Tensor Variable Elimination for Plated Factor Graphs


Uber AI, harvardnlp, Stanford
Outline

- Background and Motivation: Discrete Latent Variables
- Models: Plated Factor Graphs
- Inference Algorithm: Tensor Variable Elimination
- Implementation in Pyro
- Experiments and Discussion
Outline

- Background and Motivation: Discrete Latent Variables
- Models: Plated Factor Graphs
- Inference Algorithm: Tensor Variable Elimination
- Implementation in Pyro
- Experiments and Discussion
Learning and inference with discrete latent variables

(Kingma et al. 2014) (McClintock et al. 2016) (Obermeyer et al. 2019)
Learning and inference with discrete latent variables

Probabilistic inference offers a unified approach to uncertainty estimation, model selection, and imputation.

Exact inference is theoretically tractable in many popular discrete latent variable models.

Algorithms and software have not kept up with growth of models and data, and integration with deep learning is difficult and time-consuming.
Factor graphs represent products of functions of many variables. They are a unifying intermediate representation for many types of discrete probabilistic models, like directed graphical models.
Background: Factor graph inference

Probabilistic inference is an instance of a sum-product problem:

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

Sum-product computations on factor graphs are performed by variable elimination:

$$F \xrightarrow{X} H \xrightarrow{Y} G \rightarrow P(Z = z)$$

$$F \xrightarrow{X} B \rightarrow P(Z = z)$$
Outline

- Background and Motivation: Discrete Latent Variables
- Models: Plated Factor Graphs
- Inference Algorithm: Tensor Variable Elimination
- Implementation in Pyro
- Experiments and Discussion
Focus: Plated factor graphs

Plates represent repeated structure in graphical models:

Can we use plates to represent repeated structure in variable elimination algorithms?
Plated factor graph inference

Define the plated sum-product problem on a plated factor graph as the sum-product problem on an *unrolled* version of the plated factor graph:

\[
\text{PlatedSumProduct}(G, M) \equiv \text{SumProduct}(F', V')
\]
Challenges: Plated factor graph inference

Although mathematically convenient, unrolling may limit parallelism, use memory inefficiently, and obscure the relationship to the original model.

Can we derive a variable elimination algorithm that solves the PlatedSumProduct problem directly?
Outline

- Background and Motivation: Discrete Latent Variables
- Models: Plated Factor Graphs
- Inference Algorithm: Tensor Variable Elimination
- Implementation in Pyro
- Experiments and Discussion
Algorithm: Tensor variable elimination

while any factors in graph G have plates:

    L <- maximal factor plate set in G
    G_L <- subgraph of G in L
    for subgraph G_C in \text{Partition}(G_L):
        f <- \text{SumProduct}(G_C)
        L' <- plates of all variables of f in G
        f' <- \text{Product}(f, L - L')
        remove G_C from G and insert f' into G

return \text{SumProduct}(G)
Algorithm: Tensor variable elimination

while any factors in graph G have plates:

L <- maximal factor plate set in G
G_L <- subgraph of G in L
for subgraph G_C in Partition(G_L):
  f <- SumProduct(G_C)
  L' <- plates of all variables of f in G
  f' <- Product(f, L - L')
  remove G_C from G and insert f' into G

return SumProduct(G)

We rely on three plate-aware subroutines to avoid unrolling:

Compute strongly connected components of a bipartite graph

Perform variable elimination on a batch of structurally identical factor graphs

Compute the elementwise product of factors along one or more plate indices
Algorithm: Tensor variable elimination

while any factors in graph G have plates:

L <- maximal factor plate set in G
G_L <- subgraph of G in L
for subgraph G_C in Partition(G_L):
    f <- SumProduct(G_C)
    L' <- plates of all variables of f in G
    f' <- Product(f, L - L')
    remove G_C from G and insert f' into G

return SumProduct(G)
Algorithm: Tensor variable elimination

while any factors in graph G have plates:

L <- maximal factor plate set in G
G_L <- subgraph of G in L
for subgraph G_C in Partition(G_L):
    f <- SumProduct(G_C)
    L' <- plates of all variables of f in G
    f' <- Product(f, L - L')
    remove G_C from G and insert f' into G

return SumProduct(G)
Algorithm: Tensor variable elimination

while any factors in graph G have plates:

L <- maximal factor plate set in G
G_L <- subgraph of G in L
for subgraph G_C in Partition(G_L):
    f <- SumProduct(G_C)
    L' <- plates of all variables of f in G
    f' <- Product(f, L - L')
    remove G_C from G and insert f' into G

return SumProduct(G)
Algorithm: Tensor variable elimination

while any factors in graph G have plates:

L <- maximal factor plate set in G
G_L <- subgraph of G in L
for subgraph G_C in Partition(G_L):
    f <- SumProduct(G_C)
    L' <- plates of all variables of f in G
    f' <- Product(f, L - L')
    remove G_C from G and insert f' into G

return SumProduct(G)
Algorithm: Tensor variable elimination

while any factors in graph $G$ have plates:

$\text{L} \leftarrow$ maximal factor plate set in $G$
$G_L \leftarrow$ subgraph of $G$ in $L$
for subgraph $G_C$ in $\text{Partition}(G_L)$:
    $f \leftarrow \text{SumProduct}(G_C)$
    $L' \leftarrow$ plates of all variables of $f$ in $G$
    $f' \leftarrow \text{Product}(f, L - L')$
    remove $G_C$ from $G$ and insert $f'$ into $G$

return $\text{SumProduct}(G)$
Algorithm: Tensor variable elimination

while any factors in graph G have plates:

L <- maximal factor plate set in G
G_L <- subgraph of G in L
for subgraph G_C in Partition(G_L):
    f <- SumProduct(G_C)
    L' <- plates of all variables of f in G
    f' <- Product(f, L - L')
    remove G_C from G and insert f' into G

return SumProduct(G)
Algorithm: Tensor variable elimination

while any factors in graph G have plates:

L <- maximal factor plate set in G
G_L <- subgraph of G in L
for subgraph G_C in Partition(G_L):
    f <- SumProduct(G_C)
    L’ <- plates of all variables of f in G
    f’ <- Product(f, L – L’)
    remove G_C from G and insert f’ into G

return SumProduct(G)
Algorithm: Tensor variable elimination

while any factors in graph G have plates:

   L <- maximal factor plate set in G
   G_L <- subgraph of G in L
   for subgraph G_C in Partition(G_L):
      f <- SumProduct(G_C)
      L' <- plates of all variables of f in G
      f' <- Product(f, L - L')
      remove G_C from G and insert f' into G

return SumProduct(G)
Algorithm: Tensor variable elimination

while any factors in graph $G$ have plates:

$L \leftarrow$ maximal factor plate set in $G$
$G_L \leftarrow$ subgraph of $G$ in $L$
for subgraph $G_C$ in $\text{Partition}(G_L)$:
  $f \leftarrow \text{SumProduct}(G_C)$
  $L' \leftarrow$ plates of all variables of $f$ in $G$
  $f' \leftarrow \text{Product}(f, L - L')$
  remove $G_C$ from $G$ and insert $f'$ into $G$

return $\text{SumProduct}(G)$
Algorithm: Tensor variable elimination

while any factors in graph G have plates:

L <- maximal factor plate set in G
G_L <- subgraph of G in L
for subgraph G_C in Partition(G_L):
  f <- SumProduct(G_C)
  L’ <- plates of all variables of f in G
  f’ <- Product(f, L - L’)
  remove G_C from G and insert f’ into G

return SumProduct(G)
Algorithm: Computational complexity

**Theorem:** for any PlatedSumProduct instance, the following are equivalent:

1. The PlatedSumProduct instance has complexity polynomial in all plate sizes
2. Tensor variable elimination solves the instance in time polynomial in all plate sizes
**Theorem:** for any PlatedSumProduct instance, the following are equivalent:

1. The PlatedSumProduct instance has complexity polynomial in all plate sizes
2. Tensor variable elimination solves the instance in time polynomial in all plate sizes
3. Neither of the following graph minors appear in the plated factor graph:
Algorithm: Computational complexity

Hard:

\[ \sum_{x_1} \ldots \sum_{x_N} F(x_1, \ldots, x_n) \]

Fully coupled joint distribution

Hard:

\[ \sum_{x_1} \ldots \sum_{x_1} \sum_{y_1} \ldots \sum_{y_J} \prod_{i,j} F_{i,j, x_i, y_j} \]

Restricted Boltzmann Machine
Outline

- Background and Motivation: Discrete Latent Variables
- Models: Plated Factor Graphs
- Inference Algorithm: Tensor Variable Elimination
- Implementation in Pyro
- Experiments and Discussion
Implementation: exploiting existing software

while any factors in graph G have plates:

L <- maximal factor plate set in G
G_L <- subgraph of G in L
for subgraph G_C in Partition(G_L):
    f <- SumProduct(G_C)
    L' <- plates of all variables of f in G
    f' <- Product(f, L – L')
    remove G_C from G and insert f' into G

return SumProduct(G)
Implementation: Integration with the Pyro PPL

@pyro.infer.config_enumerate
def model(z):
  I, J = z.shape
  x = pyro.sample("x", Bernoulli(Px))
  with pyro.plate("I", I):
    y = pyro.sample("y", Bernoulli(Py))
    with pyro.plate("J", J):
      pyro.sample("z", Bernoulli(Pz[x,y]),obs=z)

pyro.ops.contract.einsum("x,iy,ijxy-", F, G, H, plates="ij")
Implementation: Scaling with parallel hardware

**Theorem:** if TVE runs in **sequential time** $T$ when plates all have size 1, then it runs in time $T + O(\log(\text{plate sizes}))$ on a parallel machine with $\text{prod}(\text{plate sizes})$-many processors, with perfect efficiency.

**Experiment:** our GPU-accelerated implementation in Pyro achieves this scaling:
Outline

- Background and Motivation: Discrete Latent Variables
- Models: Plated Factor Graphs
- Inference Algorithm: Tensor Variable Elimination
- Implementation in Pyro
- Experiments and Discussion
Experiments

We evaluated our implementation on three real-world tasks with large datasets, multiple overlapping plates and a wide variety of graphical model structures:

1. Learning generative models of polyphonic music
2. Explaining animal behavior with discrete state-space models
3. Inferring word sentiment from sentence-level labels

Our results illustrate the scalability and ease of model iteration afforded by TVE.
We aim to learn generative models with tractable likelihoods and samplers for three polyphonic music datasets.

We use Pyro to implement a variety of discrete state space models with autoregressive likelihoods and neural transition functions.
Experiment 1: Polyphonic Music Modeling

<table>
<thead>
<tr>
<th>Model</th>
<th>Dataset JSB</th>
<th>Dataset Piano</th>
<th>Dataset Nottingham</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMM</td>
<td>8.28</td>
<td>9.41</td>
<td>4.49</td>
</tr>
<tr>
<td>FHMM</td>
<td>8.40</td>
<td>9.55</td>
<td>4.72</td>
</tr>
<tr>
<td>PFHMM</td>
<td>8.30</td>
<td>9.49</td>
<td>4.76</td>
</tr>
<tr>
<td>2HMM</td>
<td>8.70</td>
<td>9.57</td>
<td>4.96</td>
</tr>
<tr>
<td>arHMM</td>
<td>8.00</td>
<td>7.30</td>
<td>3.29</td>
</tr>
<tr>
<td>arFHMM</td>
<td>8.22</td>
<td>7.36</td>
<td>3.57</td>
</tr>
<tr>
<td>arPFHMM</td>
<td>8.39</td>
<td>9.57</td>
<td>4.82</td>
</tr>
<tr>
<td>ar2HMM</td>
<td>8.19</td>
<td><strong>7.11</strong></td>
<td>3.34</td>
</tr>
<tr>
<td>nnHMM</td>
<td><strong>6.73</strong></td>
<td>7.32</td>
<td><strong>2.67</strong></td>
</tr>
<tr>
<td>nnFHMM</td>
<td>6.86</td>
<td>7.41</td>
<td>2.82</td>
</tr>
<tr>
<td>nnPFHMM</td>
<td>7.07</td>
<td>7.47</td>
<td>2.81</td>
</tr>
<tr>
<td>nn2HMM</td>
<td>6.78</td>
<td>7.29</td>
<td>2.81</td>
</tr>
</tbody>
</table>
Experiment 2: Animal population movement

We model group foraging behavior of a colony of harbour seals using GPS data.

Real-world scientific application where variation between individuals and sexes requires more complex model.

We replicate the original analysis without writing custom inference code.
Experiment 2: Animal population movement

<table>
<thead>
<tr>
<th>Model</th>
<th>AIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>No RE (HMM)</td>
<td>$353 \times 10^3$</td>
</tr>
<tr>
<td>Individual RE</td>
<td>$341 \times 10^3$</td>
</tr>
<tr>
<td>Group RE</td>
<td>$342 \times 10^3$</td>
</tr>
<tr>
<td>Individual+Group RE</td>
<td>$341 \times 10^3$</td>
</tr>
</tbody>
</table>
Experiment 3: word sentiment from weak supervision

An example sentence from the Sentihood dataset:

“Other places to look at in South London are Streatham (good range of shops and restaurants, maybe a bit far out of central London but you get more for your money) Brixton (good transport links, trendy, can be a bit edgy) Clapham (good transport, good restaurants/pubs, can feel a bit dull, expensive) ...”

A synthetic example with Sentihood-style annotations:

<table>
<thead>
<tr>
<th>Sentence</th>
<th>Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>location1 is very safe and location2 is too far</td>
<td>(location1,safety,Positive)</td>
</tr>
<tr>
<td></td>
<td>(location1,transit-location,None)</td>
</tr>
<tr>
<td></td>
<td>(location2,safety,None)</td>
</tr>
<tr>
<td></td>
<td>(location2,transit-location,Negative)</td>
</tr>
</tbody>
</table>

(Saeidi et al 2016)
Experiment 3: word sentiment from weak supervision

Neural CRF inference and learning in one line of Python code:

\[
Z, \ hy = \text{pyro.ops.contract.einsum}(\text{"ntz,ntyz,ny->n,ny"}, \ F, \ G, \ P_Y, \ \text{plates="t"})
\]
Find tutorials, examples, and more online at pyro.ai

Install Pyro and get started today!

pip install -U pyro-ppl
Algorithm: Tensor variable elimination

while any factors in graph G have plates:

L <- maximal factor plate set in G
G_L <- subgraph of G in L
for subgraph G_C in Partition(G_L):
    f <- SumProduct(G_C)
    L' <- plates of all variables of f in G
    f' <- Product(f, L - L')
    remove G_C from G and insert f' into G

return SumProduct(G)
Algorithm: Tensor variable elimination

while any factors in graph G have plates:

L <- maximal factor plate set in G
G_L <- subgraph of G in L
for subgraph G_C in Partition(G_L):
    f <- SumProduct(G_C)
    L' <- plates of all variables of f in G
    f' <- Product(f, L - L')
    remove G_C from G and insert f' into G

return SumProduct(G)