Scaling Structured Inference with Randomization

Yao Fu 1  Mirella Lapata 1

Abstract

The scale of the state space of discrete graphical models is crucial for model capacity in the era of deep learning. Existing dynamic programming (DP) based inference typically works with a small number of states (usually less than hundreds). In this work, we propose a family of randomized dynamic programming (RDP) algorithms for scaling structured models to tens of thousands of latent states. Our method is widely applicable to classical DP-based inference (partition, marginal, reparameterization, entropy, etc) and different graph structures (chains, trees, and more general hypergraphs). It is also compatible with automatic differentiation so can be integrated with neural networks seamlessly and learned with gradient-based optimizers. Our core technique is randomization, which is to restrict and reweight DP on a small selected subset of nodes, leading to computation reduction by orders of magnitudes. We further achieve low bias and variance with Rao-Blackwellization and importance sampling. Experiments on different inferences over different graphs demonstrate the accuracy and efficiency of our methods. Furthermore, when using RDP to train a scaled structured VAE, it outperforms baselines in terms of test likelihood and successfully prevents posterior collapse.

1. Introduction

Deep structured models (Wiseman et al., 2018; Martins et al., 2019; Paulus et al., 2020) for language have enjoyed inspiring progress recently (Dozat & Manning, 2016; Zhang et al., 2020), yet their capacity is limited by issues of scaling (Sun et al., 2019; Chiu & Rush, 2020; Yang et al., 2021). Traditional dynamic programming based inference for exponential families has limited scalability with large combinatorial spaces. When integrating exponential families with neural networks (e.g., a VAE with a structured inference network), the limited latent space capacity contrast sharply to the modeling capacity of modern neural networks, especially with pretrained language models (Li et al., 2020). Existing work has already observed improved performance by scaling certain types of latent structures (Yang et al., 2021; Li et al., 2020; Chiu & Rush, 2020), and researchers are eager to know if we could have general techniques for scaling structured models.

Challenges for scaling structured models primarily come from memory complexity. For example, consider linear-chain CRFs (Sutton & McCallum, 2006), a popular structured model family that uses dynamic programming (the Forward algorithm) for exact inference. This algorithm requires \( O(TN^2) \) computation where \( N \) is the number of latent states and \( T \) is the length of the sequence. It is precisely the \( N^2 \) term that is problematic in terms of memory consumption. This limitation is more severe under automatic differentiation (AD) frameworks as all DP computations are stored for gradient construction. Consequently, when the number of latent states increases, memory overflow usually happens when calling back-propagation. Generally, DP-based inference algorithms are not optimized for modern computational devices like GPUs and typically work under small-data regimes, with \( N \) in the range \([10, 100]\) (Ma & Hovy, 2016; Wiseman et al., 2018). With larger \( N \), inference becomes intractable since gradients do not easily fit into GPU memory (Sun et al., 2019).

This work proposes a randomization framework for scaling structured models, which induces a family of randomized dynamic programming algorithms with a wide coverage of different structures and inference (Table 1). Within our randomization framework, instead of summing over all possible combinations of latent states, we only sum over paths with most probable states, and sample a subset of less likely paths to correct the bias according to a reasonable proposal. Since we only calculate the chosen paths, memory consumption can be reduced to a small controllable budget which is scale-invariant. We thus recast the memory complexity challenge into a tradeoff between memory budget, proposal accuracy, and estimation error. In practice, we show RDP scales existing models by two orders of magnitude with memory complexity as small as one percent.

---

1Institute for Language, Cognition and Computation, University of Edinburgh, Edinburgh, United Kingdom. Correspondence to: Yao Fu <yao.fu@ed.ac.uk>, Mirella Lapata <mlap@inf.ed.ac.uk>.

Preprint.
In addition to the significantly increased scale, we highlight the following advantages of RDP: (1). applicability to different structures (chains, trees, and hypergraphs) and inference operations (partition, marginal, reparameterization, and entropy). (2). compatibility with automatic differentiation and existing efficient libraries (like Torch-struct in Rush, 2020). (3). statistically principled controllability on bias and variance. These advantages largely overcome the limitation of existing methods. As a concrete application, we show that RDP can be used for learning a structured VAE with a scaled inference network. In experiments, we first demonstrate that RDP algorithms estimate partition and entropy for chains and hypertrees with lower mean square errors than baselines. Then we show their joint effectiveness for learning the scaled VAE. RDP outperforms baselines in terms of test likelihood and successfully prevents posterior collapse.

### 2. Problem Statement and Background

We start with an example structured VAE for language. Let \( y = [y_1, \ldots, y_T] \) be an observed sentence as a sequence of words. Let \( x \) be any latent structure (sequences of latent tags, parse trees, or general latent graphs) that generates \( y \). Then we consider the following structured VAE model:

\[
\mathcal{L} = \mathbb{E}_{q_\theta(x|y)}[p_\psi(x, y)] + \mathcal{H}(q_\theta(x|y)) \tag{1}
\]

where the inference model takes the form of a discrete exponential family (e.g., linear-chain CRFs in Fu et al., 2020 or probabilistic context-free grammars in Yang et al., 2021). This formulation and its variants are widely used in deep structured prediction models. Examples include chains (Mensch & Blondel, 2018), semi-Markov (Li & Rush, 2020; Wiseman et al., 2018), constituency trees (Kim et al., 2019), dependency trees (Corro & Titov, 2018) and so on.

To make gradient-based learning possible, there are a few requirements about the tractability of the inference model \( q_\theta \): (a). likelihood evaluation, which requires the computation of the partition function \( Z \); (b). sampling and reparameterization, which allows Monte-Carlo gradients for the inference model parameters (Mohamed et al., 2020); (c). entropy calculation, which is important for regularization (Alemi et al., 2018). The existing dynamic programming toolbox (Table 1) provides exact solutions to the above requirements, but only works for small latent space (e.g., a linear-chain CRF with the number of states smaller than 100). Since the computational complexity of DP-based inference is usually at least quadratic to the size of the latent states, it would give us memory overflow if we want to scale it to tens of thousands (e.g., the size of CCG supertags in Steedman, 2000 or lexicalized PCFGs in Zhu et al., 2020).

Scaling the above inference algorithms is particularly challenging under automatic differentiation frameworks. At the first sight, one may think the memory requirement for some algorithms is not so large. For example, the memory complexity of batchified Forward algorithm is \( O(BTN^2) \). Consider batch size \( B = 10 \), sentence length \( T = 100 \), number of states \( N = 1000 \), then the complexity is \( O(10^{13}) \), which seems to be acceptable, let alone the existence of multiple implementation tricks. However, this is not the case under automatic differentiation. The AD framework not only invalidates tricks like dropping intermediate variables (because all computation should be stored), but also multiplies the memory complexity when building the adjacent gradient graph. The situation is more severe if we want to compute higher-order gradients, or if the underlying DP has higher-order complexity (e.g., the Inside algorithm with \( O(T^2N^3) \) complexity). Furthermore, many efficient libraries heavily integrate or rely on existing AD libraries (like Torch-struct Rush, 2020). These are the reasons that we should work within AD frameworks.

Although there exists scaling techniques for structured models, many of them have intrinsic limitations. In addition to the above AD-compatibility restrictions, many existing techniques either make many stringent assumptions (e.g., sparsity in Lavergne et al., 2010; Sokolovska et al., 2010; Correa et al., 2020 or pre-clustering in Chiu & Rush, 2020), rely on handcrafted heuristics for bias correction (Jeong et al., 2009), or cannot be easily adapted to modern GPUs with tensorization and parallelization (Klein & Manning, 2003). The range of applicable models of existing methods is also limited: Chiu & Rush (2020) only consider chains and Yang et al. (2021) only consider PCFGs. The most promising technique is from Sun et al. (2019), yet they only consider the most probable states thus being severely biased. Our randomized DP is inspired by the top K summation approach in Sun et al. (2019) and the randomization approach in Oktay et al. (2020). RDP covers a large spectrum of graph structures and inference operations while being compatible.
Scaling Structured Inference with Randomization

Figure 1. Scaling inference by randomization. (A): randomized summation (Eq.2). (B): Randomized Forward (Alg. 1) recursively applies randomized summation at each DP step. Memory reduction is achieved by restricting computation on edges linking the sampled nodes. Second-order randomized DPs (Entropy and Reparameterization, Alg. 3 and 4) reuse this graph and share the same path as the gradients. (C): Doubly-Randomized Inside (Alg. 2) applies randomized summation twice at each DP step. All algorithms in our randomized DP family are compatible to automatic differentiation, and their gradient direction are show by the red dashed arrows.

3. Scaling Inference with Randomization

3.1. Preliminary: Randomized Summation

The foundation of our randomized DP lays in a simple setting: speeding up summation by randomization. Consider the sum of a sorted list \( a \) of positive numbers: \( S = \sum_{i=1}^{N} a_i \). This requires \( N-1 \) additions, which could be expensive when \( N \) is large. Suppose we would like to reduce the number of summands to \( K \), we have the following sum-and-sample estimator:

\[
\hat{S} = \sum_{i=1}^{K_1} a_i + \frac{1}{K_2} \sum_{j=K_1+1}^{K_2} a_j \delta_j \tag{2}
\]

where \( q \) is a proposal distribution upon the tail summands \([a_{K_1+1}, \ldots, a_N]\). The oracle proposal \( q^* \) is the normalized tail summands: \( q^*_i = a_i / \sum_{j=K_1+1}^{N} a_j \), under which the estimate becomes exact: \( \hat{S} = S \). Essentially, it sums over the first top \( K_1 \) summands, then gets a \( K_2 \)-sized sample from the tail summands and correct the bias by dividing the corresponding probability. This estimator is also discussed in (Kool et al., 2020) for gradient estimation. In our case, we will adjust it to hypergraph structures, and recursively use it in DP-based inference.

Bias and Variance. One can show that this estimator is unbiased: \( E_q[\hat{S}] = S \), irrespective of how we choose the proposal \( q \). Thus variance can be reduced by: (1). increasing \( K_1 \) (top summands), as the effect of Rao-Blackwellization (Liu et al., 2019). This would be particularly effective when the underlying summands are long-tailed and when \( K_1 \) is at least the number of modes (Fig. 1 A); (2), increasing \( K_2 \) (number of sample), as the effect of central limit theorem; (3), reducing the gap to the oracle proposal \( \epsilon_i = |q_i - q^*_i| \), as the effect of importance sampling, which can be achieved by constructing a correlated proposal based on our knowledge about the summands. When we are not confident about our knowledge of \( a \), we can always retreat to a uniform proposal, under which the variance is correlated to the entropy of the oracle distribution (since entropy measures how the oracle diverges from uniform).

3.2. Framework: Randomized Sum-Product

Sum-Product Recap. Now we show our general recipe of scaling DP-based inference by recursively using estimator Eq. 2 in DP. Given a discrete exponential family in standard overparameterization (Wainwright & Jordan, 2008):

\[
p_\theta(X) \propto \exp\left\{ \sum_{s \in V} \phi_s(x_s) + \sum_{(s,t) \in E} \phi_{st}(x_s, x_t) \right\} \tag{4}
\]

where \( X = [X_1, \ldots, X_M] \) is a random vector as nodes in a graphical model with each node takes discrete values \( X_t \in \{1, 2, \ldots, N\} \). \( N \) is the number of states, \( \phi_{st} \) is the edge potential, \( \phi_s \) is the node potential. Here we use a general notation of nodes \( V \) and edges \( E \). We discuss specific structures later. Suppose we want to compute its marginals and log partition. A first solution is the sum-product algorithm that recursively updates the message at each edge:

\[
\mu_{ts}(x_s) \propto \sum_{x'_t=1}^{N} \left\{ \phi_{st}(x_s, x'_t) \phi_t(x'_t) \prod_{u \in V^*_t \setminus t} \mu_{ut}(x'_t) \right\} \tag{5}
\]

where \( \mu_{ts}(x_s) \) denotes the message from node \( t \) to node \( s \) evaluated at \( X_s = x_s \), \( V^*_t \) denotes the set of neighbor
nodes of $t$ except $s$. Upon convergence, it gives us the Bethe approximation of the partition and marginals. When the underlying graph is a tree, this approach becomes exact. The complexity is quadratic to $N$, which makes it hard to scale under AD.

**Randomized Sum-Product Principle.** Our solution for scaling is to recursively use the random sum estimator in Eq. 2 to approximate the message in Eq. 5. Specifically, for each node $X_t$, given a pre-constructed proposal $q_t = [q_t(1), ..., q_t(N)]$ (we discuss how to construct a correlated proposal later), we retrieve the top $K_1$ index from $q_t$, and get a $K_2$-sized sample from the rest of $q_t$:

\[
[σ_{t,1}, ..., σ_{t,K_1}, ..., σ_{t,N}] \leftarrow \text{arg sort}\{q_t(i)\}_{i=1}^N
\]

\[
δ_{s,j} \sim \text{i.i.d. Categorical}\{q_t(σ_{t,K_1+1}), ..., q_t(σ_{t,N})\}
\]

\[
Ω_t^{K_1} = \{σ_{t,i}\}_{i=1}^{K_1} \quad Ω_t^{K_2} = \{δ_{t,j}\}_{j=1}^{K_2}
\]  

(8)

plugging Eq. 2 for each node $t$, we substitute the full index $\{1, ..., N\}$ with the the top $Ω_t^{K_1}$ and the sampled index $Ω_t^{K_2}$:

\[
\hat{μ}_{ts}(x_s) \propto \sum_{σ_{t,i} ∈ Ω_t^{K_1}} \left\{ φ_{st}(x_s, σ_t) φ_t(σ_t) \prod_{u∈V_t} \hat{μ}_{ut}(σ_t) \right\} + \sum_{δ_{t,j} ∈ Ω_t^{K_2}} \left\{ \frac{1}{K_{2q_t}} φ_{st}(x_s, δ_t) φ_t(δ_t) \prod_{u∈V_t} \hat{μ}_{ut}(δ_t) \right\}
\]

(9)

where the oracle proposal is proportional to the actual summands (Eq. 5). Note that the proposal and index are precomputed and outside the DP, thus are not involved in automatic differentiation. Arguments about bias and variance root from the arguments for the vanilla sum estimator (Eq. 2). Below we instantiate this general randomized message passing principle on common graphs.

3.3. First-Order Randomized DP

We consider two typical structures: chains and tree-structured hypergraphs. Exponential families defined on these structures are widely used for structured prediction tasks like sequence tagging and dependency parsing (Eisner, 2016). The chain structure covers models like HMMs and linear-chain CRFs. Tree-structured hypergraphs cover models like PCFGs and Dependency CRFs. We study partition function estimation when $N$ is large. The conventional message passing then becomes the Forward algorithm for chains, and the Inside algorithm for tree-structured hypergraphs.

**Randomized Forward.** Algorithm 1 shows the randomized forward algorithm for approximating the partition function of chain-structured graphs. The core recursion in Eq. 10 estimates the alpha variable $\hat{α}_t(i)$ as the sum of all possible sequences up to step $t$ at state $i$. It corresponds to the general Eq. 9 applied to chains (Fig. 1B). We can recover the classical Forward algorithm by changing the chosen index set $Ω_t^{K_2}$ to the full index $\{1, ..., N\}$.

**Bias.** Note the chain-structured recursion as the current step $\hat{α}_t$ depend on the previous step $\hat{α}_{t-1}$, which indicates the proof of unbiasedness by induction:

\[
E[\hat{α}_{t-1}] = α_{t-1} \Rightarrow E[\hat{α}_t] = α_t
\]

(22)

where the unbiasedness of the first step $E[\hat{α}_2] = α_2$ can be easily proved, similar to Eq. 2. Setting $t = T$, we get the unbiasedness of the estimated partition: $E[Z] = Z$. When implementing on log semirings (Rush, 2020), the estimate becomes a lower bound due to Jensen’s inequality.
Algorithm 3 Randomized Entropy DP

**Input:** potentials \( \phi(x_{t-1}, x_t, y_t) \), top \( K_1 \) index set \( \Omega_{t+1}^{K_1} \), sampled \( K_2 \) index set \( \Omega_{t+2}^{K_2} \), estimated \( \hat{Z}, \hat{\alpha} \) from the Randomized Forward

**Initialize** \( H_t(i) = 0 \).

for \( t = 1 \) to \( T - 1 \) do

Compute recursion:

\[
\hat{p}_t(i, j) = \phi(i, j, x_t) \hat{\alpha}_t(i) / \hat{\alpha}_{t+1}(j)
\]

\[
H_{t+1}(j) = \sum_{\sigma \in \Omega_{t+1}^{K_1}} \hat{p}_t(\sigma) [H_t(\sigma) - \log \hat{p}_t(\sigma)] + \sum_{\delta \in \Omega_{t+2}^{K_2}} \frac{1}{K_{2q}(\delta)} \hat{p}_t(\delta) [H_t(\delta) - \log \hat{p}_t(\delta)]
\]

end for

Return \( \hat{H} \)

\[
\hat{p}_T(i) = \hat{\alpha}_T(i) / \hat{Z}
\]

\[
\hat{H} = \sum_{\sigma \in \Omega_{t+1}^{K_1}} \hat{p}_T(\sigma) [H_T(\sigma) - \log \hat{p}_T(\sigma)] + \sum_{\delta \in \Omega_{t+2}^{K_2}} \frac{1}{K_{2q}(\delta)} \hat{p}_T(\delta) [H_T(\delta) - \log \hat{p}_T(\delta)]
\]

Variance. Similar to the variance arguments of Eq 2, we reduce the variance of the randomized forward by: (1) Rao-Blackwellization: increasing \( K_1 \) to cover modes of the summands; (2) importance sampling: to construct a proposal correlated to the actual summands. So the variance comes from the gap between the proposal and the oracle (only accessible with full DP), as is shown in the green bars in Fig 1B. Variance is also closely related to how long-tailed the underlying distribution is: the longer the tail, the more effective Rao-Blackwellization will be. The more flat the tail is, the lower variance a uniform proposal will have.

Efficiency. Essentially, the Sampled Forward restricts the DP computation from full graph to the subgraph with chosen nodes (\( \Omega_{t}^{K_1} \) and \( \Omega_{t+1}^{K_2} \) for all \( t \)), quadratically reducing the memory complexity from \( O(TN^2) \) to \( O(TK^2) \).

Compatibility with Automatic Differentiation. Since all computation in Alg. 1 are differentiable, one could directly compute gradients of the estimated partition with any AD library, thus enabling gradient-based optimization. As a useful side product, when gradients are computed with regard to the edge potential, we get the marginal probability of the sampled edges (as the standard property of exponential family, see Wainwright & Jordan, 2008):

\[
\hat{\mu}(x_{t-1}, x_t) = \nabla_\phi(x_{t-1}, x_t, y_t) \hat{Z}
\]

where back-propagation shares the same DP graph as the Randomized Forward, but reverses its direction (Fig. 1B).

Algorithm 4 Randomized Gumbel Backward Sampling

**Input:** potentials \( \phi(x_{t-1}, x_t, y_t) \), top \( K_1 \) index set \( \Omega_{t+1}^{K_1} \), sampled \( K_2 \) index set \( \Omega_{t+2}^{K_2} \), estimated \( \hat{Z}, \hat{\alpha} \) from the Randomized Forward

**Initialize:**

\[
\hat{p}_T(i) = \hat{\alpha}_T(i) / \hat{Z}
\]

\[
\hat{x}_T = \text{softmax}(\log \hat{p}_T(i) + y_T(i))
\]

for \( t = T - 1 \) to \( 1 \) do

Compute recursion:

\[
\hat{p}_t(i, j) = \phi(i, j, y_t) \hat{\alpha}_t(i) / \hat{\alpha}_{t+1}(j)
\]

\[
\hat{x}_t = \text{softmax}(\log \hat{p}_t(i, x_{t+1}) + y_t(i))
\]

end for

Return relaxed sample \( \{\hat{x}_t\}_{t=1}^T \), hard sample \( \{\hat{x}_t\}_{t=1}^T \)

Doubly-Randomized Inside. Now we turn our focus to the randomized inside algorithm for approximating the partition function of tree-structured hypergraphs (Alg. 2), which is widely used in natural language parsing. This algorithm recursively estimates the inside variables \( \hat{\alpha}(i, j, k) \) which sums over all possible tree branchings (index \( m \) in Eq. 11) and chosen state combinations (index \( \sigma_1, \sigma_2, \delta_1, \delta_2 \), Fig 1C). Index \( i, j \) denotes a span from location \( i \) to \( j \) in a given sequence, and \( k \) denotes the state. Different than the Forward case, this algorithm is doubly-randomized in a way that it computes a product of the two randomized summations that represent two subtrees, i.e., \( (i, m) \) and \( (m+1, j) \).

The proposal is constructed for each subtree. This Doubly-randomization makes the bias correction tricky, as we need to divide the proposal probability and \( K_2 \) twice.

Efficiency, AD-Compatibility, Bias and Variance. The Randomized Inside restricts the computation on the sampled states of subtrees, reducing the complexity from \( O(T^2N^3) \) to \( O(T^2K^3) \). The output partition is again differentiable. The gradients reverse the computation graph and equal to the marginals. Unbiasedness can be proved by the following tree-structured induction:

\[
\mathbb{E}[\hat{\alpha}(i, m)] = \alpha(i, m) \wedge \mathbb{E}[\hat{\alpha}(m+1, j)] = \alpha(m+1, j)
\]

\[
\rightarrow \mathbb{E}[\hat{\alpha}(i, j)] = \alpha(i, j)
\]

When implemented in the log space, the estimate becomes a lower bound with twice application of Jensen’s Inequality. The variance, again, can be reduced by: (a) increasing \( K_1 \) to cover the number of modes (Rao-Blackwellization); (b) constructing a correlated proposal (importance sampling). In Fig. 1C, green bars represent gaps to the oracle proposal.
which is a major source of estimation error.

3.4. Second-Order Randomized DP

Now we generalize RDP to more inference operations. We consider entropy estimation and reparameterized sampling, both are widely used in modern deep generative models (recall the example in Sec. 2). Generally, second-order RDPs reuse the computational graph and intermediate outputs of first-order RDPs, and are further connected to the gradients of first-order RDPs. We focus on chain structures (HMMs and linear-chain CRFs) for simplicity.

**Randomized Entropy DP.** Algorithm 3 shows the randomized entropy DP. It reuses the chosen index \( \Omega_{t}^{K} \) and the intermediate variables \( \hat{H}_{t} \) and recursively estimates the conditional entropies \( \hat{H}_{t}(j) \) which represents the entropy of the chain ending at step \( t \), state \( j \). Unbiasedness can be achieved by the following induction:

\[
E[\hat{H}_{t}] = H_{t} \rightarrow E[\hat{H}_{t+1}] = H_{t+1} \tag{25}
\]

**Connections to First-Order Gradients.** It is important to note that the entropy DP computation shares the same path as the gradient of the log-partition, making the entropy computationally comparable to first-order gradients. This property makes RDP compatible to existing implementations that redefine entropy as gradients, like Rush (2020). Consequently, the gradients of entropy would be computationally comparable to second-order gradients, which would induce further engineering challenges if without AD. This again supports the necessity of the compatibility with AD-engines since they really simplify the implementation.

**Randomized Gumbel Backward Sampling.** Now we discuss another second-order RDP, namely the randomized Gumbel backward sampling (Alg. 4). This algorithm is based on the recently proposed Gumbel-FBFS algorithm for reparameterized gradient estimation of CRFs (Fu et al., 2020), and generalizes it to CRFs with tens of thousands of states. It reuses the DP paths of the randomized Forward and recursively computes the hard sample \( \hat{x}_{t} \) and soft sample \( \tilde{x}_{t} \) (a relaxed one-hot vector) based on the chosen index \( \Omega_{t}^{K} \). When using these soft sample for training structured VAEs, they induce biased but low-variance reparameterized gradients, as is demonstrated by many previous work (Fu et al., 2020; Paulus et al., 2020). We will use this algorithm for training our structured VAE example.

4. Composing Scaled DP with Neural Networks: A Structured VAE Example

We study a concrete example that put together the above introduced RDP algorithms. Recall the VAE framework discussed in Sec. 3. Let \( x = [x_{1}, \ldots, x_{T}] \) be a sequence of discrete latent states, and \( y = [y_{1}, \ldots, y_{T}] \) a sequence of observed words. We consider an autoregressive generative model parameterized by an LSTM decoder:

\[
p(x, y) = \prod_{t} p(x_{t}|x_{<t}, y_{<t}) \cdot p(y_{t}|x_{t}, x_{<t}, y_{<t})
\]

\[
p(x_{t}|x_{<t}, y_{<t}) = \text{softmax}(\text{MLP}(\text{LSTM}(x_{t}, y_{<t})))
\]

\[
p(y_{t}|x_{t}, x_{<t}, y_{<t}) = \text{softmax}(\text{MLP}(\text{LSTM}(x_{1:t}, y_{<t})))
\]

Since the posterior is intractable, we use an structured inference model parameterized by a CRF with a neural encoder:

\[
q(x|y) = \prod_{t} \Phi(x_{t-1}, x_{t}) \phi(x_{t}, y_{t}) / Z \tag{26}
\]

where \( \Phi \) is the \( N \times N \) transition potential matrix and \( \phi \) is the emission potential. This formulation and its tree variants are also used in multiple previous work (Ammar et al., 2014; Cai et al., 2017; Li & Rush, 2020; Kim et al., 2019). Yet previous work only consider small \( N \). When \( N \) is large, parameterization and inference require careful treatment. Specifically, we firstly use a neural encoder to map \( y \) to a set of contextualized representations:

\[
[r_{1}, \ldots, r_{T}] = \text{Enc}(y_{1}, \ldots, y_{t}) \tag{27}
\]

As direct parameterization of an \( N \times N \) transition matrix would be memory expensive when \( N \) is large, we decompose it by associating each state \( i \) with a state embedding \( e_{i} \). Then the potentials are:

\[
\Phi(x_{t-1}, x_{t}) = e_{x_{t-1}}^{T} e_{x_{t}} \quad \phi(x_{t}, y_{t}) = e_{x_{t}}^{T} r_{t} \tag{28}
\]

For training, we use the Randomized Gumbel Backward sampling for reparameterized gradient estimation. Recall that it calls the Randomized Forward as a subroutine. Entropy is also estimated by our RDP, as the regularization of the inference network.

**Proposal Construction.** we consider the following:

\[
q_{t}(i) = \frac{\phi(i, y_{t})}{2 \sum_{j=1}^{N} \phi(j, y_{t})} + \frac{||e_{i}||_{1}}{\sum_{j=1}^{N} ||e_{j}||_{1}} \tag{29}
\]

which consists of: (1). local weights, as the first term is the normalized local emission. The intuition is that states with larger local weights should be more likely to be observed. (2). global prior, as the second term represents. The intuition is that larger norms are usually associated to large dot products, thus larger potentials. This proposal exploits the common long-tail property of a scaled model: only few states are most probable (which is handled by the top \( K_{1} \) summation), and the rest are less likely (which is handled by the \( K_{2} \) tail sample).

**Exploration-Exploitation Tradeoff.** It is important to note that the gradients only pass through the top \( K_{1} \) index \( \Omega_{t}^{K_{1}} \) and tail sample \( \Omega_{t}^{K_{2}} \) during training. This means
5. Experiments

We first evaluate the estimation error of individual RDP algorithms for three unit cases, namely when the underlying distributions are long-tail, intermediate level, or dense. Then we evaluate RDP for training the structured VAE.

5.1. Evaluation of RDP Algorithms

**Setting.** We evaluate the mean square error (MSE) for the estimation of linear-chain log partition (Alg. 1), hypertree log partition (Alg. 2), and linear-chain entropy (Alg. 3) on the three types of distributions. We simulate the distributions by controlling their entropy: the smaller the entropy, the more long-tailed the underlying distribution is. We set \( N \), the number of states, to be 2000 and 10000, which are orders of magnitudes larger than previous work (Wiseman et al., 2018; Yang et al., 2021). We run RDPs 100 times and calculate their MSE against the exact results from the full DP. For all estimators, we set \( K_2 = 1 \) and \( K_1 = K - 1 \), and control \( K \) to be \([1, 10, 20]\) percent of \( N \). For linear-chains, we use the proposal discussed in Eq. 29. For hypertrees, we use a uniform proposal.

**Baselines.** Since there are limited methods that are applicable to this range of structures and inference, we choose the topK summation method in Sun et al. (2019) as our baseline. This method was originally developed only for linear-chain partition approximation, and can be viewed as setting \( K_2 = 0 \) in our case (topK summation only, no sampling). This method is also used in Correia et al. (2020), yet they only consider sparse underlying distributions. Here it is important to differentiate long-tail vs. sparse distributions: while they both mean that head elements take a large portion of probability, sparse means tail elements have no probability while long-tail means tail elements have non-negligible probability. This entails that the topK approach in Sun et al. (2019); Correia et al. (2020) may significantly underestimate the partition and entropy.

**Results.** Table 2 shows the MSE results. Our method outperforms the topK baseline approach on all distributions with significantly less memory budgets. We highlight two important observations: (1). since the TopK summation approach requires sparsity, it consistently underestimate the partition and entropy, particularly on dense distributions, while RDP does not have such limitation; (2). by choosing

| \( N = 2000 \) | Linear-chain Log Partition | Hypertree Log Partition | Linear-chain Entropy |
|---------------|---------------------------|-------------------------|----------------------|
| TopK 20% \( N \) | 3.874, 1.015, 0.162 | 36.127, 27.435, 21.783 | 443.7, 84.35, 8.011 |
| TopK 50% \( N \) | 0.990, 0.251, 0.031 | 2.842, 2.404, 2.047 | 131.8, 22.100, 1.816 |
| RDP 1% \( N \) (ours) | 0.146, 0.066, 0.076 | 26.331, 37.669, 48.863 | 5.925, 1.989, 0.691 |
| RDP 10% \( N \) (ours) | 0.067, 0.033, 0.055 | 1.193, 1.530, 1.384 | 2.116, 1.298, 0.316 |
| RDP 20% \( N \) (ours) | **0.046, 0.020, 0.026** | **0.445, 0.544, 0.599** | **1.326, 0.730, 0.207** |

| \( N = 10000 \) | Linear-chain Log Partition | Hypertree Log Partition | Linear-chain Entropy |
|---------------|---------------------------|-------------------------|----------------------|
| TopK 20% \( N \) | 6.395, 6.995, 6.381 | 78.632, 63.762, 43.556 | 227.36, 171.97, 141.91 |
| TopK 50% \( N \) | 2.134, 2.013, 1.647 | 35.929, 26.677, 17.099 | 85.063, 59.877, 46.853 |
| RDP 1% \( N \) (ours) | 0.078, 0.616, 0.734 | 3.376, 5.012, 7.256 | 6.450, 6.379, 4.150 |
| RDP 10% \( N \) (ours) | 0.024, 0.031, 0.024 | 0.299, 0.447, 0.576 | 0.513, 1.539, 0.275 |
| RDP 20% \( N \) (ours) | **0.004, 0.003, 0.003** | **0.148, 0.246, 0.294** | **0.144, 0.080, 0.068** |

Table 2. Mean square error comparison between RDP algorithms v.s. TopK approximation. D = Dense, I = intermediate, L = Long-tailed distributions. Our method outperforms the baseline on all unit cases with significantly less memory.

Table 3. Bias-Variance decomposition of Randomized Forward.

| \( N = 2000 \) | Linear-chain Log Partition | Hypertree Log Partition | Linear-chain Entropy |
|---------------|---------------------------|-------------------------|----------------------|
| TopK 20% \( N \) | 3.874, 1.015, 0.162 | 36.127, 27.435, 21.783 | 443.7, 84.35, 8.011 |
| TopK 50% \( N \) | 0.990, 0.251, 0.031 | 2.842, 2.404, 2.047 | 131.8, 22.100, 1.816 |
| RDP 1% \( N \) (ours) | 0.146, 0.066, 0.076 | 26.331, 37.669, 48.863 | 5.925, 1.989, 0.691 |
| RDP 10% \( N \) (ours) | 0.067, 0.033, 0.055 | 1.193, 1.530, 1.384 | 2.116, 1.298, 0.316 |
| RDP 20% \( N \) (ours) | **0.046, 0.020, 0.026** | **0.445, 0.544, 0.599** | **1.326, 0.730, 0.207** |

Table 4. Comparison of proposal on Randomized Forward.
Figure 2. Modulating the posterior by controlling $K_1/K_2$. The topK only approach leads to a collapsed posterior, which is overcome by the $K_2$ randomization. Increasing $K_2$ encourages exploring less certain states, and consequently leads to an increasingly heavier tail.

Table 5. Negative Log Likelihood comparison.

|                | Dev NLL   | Test NLL   |
|----------------|-----------|------------|
| FULL-100       | 39.64 ± 0.06 | 39.71 ± 0.07 |
| TopK-100       | 39.71 ± 0.13 | 39.76 ± 0.11 |
| RDP-100 (ours) | **39.59 ± 0.10** | **39.59 ± 0.08** |
| TopK-2000 (ours) | 39.81 ± 0.30 | 39.84 ± 0.31 |
| RDP-2000 (ours) | **39.47 ± 0.11** | **39.48 ± 0.14** |

only 1 percent of the full index, we are able to outperform TopK summation in many cases and get decent estimates.

Controlling Bias-Variance with Computation Budgets. Table 3 shows detailed bias-variance decomposition of the Randomized Forward algorithm. The implementation on the log semiring has its pros and cons: (1). the downside is that it makes the estimate become a lower bound (biased) due to Jensen’s inequality. (2). the upside is that it trivially reduces the variance to the log scale, and improves numerical stability. By increasing the computation budget $K$, we are able to simultaneously reduce bias and variance.

Improving MSE with Correlated Proposals. Table 4 shows how a correlated proposal (as introduced in Eq. 29) can improve estimates of the Randomized Forward. Generally, scaled models usually require certain types of decomposition (like the transition decomposition in Eq. 28). Proposals are closely related to specific parameterization. For different structures, one can follow the local-global principle to construct proposals, or retreat to a uniform proposal.

5.2. Evaluation of the Structured VAE

Setting. Now we use RDP to scale the latent space of the structured VAE. This experiment primarily tests the performance of the Randomized Gumbel Backward Sampling algorithm (Alg. 4), which is used as a reparameterized gradient estimator for training the VAE. We use an LSTM with 256 dimensional hidden states for the generative model. For the inference model, we use a pretrained GPT2 (base size) to show our methods’ compatibility with pretrained language models. We follow Fu et al. (2020) and use the MSCOCO dataset and reuse their processed data for simplicity. Our setting scales the original Gumbel-CRF paper (Fu et al., 2020). Specifically, we set $N = [100, 2000]$. With $N = 100$ we are able to perform full DP. Note that the full DP still gives biased gradients due to the continuous relaxation. With $N = 2000$, full DP gives memory overflow on a 16G GPU, so we only compare to the TopK approach. As the standard practice, we report negative log likelihood estimated by importance sampling with 500 sample (Kim et al., 2019).

Improving Likelihood with RDP Scaling. Table 5 shows the performance of LVM training. We observe that (1). RDP outperforms the baselines for both $N = 100$ and 2000, which demonstrates its effectiveness of training VAEs. (2). the overall NLL decreases as $N$ increase from 100 to 2000, which shows the advantage of scaling.

Modulating Posterior with $K_1/K_2$ Ratio. We further show that the ratio of $K_1/K_2$, which is interpreted as the exploration-exploitation tradeoff, can be used for modulating the posterior and prevent posterior collapse. Specifically, we draw the frequency of states of the aggregated posterior. Recall that the aggregated posterior is obtained by sampling the latent states from the inference network for all training instances, then calculating the overall frequency (Mathieu et al., 2019). Figure 2 shows that the topK summation baseline leads to the posterior collapse as their exists multiple inactive states (frequency = 0). Increasing $K_2$ encourages the exploration of the tails states during training, which consequently leads to the increased tail frequency of the aggregated posterior (after convergence).

6. Conclusion

In this work, we propose the randomized dynamic programming algorithm family for scaling classical structured prediction models. Our method scales classical algorithms to tens of thousands of latent states while reducing the computation complexity by orders of magnitudes. Our method significantly overcomes the limitation of previous approaches by being (1). widely applicable to a spectrum of structures and inference, and (2). compatible with automatic differentiation and modern neural networks. We hope this method will open new possibilities of large-scale deep structured prediction models.
References

Alemi, A., Poole, B., Fischer, I., Dillon, J., Saurous, R. A., and Murphy, K. Fixing a broken elbo. In *International Conference on Machine Learning*, pp. 159–168. PMLR, 2018.

Ammar, W., Dyer, C., and Smith, N. A. Conditional random field autoencoders for unsupervised structured prediction. In *Advances in Neural Information Processing Systems*, pp. 3311–3319, 2014.

Cai, J., Jiang, Y., and Tu, K. Crf autoencoder for unsupervised dependency parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1638–1643, 2017.

Chiu, J. and Rush, A. M. Scaling hidden markov language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1341–1349, 2020.

Correia, G., Niculae, V., Aziz, W., and Martins, A. Efficient marginalization of discrete and structured latent variables via sparsity. *Advances in Neural Information Processing Systems*, 33, 2020.

Corro, C. and Titov, I. Differentiable perturb-and-parse: Semi-supervised parsing with a structured variational autoencoder. *arXiv preprint arXiv:1807.09875*, 2018.

Dozat, T. and Manning, C. D. Deep biaffine attention for neural dependency parsing. *ICLR*, 2016.

Eisner, J. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In *Proceedings of the Workshop on Structured Prediction for NLP*, pp. 1–17, Austin, TX, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/W16-5901. URL https://aclanthology.org/W16-5901.

Fu, Y., Tan, C., Bi, B., Chen, M., Feng, Y., and Rush, A. M. Latent template induction with gumbel-crf. In *NeurIPS*, 2020.

Jeong, M., Lin, C.-Y., and Lee, G. G. Efficient inference of crfs for large-scale natural language data. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pp. 281–284, 2009.

Kim, Y., Rush, A. M., Yu, L., Kuncoro, A., Dyer, C., and Melis, G. Unsupervised recurrent neural network grammars. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 1105–1117, 2019.

Klein, D. and Manning, C. D. A* parsing: Fast exact viterbi parse selection. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 119–126, 2003.

Kool, W., van Hoof, H., and Welling, M. Estimating gradients for discrete random variables by sampling without replacement. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rklEj2EFvB.

Lavergne, T., Cappé, O., and Yvon, F. Practical very large scale crfs. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pp. 504–513, 2010.

Li, C., Gao, X., Li, Y., Peng, B., Li, X., Zhang, Y., and Gao, J. Optimus: Organizing sentences via pre-trained modeling of a latent space. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4678–4699, 2020.

Li, X. L. and Rush, A. Posterior control of blackbox generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2731–2743, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.243. URL https://aclanthology.org/2020.acl-main.243.

Liu, R., Regier, J., Tripuraneni, N., Jordan, M., and McAuliffe, J. Rao-blackwellized stochastic gradients for discrete distributions. In *International Conference on Machine Learning*, pp. 4023–4031. PMLR, 2019.

Ma, X. and Hovy, E. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1064–1074, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1101. URL https://aclanthology.org/P16-1101.

Martins, A. F. T., Mihaylova, T., Nangia, N., and Niculae, V. Latent structure models for natural language processing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pp. 1–5, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-4001. URL https://aclanthology.org/P19-4001.

Mathieu, E., Rainforth, T., Siddharth, N., and Teh, Y. W. Disentangling disentanglement in variational autoencoders. In *International Conference on Machine Learning*, pp. 4402–4412. PMLR, 2019.
Scaling Structured Inference with Randomization

Mensch, A. and Blondel, M. Differentiable dynamic programming for structured prediction and attention. In *International Conference on Machine Learning*, pp. 3462–3471. PMLR, 2018.

Mohamed, S., Rosca, M., Figurnov, M., and Mnih, A. Monte carlo gradient estimation in machine learning. *J. Mach. Learn. Res.*, 21(132):1–62, 2020.

Oktay, D., McGreivy, N., Aduol, J., Beatson, A., and Adams, R. P. Randomized automatic differentiation. In *International Conference on Learning Representations*, 2020.

Paulus, M. B., Choi, D., Tarlow, D., Krause, A., and Maddison, C. J. Gradient estimation with stochastic softmax tricks. *arXiv preprint arXiv:2006.08063*, 2020.

Rush, A. M. Torch-struct: Deep structured prediction library. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 335–342, 2020.

Sokolovska, N., Lavergne, T., Cappé, O., and Yvon, F. Efficient learning of sparse conditional random fields for supervised sequence labeling. *IEEE Journal of Selected Topics in Signal Processing*, 4:953–964, 2010.

Steedman, M. *The Syntactic Process*. MIT Press, February 2000. ISBN 9780262194204.

Sun, Z., Li, Z., Wang, H., He, D., Lin, Z., and Deng, Z. Fast structured decoding for sequence models. *Advances in Neural Information Processing Systems*, 32:3016–3026, 2019.

Sutton, C. and McCallum, A. An introduction to conditional random fields for relational learning. *Introduction to statistical relational learning*, 2:93–128, 2006.

Wainwright, M. J. and Jordan, M. I. *Graphical models, exponential families, and variational inference*. Now Publishers Inc, 2008.

Wiseman, S., Shieber, S., and Rush, A. Learning neural templates for text generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3174–3187, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1356. URL https://aclanthology.org/D18-1356.

Yang, S., Zhao, Y., and Tu, K. Pcfgs can do better: Inducing probabilistic context-free grammars with many symbols. *arXiv preprint arXiv:2104.13727*, 2021.

Zhang, Y., Li, Z., and Zhang, M. Efficient second-order treecrf for neural dependency parsing. *arXiv preprint arXiv:2005.00975*, 2020.

Zhu, H., Bisk, Y., and Neubig, G. The return of lexical dependencies: Neural lexicalized pcfgs. *Transactions of the Association for Computational Linguistics*, 8:647–661, 2020.