Tensor Variable Elimination for Plated Factor Graphs

Fritz Obermeyer [*1] Eli Bingham [*1] Martin Jankowiak [*1] Justin Chiu [2] Neeraj Pradhan [1] Alexander M. Rush [2]
Noah Goodman [1 3]

Abstract

A wide class of machine learning algorithms can be reduced to variable elimination on factor graphs. While factor graphs provide a unifying notation for these algorithms, they do not provide a compact way to express repeated structure when compared to plate diagrams for directed graphical models. To exploit efficient tensor algebra in graphs with plates of variables, we generalize undirected factor graphs to plated factor graphs and variable elimination to a tensor variable elimination algorithm that operates directly on plated factor graphs. Moreover, we generalize complexity bounds based on treewidth and characterize the class of plated factor graphs for which inference is tractable. As an application, we integrate tensor variable elimination into the Pyro probabilistic programming language to enable exact inference in discrete latent variable models with repeated structure. We validate our methods with experiments on both directed and undirected graphical models, including applications to polyphonic music modeling, animal movement modeling, and latent sentiment analysis.

1. Introduction

Factor graphs (Kschischang et al., 2001) provide a unifying representation for a wide class of machine learning algorithms as undirected bipartite graphs between variables and factors. Factor graphs can be used with both directed and undirected graphical models to represent probabilistic inference algorithms performed by variable elimination (Pearl, 1986; Lauritzen & Spiegelhalter, 1988). In the most common case, variable elimination is performed by sum-product inference, but other variable elimination algorithms can be derived through alternative semirings and adjoints.

*Equal contribution [1]Uber AI Labs [2]Harvard University [3]Stanford University. Correspondence to: Fritz Obermeyer <fritzo@uber.com>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

In recent years researchers have exploited the equivalence of sum-product on discrete factor graphs and tensor contraction (Hirata, 2003; Smith & Gray, 2018). Standard tensor contraction can provide efficient implementations of sum-product inference and the tensor contraction operations can be generalized to alternate semirings (Kohlas & Wilson, 2008; Belle & De Raedt, 2016; Khamis et al., 2016).

Yet, a major downside of factor graphs as an intermediary representation is that they discard useful information from higher-level representations, in particular, *repeated structure*. Directed graphical models explicitly denote repeated structure through plate notation (Buntine, 1994). Plates have not seen widespread use in factor graphs or their inference algorithms ((Dietz, 2010) being an exception for directed factor graphs). Nor have plates been exploited by tensor contraction, despite the highly parallel nature of variable elimination algorithms. This gap can result in suboptimal algorithms, since repeated structure can provide information that can be directly exploited for inference optimizations.

In this work we consider the class of *plated factor graphs* and the corresponding tensor variable elimination algorithms. We propose a natural definition for plated factor graphs and lift a number of classic results and algorithms from factor graphs to the plated setting. In particular, we generalize treewidth-based bounds on computational complexity for factor graphs to bounds depending on plate sizes, and characterize the boundary between plated factor graphs leading to computational complexity either polynomial or exponential in the sizes of plates in the factor graph.

We consider several different applications of these techniques. First we describe how plated factor graphs can provide an efficient intermediate representation for generative models with discrete variables, and incorporate a tensor-based contraction into the *Pyro* probabilistic programming language. Next, we develop models for three real-world problems: polyphonic music modeling, animal movement modeling, and latent sentiment analysis. These models combine directed networks (Bayesian networks), undirected networks (conditional random fields), and deep neural networks. We show how plated factor graphs can be used to concisely represent structure and provide efficient general-purpose inference through tensor variable elimination.

2. Related Work

Dense sum-product problems have been studied by the HPC community under the name tensor contraction. Hirata (2003) implemented a Tensor Contraction Engine for performing optimized parallel tensor contractions, an instance of dense sum-product message passing. Solomonik et al. (2014) implement a similar framework for large-scale distributed computation of tensor contractions. Wiebe (2011. URL https://mail.python.org/pipermail/numpy-discussion/2011-January/054586.html) implemented a popular interface `np.einsum` for performing tensor contractions in NumPy. Smith & Gray (2018) implement a divide-and-conquer optimizer for `einsum` operations; we extended their implementation and use it as a primitive for non-plated variable elimination throughout this paper. Abseher et al. (2017) also address the problem of finding efficient junction tree decompositions.

Sparse sum-product problems have a long history in database query optimization, as they can be seen as a specific type of a join-groupby-aggregate query. Kohlas & Wilson (2008) define an abstract framework for variable elimination over valuations and semirings. Khamis et al. (2016) formulate an abstract variable elimination algorithm, prove complexity bounds, and connect these algorithms to database query optimization. Lifted inference algorithms (Taghipour et al., 2013) developed for probabilistic databases are also concerned with extending classical methods for sum-product problems to exploit repeated structure or symmetry in graphical models.

In the context of probabilistic inference, Bilmes (2010) leverages repeated (typically dynamic) structure in graphical models to quickly compute a variable elimination schedule that is then executed sequentially. By contrast our algorithm addresses the narrower class of models which exclude dependencies between plate instances, and can thereby compute a schedule for parallel variable elimination. Infer.Net (Minka et al., 2018) introduces a `ForEach` construct that enables parallelization; our `pyro.plate` construct similarly enables parallism but also declares statistical independence.

3. Model: Plated Factor Graphs

Definition 1. A *factor graph* is a bipartite graph (V, F, E) whose vertices are either variables $v \in V$ or factors $f \in F$, and whose edges $E \subseteq V \times F$ are pairs of vertices. We say factor f involves variable v iff $(v, f) \in E$. Each variable v has domain $\mathrm{dom}(v)$, and each factor f involving variables $\{v_1, \ldots, v_K\}$ maps values $\underline{x} \in \mathrm{dom}(v_1) \times \cdots \times \mathrm{dom}(v_K)$ to scalars.

In this work we are interested in discrete factor graphs where variable domains are finite and factors f are tensors with one dimension per neighboring variable. The key quantity of interest is defined through the sum-product contraction,

$$\textsc{SumProduct}(F, \{v_1, \ldots, v_K\}) = \sum_{x_1 \in \mathrm{dom}(v_1)} \cdots \sum_{x_K \in \mathrm{dom}(v_K)} \prod_{f \in F} f[v_1 = x_1, \ldots, v_K = x_K]$$

where we use named tensor dimensions and assume that tensors broadcast by ignoring uninvolved variables, i.e. if $(v, f) \notin E$ then $f[v = x, v' = x'] = f[v' = x']$.

Definition 2. A *plated factor graph* is a labeled bipartite graph (V, F, E, P) whose vertices are labeled by the *plates* on which they are replicated $P : V \cup F \to \mathcal{P}(B)$, where B is a set of plates. We require that each factor is in each of the plates of its variables: $\forall (v, f) \in E,\ P(v) \subseteq P(f)$.[1]

To instantiate a plated factor graph, we assume a map M that specifies the number of times $M(b)$ to replicate each plate b. Under this definition each plated factor is represented by a tensor with dimensions for both its plates and its involved variables. Using the same partial tensor notation above, we can access a specific grounded factor by indexing its plate dimensions, i.e. $f[b_1 = i_1, \ldots, b_L = i_L]$, where for each dimension $i_l \in \{1, \ldots, M(b_l)\}$ and $f[b = i, b' = i'] = f[b = i]$ if $b' \notin P(f)$.

The key operation for plated factor graphs will be "unrolling" to standard factor graphs.[2] First define the following plate notation for either a factor or variable z: $\mathcal{M}_z(b) = \{1, \ldots, M(b)\}$ if $b \in P(z)$ and $\{1\}$ otherwise. This is the set of indices that index into the replicated variable or factor. Now define a function to unroll a plate,

$$(V', F', E', P') = \mathrm{unroll}((V, F, E, P), M, b)$$

where v_i indicates an unrolled index of v and,

$$\begin{aligned}
V' &= \{v_i \mid v \in V,\ i \in \mathcal{M}_v(b)\} \\
F' &= \{f_i \mid f \in F,\ i \in \mathcal{M}_f(b)\} \\
E' &= \{(v_i, f_j) \mid (v, f) \in E,\ i \in \mathcal{M}_v(b),\ j \in \mathcal{M}_f(b), \\
&\qquad (i = j) \vee b \notin (P(v) \cap P(f))\} \\
P'(z) &= P(z) \setminus \{b\}
\end{aligned}$$

Sum-product contraction naturally generalizes to plated factor graphs via unrolling. We define:

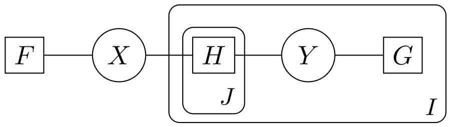
$$\textsc{PlatedSumProduct}(G, M) \equiv \textsc{SumProduct}(F', V')$$

where F' and V' are constructed by unrolling each plate b in the original plated factor graph G.

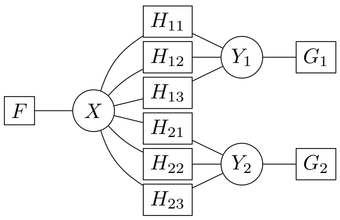
[1] While the final requirement $P(f) \supseteq P(v)$ could perhaps be relaxed, it is required if factors f are to be represented as multi-dimensional tensors.

[2] This operation is conceptual: it is used in our definitions and proofs, but none of our algorithms explicitly unroll factor graphs.

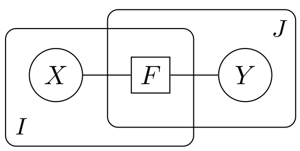
Example 3.1. Consider a plated factor graph with two variables X, Y, three factors F, G, H, and two nested plates:



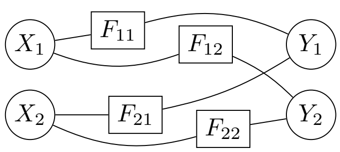
Assuming sizes $I = 2$ and $J = 3$, this plated factor graph unrolls to a factor graph:



Example 3.2. Consider a plated factor graph with two variables, one factor, and two overlapping non-nested plates, denoting a Restricted Boltzmann Machine (RBM) (Smolensky, 1986):



Assuming sizes $I = 2$ and $J = 2$, this plated factor graph unrolls to a factor graph:



4. Inference: Tensor Variable Elimination

We now describe an algorithm—tensor variable elimination—for computing PLATEDSUMPRODUCT on a tractable subset of plated factor graphs. We show that variable elimination cannot be generalized to run in polynomial time on all plated factor graphs and that the algorithm succeeds for exactly those plated factor graphs that can be run in polynomial time. Finally, we briefly discuss extensions of the algorithm, including a plated analog of the Viterbi algorithm on factor graphs.

4.1. An algorithm for tensor variable elimination

The main algorithm is formulated in terms of several standard functions: SUMPRODUCT(F, V), introduced above, computes sum-product contraction of a set of tensors F along a subset V of their dimensions (here always variable dimensions) via variable elimination[3]; PRODUCT$(f, \underline{b}, M)$ product-reduces a single tensor f along a subset $\underline{b}$ of its dimensions (here always plate dimensions)

$$\text{PRODUCT}(f, \{b_1, \ldots, b_L\}, M) = \prod_{i_1=1}^{M(b_1)} \cdots \prod_{i_L=1}^{M(b_L)} f[b_1 = i_1, \ldots, b_L = i_L]$$

and PARTITION(V, F, E) separates a bipartite graph into its connected components. SUMPRODUCT and PRODUCT each return a tensor, with dimensions corresponding to remaining plates and variables, unless all dimensions get reduced, in which case they return a scalar. Intuitively, SUMPRODUCT eliminates variables and PRODUCT eliminates plates.

At a high level, the strategy of the algorithm is to greedily eliminate variables and plates along a tree of factors. At each stage it picks the most deeply nested plate set, which we call a leaf plate. It eliminates all variables in exactly that plate set via standard variable elimination, producing a set of reduced factors. Each reduced factor is then replaced by a product factor that eliminates one or more plates. Repeating this procedure until no variables or plates remain, all scalar factors are finally combined with a product operation.

Algorithm 1 specifies the full algorithm. Elimination of variables and plates proceeds by modifying the input plated factor graph (V, F, E, P) and a partial result S containing scalars. Both loops preserve the invariant that (V, F, E, P) is a valid plated factor graph and preserve the quantity

$$\text{PLATEDSUMPRODUCT}((V, F, E, P), M) \times \text{SUMPRODUCT}(S, \{\}).$$

At each leaf plate set $L \subseteq B$, the algorithm decomposes that plate set's factor graph into connected components. Each connected component is SUMPRODUCT-contracted to a single factor f with no variables remaining in the leaf plate set L. If the resulting factor f has no more variables, it is PRODUCT-reduced to a single scalar. Otherwise the algorithm seeks a plate set $L' \subsetneq L$ where other variables of f can be eliminated; f is partially PRODUCT-reduced to a factor f' that is added back to the plated factor graph, including edges from $V_f \times F_c$ that had been removed. Finally, when no more variables or plates remain, all scalar factors are product-combined by a trivial SUMPRODUCT$(S, \{\})$. The algorithm can fail with **error** if the search for a next plate set L' fails.

To help characterize when Algorithm 1 succeeds, we now introduce the concept of a graph minor.

Definition 3. A plated graph[4] H is a *minor* of the plated graph G if it can be obtained from G by a sequence of edits

[3] Available in many machine learning libraries as `einsum`.

[4] When considering graph minors, we view factor graphs (V, F, E) as undirected graphs $(V \cup F, E)$.

Algorithm 1 TENSORVARIABLEELIMINATION

input variables V, factors F, edges $E \subseteq V \times F$,
  plate sets $P : V \cup F \to \mathcal{P}(B)$,
  plate sizes $M : B \to \mathbb{N}$.
output PLATEDSUMPRODUCT$((V,F,E,P), M)$ or **error**.
Initialize an empty list of scalars $S \leftarrow []$.
while F is not empty **do**
  Choose a leaf plate set $L \in \{P(f) \mid f \in F\}$ with a maximal number of plates.
  Let $V_L \leftarrow \{v \in V \mid P(v) = L\}$ be the variables in L.
  Let $F_L \leftarrow \{f \in F \mid P(f) = L\}$ be the factors in L.
  Let $E_L \leftarrow E \cap (V_L \times F_L)$ be the edges in L.
  for (V_c, F_c) **in** PARTITION(V_L, F_L, E_L) **do**
    Let $f \leftarrow$ SUMPRODUCT(F_c, V_c).
    Let $V_f \leftarrow \{v \mid (v, f) \in E \cap ((V \setminus V_c) \times F_c)\}$ be the set of f's remaining variables.
    Remove component (V_c, F_c) from V, F, E, P.
    if V_f is empty **then**
      Add PRODUCT(f, L, M) to scalars S.
    else
      Let $L' \leftarrow \bigcup\{P(v) \mid v \in V_f\}$ be the next plate set where f has variables.
      if $L' = L$ **then error**("Intractable!");
      Let $f' \leftarrow$ PRODUCT$(f, L \setminus L', M)$.
      Add f' to F, E, P appropriately.
return SUMPRODUCT$(S, \{\})$

of the form: deleting a vertex, deleting an edge, deleting a plate, or merging two vertices u, v connected by an edge and in identical plates $P(u) = P(v)$.

Theorem 1. *Algorithm 1 succeeds iff G has no plated graph minor* $(\{u, v, w\}, \{(u, v), (v, w)\}, P)$ *where* $P(u) = \{a\}$, $P(v) = \{a, b\}$, $P(w) = \{b\}$, $a \neq b$, *and* u, w *both include variables.*

Proof. See Appendix A.1. □

This plated graph minor exclusion property essentially excludes the RBM Example 3.2 above, which is a minimal example of an intractable input.[5] If Algorithm 1 fails, one could fall back to unrolling a plate and continuing (at cost exponential in plate size); we leave this for future research.

4.2. Complexity of tensor variable elimination

It is well known that message passing algorithms have computational complexity exponential in the treewidth of the input factor graph but only linear in the number of variables (Chandrasekaran et al., 2012; Kwisthout et al., 2010). In this section we generalize this result to the complexity of plated message passing on tensor factors. We show that for an easily identifiable class of plated factor graphs, serial complexity is polynomial in the tensor sizes of the factors, and parallel complexity is sublinear in the size of each plate. Essentially this characterizes when the tensor size of one factor of a plated factor graph determines the treewidth of the unrolled non-plated factor graph: in the polynomial case, treewidth is independent of tensor size.

Example 4.1. Consider the plated factor graph of Example 3.1 with nested plates. We wish to compute,

$$\sum_x \sum_{y_1} \cdots \sum_{y_I} F_x \Big[\prod_i G_{i,y_i}\Big] \Big[\prod_{i,j} H_{i,j,x,y_i}\Big] = \sum_x F_x \prod_i \sum_{y_i} G_{i,y_i} \prod_j H_{i,j,x,y_i}$$

where we are able to commute the sums over y_i inside the product over i, thereby reducing an algorithm of cost exponential in I to a polynomial-cost algorithm.

Example 4.2. Consider the plated factor graph of Example 3.2 with overlapping, non-nested plates. Although the plated factor graph is a tree of tensors, the unrolled factor graph has treewidth $O(I + J)$; hence complexity will be exponential in the tensor sizes I, J of dimensions i, j.

We can reach the same conclusion from sum-product,

$$\sum_{x_1} \cdots \sum_{x_I} \sum_{y_1} \cdots \sum_{y_J} \prod_{i,j} F_{i,j,x_i,y_j} = \sum_{x_1} \cdots \sum_{x_I} \prod_{i,j} \sum_{y_j} F_{i,j,x_i,y_j}$$

Here we cannot commute both Cartesian product summations inside the product and so the computation is necessarily exponential in I, J.

We now show Algorithm 1 accepts the largest class of plated factor graphs for which a polynomial time strategy exists.

Theorem 2. *Let* $G = (V, F, E, P : V \cup F \to \mathcal{P}(B))$ *be a plated factor graph. Assume variable domains have nontrivial sizes* $|\mathrm{dom}(v)| \geq 2$, $\forall v \in V$. *Then Algorithm 1 succeeds on* G *iff* PLATEDSUMPRODUCT(G, M) *can be computed with complexity polynomial in plate sizes* $M : B \to \mathbb{N}$.

Proof sketch. (see Appendix A.2 for full proof).

($\Rightarrow$) Algorithm 1 has complexity polynomial in M.

($\Leftarrow$) Appeal to (Kwisthout et al., 2010) (which assumes the Exponential Time Hypothesis) to show that the unrolled factor graph must have uniformly bounded treewidth. Apply Ramsey theory arguments to show there is a single "plated junction tree" with certain properties. Show this tree can only exist if F excludes the minor of Thm. 1, hence Algorithm 1 succeeds. □

[5] See Appendix C.1 for a detailed walk-through of Algorithm 1 on this model, leading to **error**.

When a plated factor graph is asymptotically tractable according to Thm. 2, it is also tractable on a parallel machine.

Theorem 3. *If Algorithm 1 runs in sequential time T when $M(b) = 1,\ \forall b$, then it runs in time $T + O\left(\Sigma_b \log M(b)\right)$ on a parallel machine with $\Pi_b M(b)$ processors and perfect efficiency.*

Proof. Algorithm 1 depends on M only through calls to SUMPRODUCT and PRODUCT. SUMPRODUCT operates independently over plates, and hence parallelizes with perfect efficiency. PRODUCT reductions over each plate b incur only $O(\log M(b))$ time on a parallel machine. □

This result suggests that (tractable) plated factor graphs are a practical model class for modern hardware. See Sec. 6.4 for empirical verification of the computational complexity described in Thm. 3.

While tensor variable elimination enables parallelism without increasing arithmetic operation count, it also reduces overhead involved in graph manipulation. A significant portion of runtime in SUMPRODUCT is spent on constructing a junction tree. By exploiting repeated structure, tensor variable elimination can restrict its junction tree problems to much smaller factor graphs (constructed locally for each connected component of each plate set), leading to lower overhead and the opportunity to apply better heuristics.[6]

4.3. Generic tensor variable elimination

Because Algorithm 1 is generic[7] in its two operations (plus and multiply), it can immediately be repurposed to yield other algorithms on plated factor graphs, for example a polynomial-time PLATEDMAXPRODUCT algorithm that generalizes the MAXPRODUCT algorithm on factor graphs.

Further extensions can be efficiently formulated as adjoint algorithms, which proceed by recording an adjoint compute graph alongside the forward computation and then traversing the adjoint graph backwards starting from the final result of the forward computation (Darwiche, 2003; Eisner, 2016; Azuma et al., 2017; Belle & De Raedt, 2016). These extensions include: computing marginal distributions of all variables (generalizing the forward-backward algorithm); maximum a posteriori estimation (generalizing Viterbi-like algorithms); and drawing joint samples of variables (generalizing the forward-filter backward-sample algorithm). Note that while most message passing algorithms need only assume that the sum and product operations have semiring structure (Kohlas & Wilson, 2008; Khamis et al., 2016), marginal computations additionally require division.

[6]E.g. opt_einsum uses an optimal strategy for factor graphs with up to four variables.

[7]In the sense of generic programming (Musser & Stepanov, 1988)

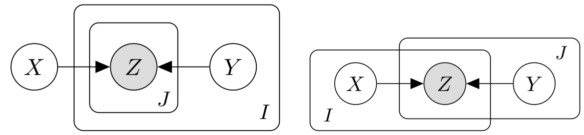


Figure 1. The tractable and intractable plated factor graphs in Examples 3.1 and 3.2 arise from the plated graphical model on the left and right, respectively. Here the two random variables X, Y are unobserved and the random variable Z is observed.

5. Application to Probabilistic Programming

Plated factor graphs provide a general-purpose intermediate representation for many applications in probabilistic modeling requiring efficient inference and easy parallelization. To make tensor variable elimination broadly usable for these applications, we integrate it into two frameworks[8] and use both frameworks in our experiments.

5.1. Plated probabilistic programs in Pyro

First we integrate our implementation into the Pyro probabilistic programming language (Bingham et al., 2018). This allows us to specify discrete latent variable models easily, programmatically constructing complex distributions and explicitly indicating repeated structure. The syntax relies on a `plate` context manager.[9] Inside these plates, `sample` statements are batched and assumed to be conditionally independent along the plate dimension.

Example 5.1. The directed graphical model in Fig. 1 (left) can be specified by the Pyro program (see Appendix D)

```
def model(z):
  I, J = z.shape
  x = pyro.sample("x", Bernoulli(Px))
  with pyro.plate("I", I, dim=-2):
    y = pyro.sample("y", Bernoulli(Py))
    with pyro.plate("J", J, dim=-1):
      pyro.sample("z", Bernoulli(Pz[x,y]),
                  obs=z)
```

To extract plated factor graphs from such programs without using static analysis, we use a nonstandard interpretation of Pyro `sample` statements (Wingate et al., 2011) as vectorized enumeration over each distribution's support. That is, when running a program forward we create a tensor at each sample site that lists all possible values of each distribution rather than draw random samples. To avoid conflict among multiple enumerated variables, we dynamically assign each variable a distinct tensor dimension along which its values are enumerated, relying on array broadcasting (Walt et al., 2011) to correctly combine results of multiple nonstandard

[8]Open-source implementations are available; see `http://docs.pyro.ai/en/dev/ops.html`

[9]We also introduce a `markov` context manager to deal with markov structure.

sample values (as in `Pz[x,y]` above).

5.2. Undirected graphical models using einsum

A standard interface for expressing inference in undirected graphical models is `einsum` notation. Basic `einsum` computes the product of a collection of tensors, with some dimensions preserved in the final result and remaining dimensions summed out. This operation behaves analogously to SUMPRODUCT, with the inputs to `einsum` given as: a string denoting the tensor dimensions of all input and output tensors, and a list of tensors. The former gives the factor graph topology and the latter the factors themselves. For example, `einsum("xy,yz->xz",F,G)` corresponds to matrix multiplication for matrices `F,G`. This `einsum` expression also corresponds to reducing the y variable in a factor graph with three nodes and two factors.

Plated `einsum` generalizes `einsum` notation, e.g.

```
einsum("xy,iyz->xz",F,G,plates="i") ==
einsum("xy,yz,yz,yz->xz",F,G[0],G[1],G[2])
```

where dimension `x,z` are preserved, variable `y` is sum-reduced, and plate `i` is product-reduced. More generally, factors can include plated variables, e.g. Example 3.1 can be written as

```
einsum("x,iy,ijxy->",F,G,H,plates="ij") ==
einsum("x,y,z,xy,xy,xy,xz,xz,xz->",
       F,G[0],G[1],H[0,0],H[0,1],
       H[0,2],H[1,0],H[1,1],H[1,2])
```

where the rightmost dimension of `G` and `H` denotes a distinct variable for each slice `i` (`y` and `z` in the unrolled version). Thus the effective number of variables grows with the size of plate `i`, but the plated notation requires only a constant number of symbols. Formally we infer each variable's plate set $P(v)$ as the largest set consistent with the input string. In addition, this version of `einsum` implements generic tensor variable elimination, and so it can also be used to compute marginals and draw samples with the same syntax.

6. Experiments

We experiment with plated factor graphs as a modeling language for three tasks: polyphonic music prediction, animal movement modeling and latent sentiment analysis.[10] Experiments consider different variants of discrete probabilistic models and their combination with neural neworks.

6.1. Hidden Markov Models with Autoregressive Likelihoods

In our first experiment we train variants of a hidden Markov model[11] (HMM) on a polyphonic music modeling task (Boulanger-Lewandowski et al., 2012). The data consist of sequences $\{\boldsymbol{y}_1, ..., \boldsymbol{y}_T\}$, where each $\boldsymbol{y}_t \in \{0,1\}^{88}$ denotes the presence or absence of 88 distinct notes. This task is a common challenge for latent variable models such as continuous state-space models (SSMs), where one of the difficulties of inference is that training is typically stochastic, i.e. the latent variables need to be sampled.[12] In contrast, the discrete latent variable models explored here—each defined through a plated factor graph—admit efficient tensor variable elimination, obviating the need for sampling.

	Dataset		
Model	JSB	Piano	Nottingham
`HMM`	8.28	9.41	4.49
`FHMM`	8.40	9.55	4.72
`PFHMM`	8.30	9.49	4.76
`2HMM`	8.70	9.57	4.96
`arHMM`	8.00	7.30	3.29
`arFHMM`	8.22	7.36	3.57
`arPFHMM`	8.39	9.57	4.82
`ar2HMM`	8.19	**7.11**	3.34
`nnHMM`	**6.73**	7.32	**2.67**
`nnFHMM`	6.86	7.41	2.82
`nnPFHMM`	7.07	7.47	2.81
`nn2HMM`	6.78	7.29	2.81

Table 1. Negative log likelihoods for HMM variants on three polyphonic music test datasets; lower is better. See Sec. 6.1 for details.

We consider 12 different latent variable models, where the emission likelihood for each note is replicated on a plate of size 88, so that notes at each time step are conditionally independent. Writing these models as probabilistic programs allows us to easily experiment with dependency structure and parameterization, with variation along two dimensions:

1. the dependency structure of the discrete latent variables
2. whether the likelihood $p(\boldsymbol{y}_t|\cdot)$ is autoregressive, i.e. whether it depends on $\boldsymbol{y}_{t-1}$, and if so whether $p(\boldsymbol{y}_t|\cdot)$ is parameterized by a neural network

Dependency structures include a vanilla HMM (`HMM`); two variants of a Factorial HMM (`FHMM` & `PFHMM`) (Ghahramani & Jordan, 1996); and a second-order HMM (`2HMM`).[13] The models denoted by `arXXX` and `nnXXX` include an autoregressive likelihood: the former are explicitly parameterized with a conditional probability table, while the latter use a neural network to parameterize the likelihood. (See the supplementary materials for detailed descriptions.)

We report our results in Table 1. We find that our ability to iterate over a large class of models[14]—in particular

[10]See the supplementary materials for plate diagrams for the models in Sec. 6.1 and Sec. 6.2.

[11]For an introduction see e.g. reference (Ghahramani, 2001).

[12]This is true for all five reference models in this section, apart from the RTRBM.

[13]For the second-order HMM we use a parsimonious Raftery parameterization of the transition probabilities (Raftery, 1985).

[14]See `https://git.io/fjc82` for a reference implemen-

different latent dependency structures, each of which requires a different message passing algorithm—proved useful, since different datasets preferred different classes of models. Autoregressive models yield the best results, and factorial HMMs perform worse across most models.

We note that these classic HMM variants (upgraded with neural network likelihoods) are competitive with baseline state space models, outperforming STORN (Bayer & Osendorfer, 2014) on 3 of 3 datasets, LV-RNN (Gu et al., 2015) on 2 of 3 datasets, Deep Markov Model (Krishnan et al., 2017) on 2 of 3 datasets, RTRBM (Sutskever et al., 2009; Boulanger-Lewandowski et al., 2012) on 1 of 3 datasets, and TSBN (Gan et al., 2015) on 3 of 3 datasets.

6.2. Hierarchical Mixed-Effect Hidden Markov Models

In our second set of experiments, we consider models for describing the movement of populations of wild animals. Recent advances in sensor technology have made it possible to capture the movements of multiple animals in a population at high spatiotemporal resolution (McClintock et al., 2013). Time-inhomogeneous discrete SSMs, where the latent state encodes an individual's behavior state (like "foraging" or "resting") and the state transition matrix at each timestep is computed with a hierarchical discrete generalized linear mixed model, have become popular tools for data analysis thanks to their interpretability and tractability (Zucchini et al., 2016; McClintock & Michelot, 2018).

Rapidly iterating over different variants of such models, with nested plates and hierarchies of latent variables that couple large groups of individuals within a population, is difficult to do by hand but can be substantially simplified by expressing models as plated probabilistic programs and performing inference with tensor variable elimination.[15]

To illustrate this, we implement a version of the model selection process for movement data from a colony of harbour seals in the United Kingdom described in (McClintock et al., 2013), fitting three-state hierarchical discrete SSMs with no random effects (`No RE`, a vanilla HMM), sex-level discrete random effects (`Group RE`), individual-level discrete random effects (`Individual RE`), and both sex- and individual-level discrete random effects (`Individual+Group RE`). See the supplement for details on the dataset, models and training procedure.

We report AIC scores for all models in Table 2. Although our models do not exactly match those in the original analysis,[16] our results support theirs in suggesting that including individual-level random effects is essential because there is

tation for a selection of HMM variants.

[15]See `https://git.io/fjc8a` for a reference implementation.

[16]See supplement for a discussion of differences.

Model	AIC
`No RE (HMM)`	353×10^3
`Individual RE`	341×10^3
`Group RE`	342×10^3
`Individual+Group RE`	341×10^3

Table 2. Akaike Information Criterion (AIC) scores for hierarchical mixed effect HMM variants fit with maximum likelihood on animal movement data. Lower is better. See Sec. 6.2.

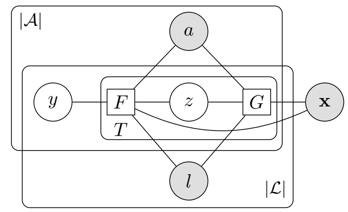


Figure 2. The graphical model for the sentiment analysis CRF in Sec. 6.3. The aspect a and location l are observed, while the word-level sentiments z_t and sentence-level sentiment y for the particular aspect and location pair must be inferred. See the supplementary material for the exact parameterization of the factors.

significant behavioral variation across individuals and sexes that is unexplained by the available covariates.

6.3. Latent Variable Classification

We next experiment with model flexibility by designing a conditional random field (Lafferty et al., 2001) model for latent variable classification on a sentiment classification task. Experiments use the Sentihood dataset (Saeidi et al., 2016), which consists of sentences containing named location entities with sentiment labels along different aspects. For a sentence $\mathbf{x} = \langle x_1, \ldots, x_T \rangle$ labels are tuples (a, l, y) that contain an aspect $a \in \mathcal{A} = \{\text{general}, \text{safety}, \ldots\}$, a location $l \in \mathcal{L} = \{\texttt{Location1}, \texttt{Location2}\}$, and a sentiment $y \in \{\text{positive}, \text{negative}, \text{none}\}$. The task is to predict the sentiment of a sentence given a location and aspect, for example $p(y \mid \mathbf{x}, a = \text{price}, l = \texttt{Location1})$.

Standard approaches to this sentence-level classification task use neural network models to directly make sentence-level predictions. We instead propose a latent variable approach that explicitly models the sentiment of each word with respect to all locations and aspects. This approach can provide clearer insight into the specific reasoning of the model and also permits conditional inference.

Our conditional random field model is represented as a plated factor graph in Figure 2. Here $\mathbf{z} = \langle z_1, \ldots, z_T \rangle$ is the latent word-level sentiment for fixed l and a. The two plated factors G and F represent the word aspect-location-sentiment potentials and the word-sentence potentials, respectively. We parameterize these factors with

	Metric	
Model	Acc	F1
`LSTM-Final`	0.821	0.780
`CRF-LSTM-Diag`	0.805	0.764
`CRF-LSTM-LSTM`	0.843	0.799
`CRF-Emb-LSTM`	0.833	0.779

Table 3. Test sentiment accuracies and aspect F1 scores for sentiment classification models on Sentihood. Higher is better for both metrics. Both metrics are calculated by ignoring examples with gold labelled 'none' sentiments; see Sec. 6.3 for details.

y	Loc1	...	but	not	too	convenient
😐	0	1	.89	.98	.84	0
☺	0	0	0	0	0	.31
☹	1	0	.11	.02	.16	.69

Figure 3. The inferred word-level sentiment z_t for aspect *transit-location* and location `Loc1` conditioned on a *negative* sentence-level sentiment. The ellipsis contains the elided words 'is a great place to live', all of which get *none* sentiment. The model assigns the *none* sentiment to most words, preventing them from influencing the polarity of the sentence-level sentiment.

a bidirectional LSTM (BLSTM) over word embeddings of $\mathbf{x}$ whose initial state is given by an embedding of the location l and aspect a. We experiment with three variants:[17] 1) `CRF-LSTM-Diag` parameterizes G with the output of the BLSTM and F with a diagonal matrix; 2) `CRF-LSTM-LSTM` parameterizes both G and F with the output of the BLSTM; and 3) `CRF-Emb-LSTM` parameterizes G with word embeddings and F with the output of the BLSTM. As a baseline we reimplement the direct BLSTM model `LSTM-Final` model from Saeidi et al. (2016). See the supplementary material for full details.

The accuracy of our models is given in Table 3.[18] The CRF models all achieve similar performance to the baseline `LSTM-Final` model. However, as a result of the factor graph representation, we demonstrate the ability to infer word-level sentiment conditioned on a sentence-level sentiment as well as sparsity of the conditional word-level sentiments for the `CRF-Emb-LSTM` model. We provide an example sentence and the inferred conditional word sentiments for the `CRF-Emb-LSTM` model in Fig. 3.

6.4. Performance Evaluation

To measure the performance of our implementation we use the benchmark plated model[19] of Fig. 4. We compute PLATEDSUMPRODUCT and four different adjoint operations: gradient, marginal, sample, and MAP. Figure 5 shows results obtained on an Nvidia Quadro P6000 GPU. We find that runtime is approximately constant until the GPU is saturated (at $I \times J \approx 10^4$), and runtime is approximately linear in $I \times J$ for larger plate sizes, empirically validating Thm 3.

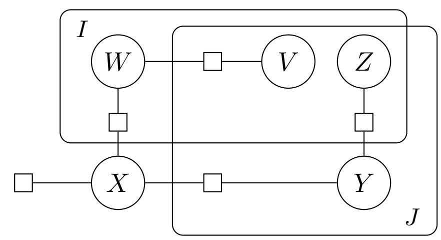


Figure 4. The tractable model used in our performance benchmark. Note that this model would be intractable if there were a factor directly connecting variables V and Z.

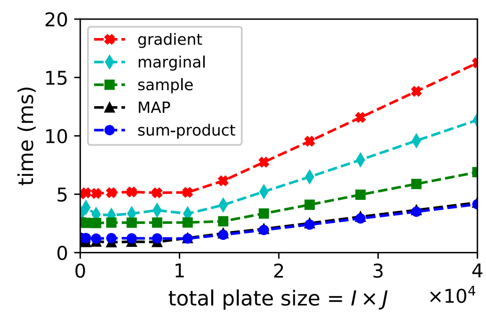


Figure 5. Runtime of Algorithm 1 as we vary plate size (where $I = J$) and fix domain size $|\mathrm{dom}(v)| = 32$ for all variables.

7. Conclusion

This work argues for plated factor graphs as an intermediate representation that preserves shared structure for calculations with discrete random variables, and develops a tensor variable elimination algorithm for efficiently computing plated sum-product and related factor graph queries for a subset of plated models. We show how this approach can be used to compute key quantities for directed and undirected graphical models and demonstrate its implementation as a general purpose intermediary representation for probabilistic programming with discrete random variables. Applications further demonstrate that this provides a simple, flexible, and efficient framework for working with discrete graphical models, and that these models can often outperform more complicated counterparts, while providing interpretable latent representations. The work itself is integrated into a widely used probabilistic programming system, and we hope it provides a framework for experiments incorporating discrete variable models into large-scale systems.

[17] See `https://github.com/justinchiu/sentclass` for a reference implementation.

[18] The results of previous work can be found summarized succinctly in Liu et al. (2018). Note that our goal is not to improve upon previous results (other models have higher accuracy); rather we aim to capitalize on the plated representation to infer latent word-level sentiment.

[19] See Appendix C.2 for a detailed walkthrough of Algorithm 1 on this model.

Acknowledgements

We would like to thank many people for early discussions and feedback, including JP Chen, Theofanis Karaletsos, Ruy Ley-Wild, and Lawrence Murray. MJ would like to thank Felipe Petroski Such for help with infrastructure for efficient distribution of experiments. AMR and JC were supported by NSF #1704834 and #1845664.

References

Abseher, M., Musliu, N., and Woltran, S. Improving the efficiency of dynamic programming on tree decompositions via machine learning. *Journal of Artificial Intelligence Research*, 58:829–858, 2017.

Azuma, A., Shimbo, M., and Matsumoto, Y. An algebraic formalization of forward and forward-backward algorithms. *arXiv preprint arXiv:1702.06941*, 2017.

Bayer, J. and Osendorfer, C. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014.

Belle, V. and De Raedt, L. Semiring programming: A framework for search, inference and learning. *arXiv preprint arXiv:1609.06954*, 2016.

Bilmes, J. Dynamic graphical models. *IEEE Signal Processing Magazine*, 27(6):29–42, 2010.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., and Goodman, N. D. Pyro: Deep Universal Probabilistic Programming. *arXiv preprint arXiv:1810.09538*, 2018.

Boulanger-Lewandowski, N., Bengio, Y., and Vincent, P. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.

Buntine, W. L. Operations for learning with graphical models. *Journal of artificial intelligence research*, 2:159–225, 1994.

Chandrasekaran, V., Srebro, N., and Harsha, P. Complexity of inference in graphical models. *arXiv preprint arXiv:1206.3240*, 2012.

Darwiche, A. A differential approach to inference in bayesian networks. *Journal of the ACM (JACM)*, 50 (3):280–305, 2003.

Dietz, L. Directed factor graph notation for generative models. *Max Planck Institute for Informatics, Tech. Rep*, 2010.

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, 2016.

Gan, Z., Li, C., Henao, R., Carlson, D. E., and Carin, L. Deep temporal sigmoid belief networks for sequence modeling. In *Advances in Neural Information Processing Systems*, pp. 2467–2475, 2015.

Ghahramani, Z. An introduction to hidden markov models and bayesian networks. *International journal of pattern recognition and artificial intelligence*, 15(01):9–42, 2001.

Ghahramani, Z. and Jordan, M. I. Factorial hidden markov models. In *Advances in Neural Information Processing Systems*, pp. 472–478, 1996.

Gu, S. S., Ghahramani, Z., and Turner, R. E. Neural adaptive sequential monte carlo. In *Advances in Neural Information Processing Systems*, pp. 2629–2637, 2015.

Hirata, S. Tensor contraction engine: Abstraction and automated parallel implementation of configuration-interaction, coupled-cluster, and many-body perturbation theories. *The Journal of Physical Chemistry A*, 107(46): 9887–9897, 2003.

Honnibal, M. and Montani, I. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*, 2017.

Khamis, M. A., Ngo, H. Q., and Rudra, A. Faq: questions asked frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pp. 13–28. ACM, 2016.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kohlas, J. and Wilson, N. Semiring induced valuation algebras: Exact and approximate local computation algorithms. *Artificial Intelligence*, 172(11):1360–1399, 2008.

Krishnan, R. G., Shalit, U., and Sontag, D. Structured inference networks for nonlinear state space models. In *AAAI*, pp. 2101–2109, 2017.

Kschischang, F. R., Frey, B. J., and Loeliger, H.-A. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

Kwisthout, J., Bodlaender, H. L., and van der Gaag, L. C. The necessity of bounded treewidth for efficient inference in bayesian networks. In *ECAI*, volume 215, pp. 237–242, 2010.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pp. 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1.

Lauritzen, S. L. and Spiegelhalter, D. J. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 157–224, 1988.

Liu, F., Cohn, T., and Baldwin, T. Recurrent entity networks with delayed memory update for targeted aspect-based sentiment analysis. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 278–283. Association for Computational Linguistics, 2018. doi: 10.18653/v1/N18-2045.

Ma, Y., Peng, H., Khan, T., Cambria, E., and Hussain, A. Sentic LSTM: a hybrid network for targeted aspect-based sentiment analysis. *Cognitive Computation*, 10(4):639–650, 2018.

McClintock, B. T. and Michelot, T. momentuhmm: R package for generalized hidden markov models of animal movement. *Methods in Ecology and Evolution*, 9(6): 1518–1530, 2018. doi: 10.1111/2041-210X.12995.

McClintock, B. T., Russell, D. J., Matthiopoulos, J., and King, R. Combining individual animal movement and ancillary biotelemetry data to investigate population-level activity budgets. *Ecology*, 94(4):838–849, 2013.

Minka, T., Winn, J., Guiver, J., Zaykov, Y., Fabian, D., and Bronskill, J. /Infer.NET 0.3, 2018. Microsoft Research Cambridge. http://dotnet.github.io/infer.

Musser, D. R. and Stepanov, A. A. Generic programming. In *International Symposium on Symbolic and Algebraic Computation*, pp. 13–25. Springer, 1988.

Pearl, J. Fusion, propagation, and structuring in belief networks. *Artificial intelligence*, 29(3):241–288, 1986.

Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. In *In EMNLP*, 2014.

Raftery, A. E. A model for high-order markov chains. *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 528–539, 1985.

Ramsey, F. P. On a problem of formal logic. In *Classic Papers in Combinatorics*, pp. 1–24. Springer, 1930.

Saeidi, M., Bouchard, G., Liakata, M., and Riedel, S. Sentihood: Targeted aspect based sentiment analysis dataset for urban neighbourhoods. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pp. 1546–1556. The COLING 2016 Organizing Committee, 2016. URL `http://aclweb.org/anthology/C16-1146`.

Smith, D. G. A. and Gray, J. `opt_einsum` - a python package for optimizing contraction order for einsum-like expressions. *Journal of Open Source Software*, 3(26):753, 2018. URL `https://doi.org/10.21105/joss.00753`.

Smolensky, P. Information processing in dynamical systems: Foundations of harmony theory. Technical report, COLORADO UNIV AT BOULDER DEPT OF COMPUTER SCIENCE, 1986.

Solomonik, E., Matthews, D., Hammond, J. R., Stanton, J. F., and Demmel, J. A massively parallel tensor contraction framework for coupled-cluster computations. *Journal of Parallel and Distributed Computing*, 74(12):3176–3190, 2014.

Sutskever, I., Hinton, G. E., and Taylor, G. W. The recurrent temporal restricted boltzmann machine. In *Advances in neural information processing systems*, pp. 1601–1608, 2009.

Taghipour, N., Fierens, D., Davis, J., and Blockeel, H. Lifted variable elimination: Decoupling the operators from the constraint language. *Journal of Artificial Intelligence Research*, 47:393–439, 2013.

Towner, A. V., Leos-Barajas, V., Langrock, R., Schick, R. S., Smale, M. J., Kaschke, T., Jewell, O. J., and Papastamatiou, Y. P. Sex-specific and individual preferences for hunting strategies in white sharks. *Functional Ecology*, 30(8):1397–1407, 2016.

Walt, S. v. d., Colbert, S. C., and Varoquaux, G. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

Wiebe, M. Numpy-discussion: einsum, January 2011. URL https://mail.python.org/pipermail/numpy-discussion/2011-January/054586.html.

Wingate, D., Goodman, N., Stuhlmüller, A., and Siskind, J. M. Nonstandard interpretations of probabilistic programs for efficient inference. In *Advances in Neural Information Processing Systems*, pp. 1152–1160, 2011.

Zucchini, W., MacDonald, I. L., and Langrock, R. *Hidden Markov models for time series: an introduction using R*. Chapman and Hall/CRC, 2016.