74
LHMM	2.60	8.89	7.60	5.80

Figure 2: Polyphonic music negative log-likelihoods (NLL), measured in nats. (Left): HMM and LHMM validation performance for various state sizes and state:rank ratios. (Right): Test-set NLLs for polyphonic music. The HMM models have $\mathcal{L} = 2^{11}$ states and the LHMM has rank $N = 2^9$, a 4:1 state:rank ratio.

Hidden Markov Models for Music Modeling We next apply LHMMs on polyphonic music modeling. This has a max effective vocabulary size of 2^{88}, as multiple notes may occur simultaneously. Unlike for language modeling, we use a factored Bernoulli emission model, modeling the presence of each note independently. Fig. 2 (right) shows that HMMs are competitive with many of the models on these datasets, including LSTMs. We find that LHMMs achieve performance slightly worse than but comparable to the unconstrained HMMs overall. Fig. 2 (left) shows that the distinction drops with more states. Both HMMs achieve low negative likelihoods (NLL) on the datasets with shorter sequences, Nottingham and JSB, but relatively poorer NLLs on the datasets with longer sequences (Muse and Piano).

Context-Free Grammars For syntactic language modeling on PTB, our linearized PCFG (LPCFG) achieves similar performance to PCFGs, as shown in Table 2 (left), with an improvement in computational complexity. The PCFG models have a complexity cubic in the number of nonterminals, so even models with $|\mathcal{N}| = 30$ nonterminals are relatively costly. Our approach achieves comparable results with $N = 8$ features. As we scale up the number of nonterminals to $|\mathcal{N}| = 100$, LPCFG stays competitive with a lower computational complexity (since $N < |\mathcal{N}|$). These experiments also demonstrate the importance of scale in syntactic language models with more than 50 point gain in perplexity over a strong starting model.

Hidden Semi-Markov Models for Video Modeling Table 2 (right) shows the results of video modeling using HSMMs. In addition to using a different hypergraph, these experiments use a continuous Gaussian emission model. By removing the state constraints from tasks, our HSMM baselines get better video-level NLLs than that from Fried et al. [2020] at the cost of more memory consumption. Due to GPU memory constraints, we can only train HSMMs up to 2^8 states, but by using our linearized parameterization, we can scale to up to 2^{10} states which yields an improved NLL. Absolute results could likely be improved with more states and by an improved emission parameterization for all models.

Improving Rank Assumptions One potential limitation of all linearized models is that they cannot learn high-rank structures with low N. We began to see this issue at a ratio of 16:1 states to features for large models. To explore the effects of this limitation, we perform an additional experiment that combines low-rank features with a sparse component. Specifically we add an efficient high-rank sparse banded transition matrix. Full details are in Appendix F. Tbl. 1 (right) shows that combination with the band structure allows for larger ratios than just the low-rank structure alone, while only adding another operation that costs $O(LN)$.

6 Related Work

Similar to our work, other approaches target matrix or tensor operations in inference, and impose structural model constraints to improve computational complexity. Many of the works on HMMs in particular take advantage of the transition structure. The Dense-mostly-constant (DMC) HMM

$\|\mathcal{N}\|$	$\|\mathcal{P}\|$	Model	N	PPL
30	60	PCFG	-	252.60
		LPCFG	8	247.02
		LPCFG	16	250.59
60	120	PCFG	-	234.01
		LPCFG	16	217.24
		LPCFG	32	213.81
100	200	PCFG	-	191.08
		LPCFG	32	203.47
		LPCFG	64	194.25

Model	L	N	NLL
Fried et al. [2020]	$\leq$151	-	1.432×10^5
HSMM	2^6	-	1.428×10^5
HSMM	2^7	-	1.427×10^5
HSMM	2^8	-	1.426×10^5
LHSMM	2^7	2^7	1.427×10^5
LHSMM	2^8	2^6	1.426×10^5
LHSMM	2^9	2^5	1.424×10^5
LHSMM	2^{10}	2^4	1.423×10^5

Table 2: (Left): Test perplexities of PCFG models on PTB. The complexity of PCFG is $O(T^3|\mathcal{N}|^3)$, whereas the complexity of LPCFG is $O(T^3|\mathcal{N}|^2N)$. (Right): Negative log likelihoods (NLL) per video of HSMM models on CROSSTASK. We cannot train HSMMs beyond 2^8 states due to GPU memory constraints, but we can train LHSMMs with up to 2^{10} states.

assigns a subset of learnable parameters per row of the transition matrix and sets the rest to a constant, leading to a sub-quadratic runtime [Siddiqi and Moore, 2005]. Other structures have also been explored, such as aligning the states of an HMM to underlying phenomena that allows inference to be sped up [Felzenszwalb et al., 2004, Roweis, 2000]. Additionally, other methods take advantage of emission structure in HMMs in order to scale, such as the Cloned HMM [Dedieu et al., 2019] and VL-HMM [Chiu and Rush, 2020]. Compared to these approaches, our method is more flexible and generic, since it can be applied in a non-application-specific manner, and even extended with high-rank components (such as banded structure).

Low-rank structure has been explored in both HMMs [Siddiqi et al., 2009] and a generalization of PCFGs called weighted tree automata [Rabusseau et al., 2015]. The reduced-rank HMM [Siddiqi et al., 2009] has at most 50 states, and relies on spectral methods for training. The low-rank weighted tree automata [Rabusseau et al., 2015] also trains latent tree models via spectral methods. We extend the low-rank assumption to neural parameterizations, which have been shown to be effective for generalization [Kim et al., 2019, Chiu and Rush, 2020], and directly optimize the evidence via gradient descent.

Concurrent work in unsupervised parsing uses a tensor decomposition to scale PCFGs to large state spaces [Yang et al., 2021]. Our low-rank decomposition of the flattened head-tail scoring matrix is more general, resulting in worse scaling for the PCFG setting but with wider applicability, as shown by experiments with HMMs.

Kernels have been used in non-parametric belief propagation for graphical models [Song et al., 2011a, 2010, 2011b], with the goal of performing approximate inference in complicated, non-conjugate models. We utilize kernels in the context of standard models, then scale them to massive sizes.

7 Conclusion

This work improves the scaling of structured models by establishing the effectiveness of linearization for hypergraph models. We show that kernel decompositions allow us to rewrite a key step of inference in structured models through a low-rank factorization of the scoring matrix, obtained through the use of an efficiently linearizable kernel. Linearized inference allows us to obtain a reduction in the asymptotic complexity of marginalization at the cost of a constrained model. Our approach applies to a wide class of models, including HMMs, HSMMs, and PCFGs. Through our experiments on language, video, and polyphonic music modeling, we demonstrate an effective approach for overcoming the practical difficulty of applying our kernel constraints in high dimensional, structured spaces by targeting and linearizing model components that bottleneck computation. Future work includes exploration of other structural constraints for speeding up matrix-vector products [Dao et al., 2020] performed in inference, as well as application to models where exact inference is intractable.

References

M. A. Aizerman, E. A. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. In *Automation and Remote Control,*, number 25, pages 821–837, 1964.

Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization. *CoRR*, abs/1905.13655, 2019. URL `http://arxiv.org/abs/1905.13655`.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null):1137–1155, March 2003. ISSN 1532-4435.

Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, page 1881–1888, Madison, WI, USA, 2012. Omnipress. ISBN 9781450312851.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

David Chiang and Darcey Riley. Factor graph grammars. *arXiv preprint arXiv:2010.12048*, 2020.

Justin Chiu and Alexander Rush. Scaling hidden Markov language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1341–1349, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.103. URL `https://www.aclweb.org/anthology/2020.emnlp-main.103`.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2020.

Andrew Cotter, Joseph Keshet, and Nathan Srebro. Explicit approximations of the gaussian kernel. *CoRR*, abs/1109.4603, 2011. URL `http://arxiv.org/abs/1109.4603`.

Tri Dao, Nimit Sohoni, Albert Gu, Matthew Eichhorn, Amit Blonder, Megan Leszczynski, Atri Rudra, and Christopher Ré. Kaleidoscope: An efficient, learnable representation for all structured linear maps. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=BkgrBgSYDS`.

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

Pedro Felzenszwalb, Daniel Huttenlocher, and Jon Kleinberg. Fast algorithms for large-state-space hmms with applications to web usage analysis. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems*, volume 16. MIT Press, 2004. URL `https://proceedings.neurips.cc/paper/2003/file/9407c826d8e3c07ad37cb2d13d1cb641-Paper.pdf`.

Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers, 2016.

Daniel Fried, Jean-Baptiste Alayrac, Phil Blunsom, Chris Dyer, Stephen Clark, and Aida Nematzadeh. Learning to segment actions from observation and narration. *arXiv preprint arXiv:2005.03684*, 2020.

Zhe Gan, Chunyuan Li, Ricardo Henao, David Carlson, and Lawrence Carin. Deep temporal sigmoid belief networks for sequence modeling, 2015.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

Shixiang Gu, Zoubin Ghahramani, and Richard E. Turner. Neural adaptive sequential monte carlo, 2015.

Suriya Gunasekar, Blake Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nathan Srebro. Implicit regularization in matrix factorization, 2017.

Junxian He, Zhiting Hu, Taylor Berg-Kirkpatrick, Ying Huang, and Eric P Xing. Efficient correlated topic modeling with topic embedding. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 225–233, 2017.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, and Douglas Eck. An improved relative self-attention mechanism for transformer with application to music generation. *CoRR*, abs/1809.04281, 2018. URL `http://arxiv.org/abs/1809.04281`.

Liang Huang and David Chiang. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64, 2005.

F. Jäkel, B. Schölkopf, and FA. Wichmann. A tutorial on kernel methods for categorization. *Journal of Mathematical Psychology*, 51(6):343–358, December 2007.

Mohammad Ali Javidian, Zhiyu Wang, Linyuan Lu, and Marco Valtorta. On a hypergraph probabilistic graphical model. *Annals of Mathematics and Artificial Intelligence*, 88(9):1003–1033, 2020.

A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

Yoon Kim, Chris Dyer, and Alexander Rush. Compound probabilistic context-free grammars for grammar induction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2369–2385, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1228. URL `https://www.aclweb.org/anthology/P19-1228`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Dan Klein and Christopher D Manning. Parsing and hypergraphs. In *New developments in parsing technology*, pages 351–372. Springer, 2004.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. ISBN 0262013193.

Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review, 2018.

Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017. URL `http://arxiv.org/abs/1711.05101`.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, June 1993. ISSN 0891-2017.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. *CoRR*, abs/1708.02182, 2017. URL `http://arxiv.org/abs/1708.02182`.

Tomas Mikolov, Anoop Deoras, Stefan Kombrink, Lukás Burget, and Jan Cernocký. Empirical evaluation and combination of advanced language modeling techniques. pages 605–608, 2011. URL `http://dblp.uni-trier.de/db/conf/interspeech/interspeech2011.html#MikolovDKBC11`.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong. Random feature attention. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=QtTKTdVrFBB.

Guillaume Rabusseau, Borja Balle, and Shay B. Cohen. Weighted tree automata approximation by singular value truncation. *CoRR*, abs/1511.01442, 2015. URL http://arxiv.org/abs/1511.01442.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NIPS*, pages 1177–1184, 2007. URL https://people.eecs.berkeley.edu/~brecht/papers/07.rah.rec.nips.pdf.

Noam Razin and Nadav Cohen. Implicit regularization in deep learning may not be explainable by norms. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21174–21187. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/f21e255f89e0f258accbe4e984eef486-Paper.pdf.

Sam Roweis. Constrained hidden markov models. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 2000. URL https://proceedings.neurips.cc/paper/1999/file/84c6494d30851c63a55cdb8cb047fadd-Paper.pdf.

Yikang Shen, Zhouhan Lin, Chin wei Huang, and Aaron Courville. Neural language modeling by jointly learning syntax and lexicon. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rkgOLb-0W.

Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. Ordered neurons: Integrating tree structures into recurrent neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1l6qiR5F7.

Sajid M. Siddiqi and Andrew W. Moore. Fast inference and learning in large-state-space hmms. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, page 800–807, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595931805. doi: 10.1145/1102351.1102452. URL https://doi-org.proxy.library.cornell.edu/10.1145/1102351.1102452.

Sajid M. Siddiqi, Byron Boots, and Geoffrey J. Gordon. Reduced-rank hidden markov models. *CoRR*, abs/0910.0902, 2009. URL http://arxiv.org/abs/0910.0902.

Kyungwoo Song, JoonHo Jang, Seungjae Shin, and Il-Chul Moon. Bivariate beta LSTM. *CoRR*, abs/1905.10521, 2019. URL http://arxiv.org/abs/1905.10521.

Le Song, Byron Boots, Sajid M. Siddiqi, Geoffrey Gordon, and Alex Smola. Hilbert space embeddings of hidden markov models. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 991–998, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Le Song, Arthur Gretton, Danny Bickson, Yucheng Low, and Carlos Guestrin. Kernel belief propagation, 2011a.

Le Song, Eric Xing, and Ankur Parikh. Kernel embeddings of latent tree graphical models. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24, pages 2708–2716. Curran Associates, Inc., 2011b. URL https://proceedings.neurips.cc/paper/2011/file/dc09c97fd73d7a324bdbfe7c79525f64-Paper.pdf.

Ingo Steinwart, Don Hush, and Clint Scovel. An explicit description of the reproducing kernel hilbert spaces of gaussian rbf kernels. *IEEE Transactions on Information Theory*, 52(10):4635–4643, 2006.

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4344–4353, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1443. URL `https://www.aclweb.org/anthology/D19-1443`.

Zhiwei Wang, Yao Ma, Zitao Liu, and Jiliang Tang. R-transformer: Recurrent neural network enhanced transformer, 2019.

Dirk Weissenborn, Oscar Täckström, and Jakob Uszkoreit. Scaling autoregressive video models. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rJgsskrFwH`.

Songlin Yang, Yanpeng Zhao, and Kewei Tu. Pcfgs can do better: Inducing probabilistic context-free grammars with many symbols, 2021.

Shun-Zheng Yu. Hidden semi-markov models. *Artificial intelligence*, 174(2):215–243, 2010.

Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. *Advances in neural information processing systems*, 19:1601–1608, 2006.

Dimitri Zhukov, Jean-Baptiste Alayrac, Ramazan Gokberk Cinbis, David Fouhey, Ivan Laptev, and Josef Sivic. Cross-task weakly supervised learning from instructional videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3537–3545, 2019.

Zachary M. Ziegler and Alexander M. Rush. Latent normalizing flows for discrete sequences, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] , we demonstrate a 3x speedup in large HMMs via kernelization in Fig. 1.
   (b) Did you describe the limitations of your work? [Yes] , we mention the limitation of the low-rank constraint in Section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] , see Appendix I.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] , we will include code in the supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] , see Section 4 and Appendix A and D.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] , we will include error bars in future revisions.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] , see Appendix D.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] we cite the datasets and models.

(b) Did you mention the license of the assets? [Yes] , we include the license in the code with the supplemental material.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] , we include code in the supplemental material.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] , the data for the Penn Treebank and polyphonic music are publicly available.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] , we are not aware of the Penn Treebank containing personally identifiable information or offensive content, and the music datasets are not applicable.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]