Tensor Variable Elimination for Plated Factor Graphs

Fritz Obermeyer*, Eli Bingham*, Martin Jankowiak*,
Justin Chiu, Neeraj Pradhan, Alexander Rush, Noah Goodman

Uber AI harvardnlp

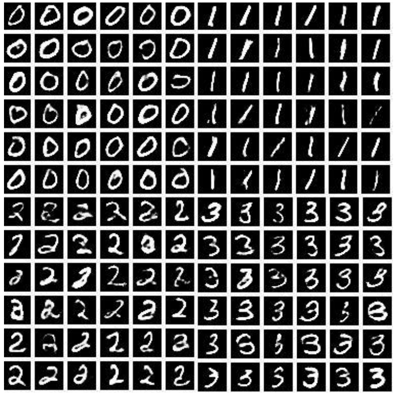
Outline

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

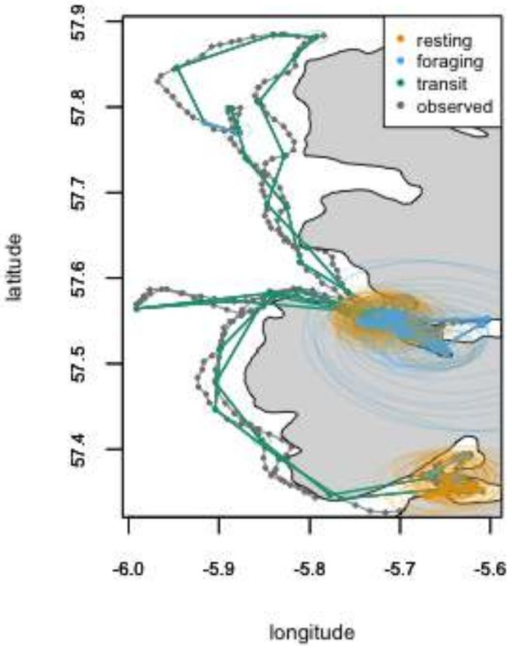
Outline

- 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)

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

(Obermeyer et al. 2019)

Learning and inference with discrete latent variables

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

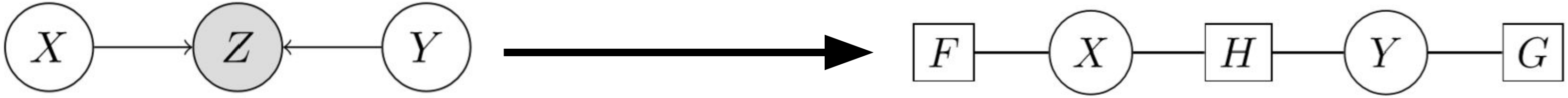
Exact inference is theoretically tractable in many popular discrete latent variable models.

Algorithms and software have not kept up with growth of models and data, and integration with deep learning is difficult and time-consuming.

Background: Factor graphs

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.

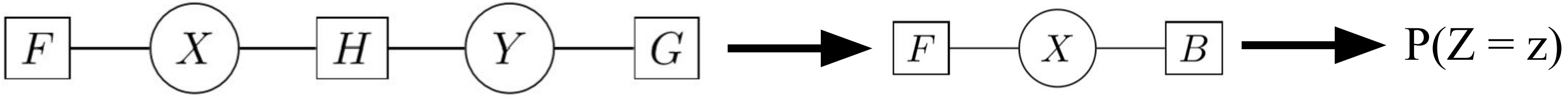


Background: Factor graph inference

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

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

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

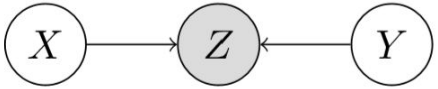


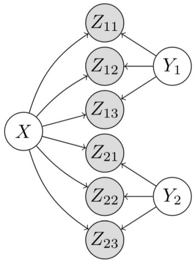
Outline

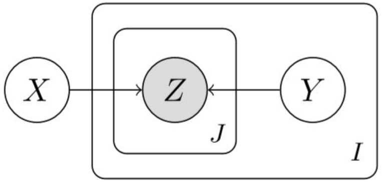
- Background and Motivation: Discrete Latent Variables
- Models: Plated Factor Graphs
- Inference Algorithm: Tensor Variable Elimination
- Implementation in Pyro
- Experiments and Discussion

Focus: Plated factor graphs

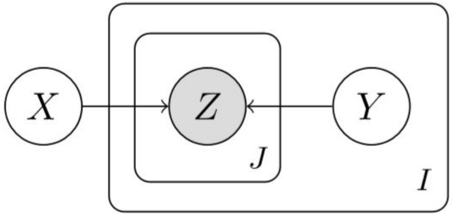
Plates represent repeated structure in graphical models:

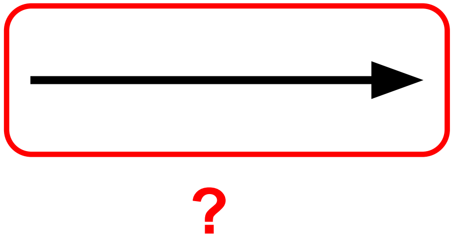


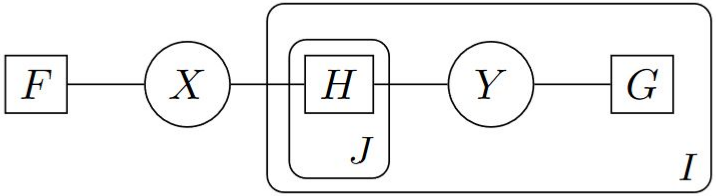




Can we use plates to represent repeated structure in variable elimination algorithms?



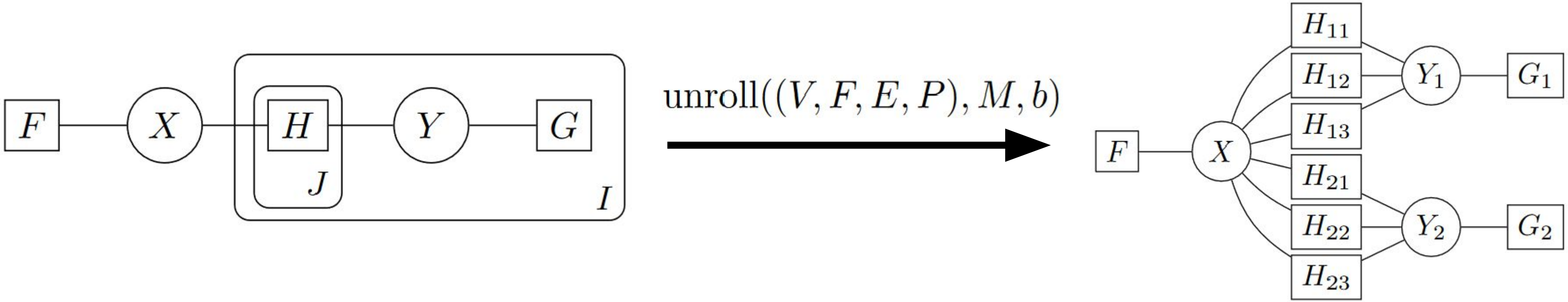




Plated factor graph inference

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:

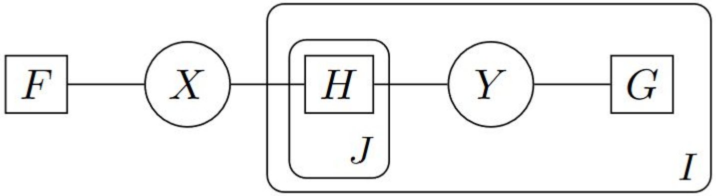
$$\textsc{PlatedSumProduct}(G, M) \equiv \textsc{SumProduct}(F', V')$$



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 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)
```

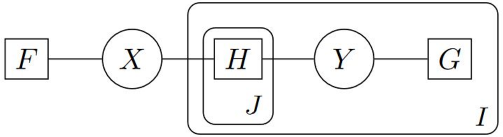
We rely on three plate-aware subroutines to avoid unrolling:

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

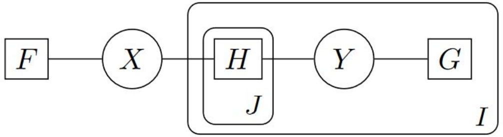
Algorithm: Tensor variable elimination



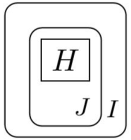

```
while any factors in graph G have plates:

  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)
```

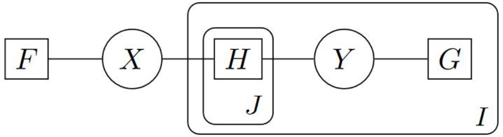


$$\{\} < \{I\} < \{I, J\}$$

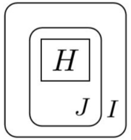


Algorithm: Tensor variable elimination

```
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)
```



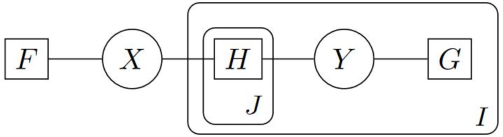
$$\{\} < \{\, I \,\} < \{\, I, J \,\}$$



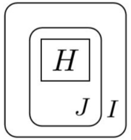
$$A_{ixy_i} = \prod_j H_{ijxy_i}$$

Algorithm: Tensor variable elimination

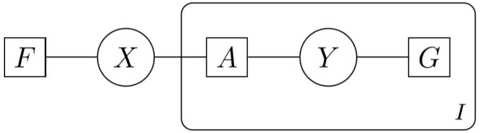
```
while any factors in graph G have plates:

  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)
```



$$\{\} < \{ I \} < \{ I, J \}$$

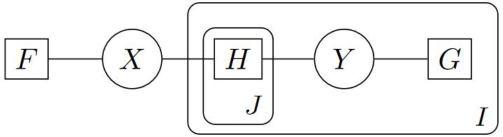


$$A_{ixy_i} = \prod_j H_{ijxy_i}$$

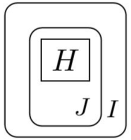


Algorithm: Tensor variable elimination

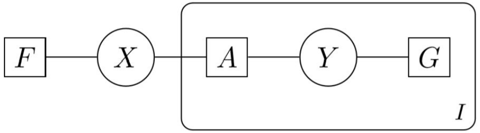
```
while any factors in graph G have plates:

  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)
```



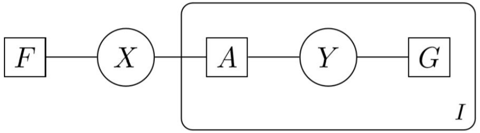
$$\{\} < \{\, I \,\} < \{\, I, J \,\}$$



$$A_{ixy_i} = \prod_j H_{ijxy_i}$$



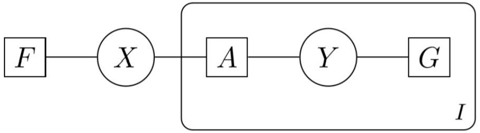
Algorithm: Tensor variable elimination



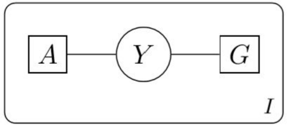

```
while any factors in graph G have plates:

  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)
```

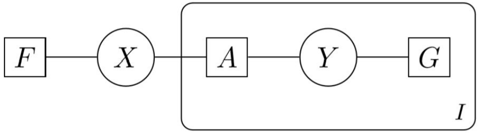


$\{\,\} < \{\,I\,\}$

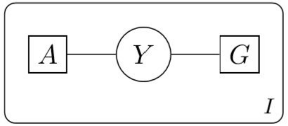


Algorithm: Tensor variable elimination

```
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)
```



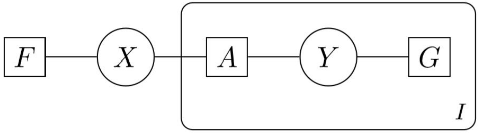
$\{\,\} < \{\,I\,\}$



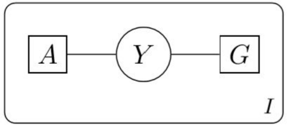
$B_x = \prod_i \sum_{y_i} A_{ixy_i} G_{iy_i}$

Algorithm: Tensor variable elimination

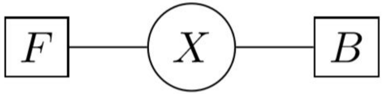
```
while any factors in graph G have plates:

  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)
```



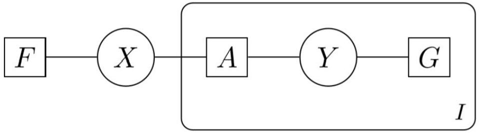
$\{\,\} < \{\,I\,\}$



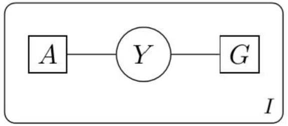
$$B_x = \prod_i \sum_{y_i} A_{ixy_i} G_{iy_i}$$



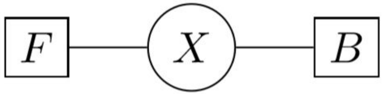
Algorithm: Tensor variable elimination



$$\{\,\} < \{\,I\,\}$$



$$B_x = \prod_i \sum_{y_i} A_{ixy_i} G_{iy_i}$$




```
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

<u>Theorem:</u> for any `PlatedSumProduct` instance, the following are equivalent:

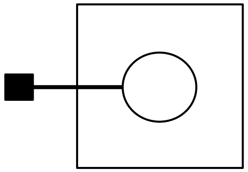
1. The `PlatedSumProduct` instance has complexity polynomial in all plate sizes
2. Tensor variable elimination solves the instance in time polynomial in all plate sizes

Algorithm: Computational complexity

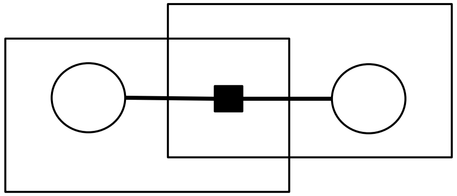
<u>Theorem:</u> for any `PlatedSumProduct` instance, the following are equivalent:

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:

Hard:

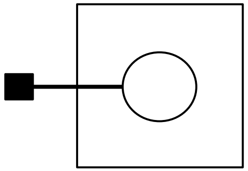


Hard:



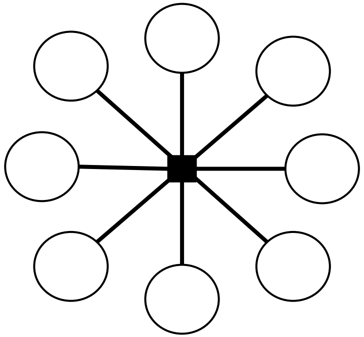
Algorithm: Computational complexity

Hard:

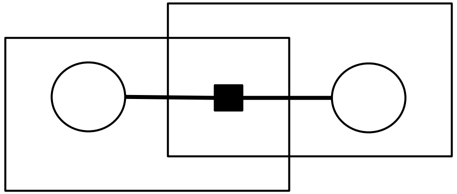


$$\sum_{x_1} \cdots \sum_{x_N} F(x_1, \ldots, x_n)$$

Fully coupled joint distribution

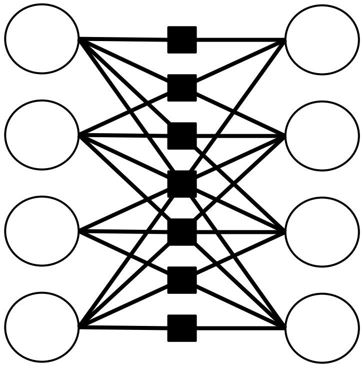


Hard:



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

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

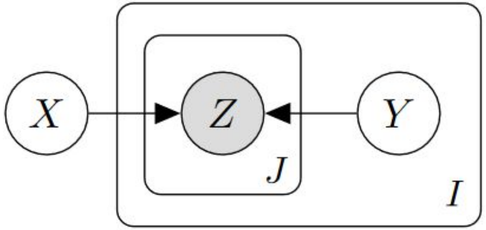
Implementation: exploiting existing software

```
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)
```

High-performance, parallelized **`SumProduct`** and **`Product`** available as tensor contractions (`einsum` and `prod` in NumPy)

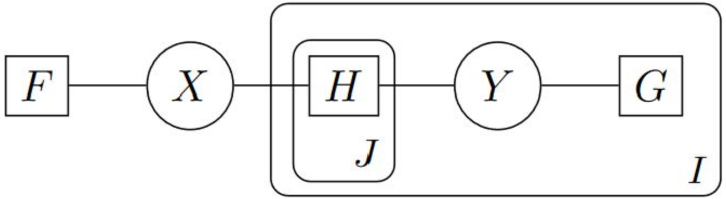
Implementation: Integration with the Pyro PPL

High-level interface for specifying generative discrete latent variable models:




```
@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)
```

Low-level interface for specifying discrete plated factor graphs directly:

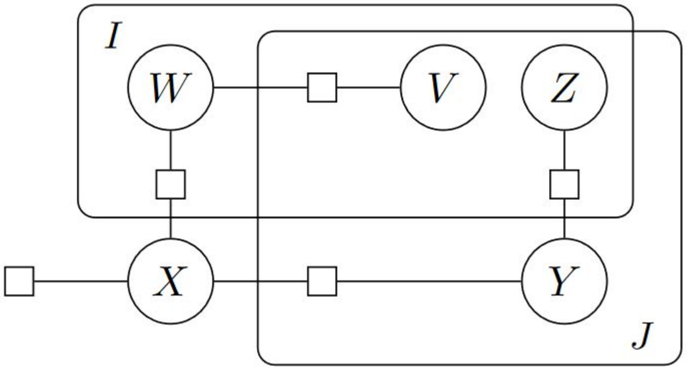


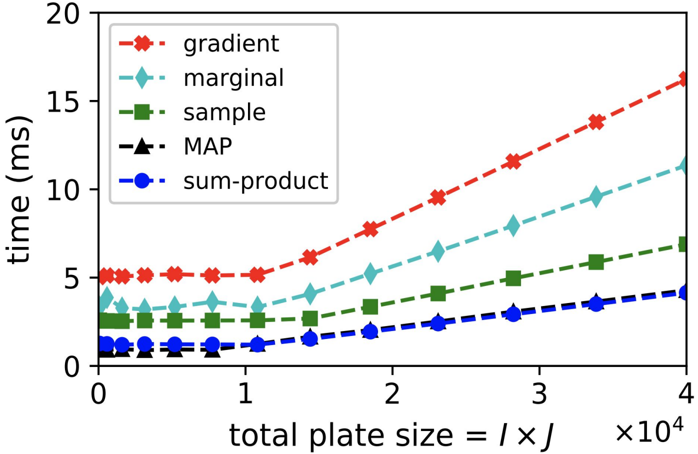

```
pyro.ops.contract.einsum(
  "x,iy,ijxy->",
  F, G, H,
  plates="ij"
)
```

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(plate sizes))* on a parallel machine** with *prod(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

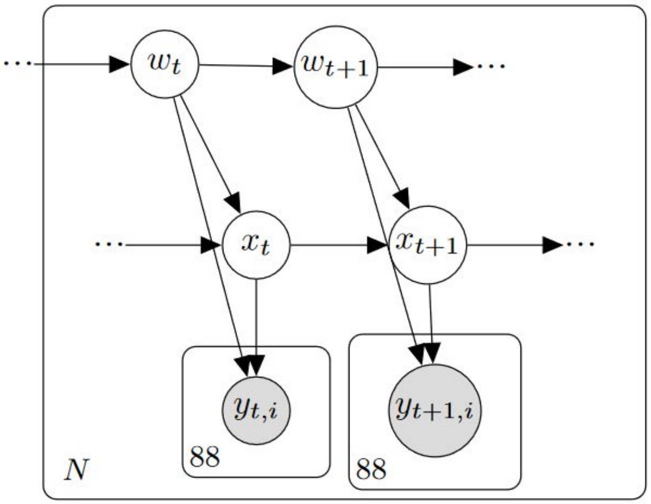
Experiments

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.

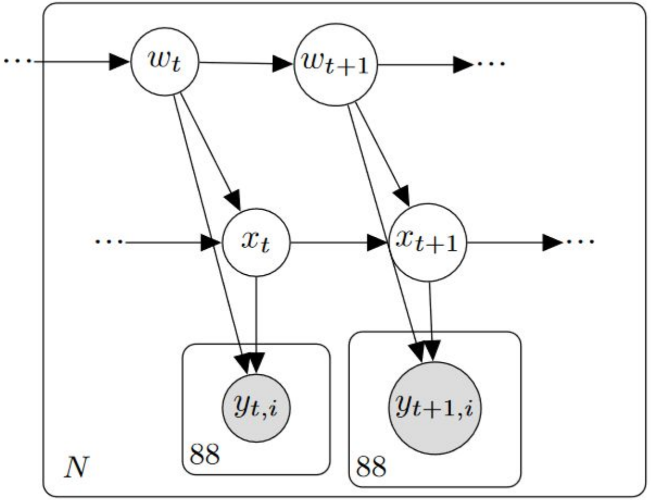
Experiment 1: Polyphonic Music Modeling



We aim to learn generative models with tractable likelihoods and samplers for three polyphonic music datasets

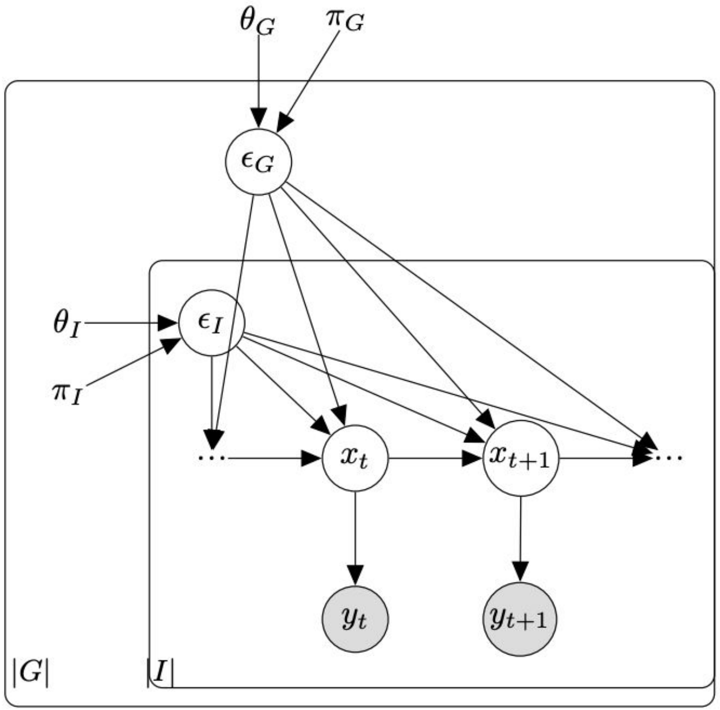
We use Pyro to implement a variety of discrete state space models with autoregressive likelihoods and neural transition functions

Experiment 1: Polyphonic Music Modeling



	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

Experiment 2: Animal population movement

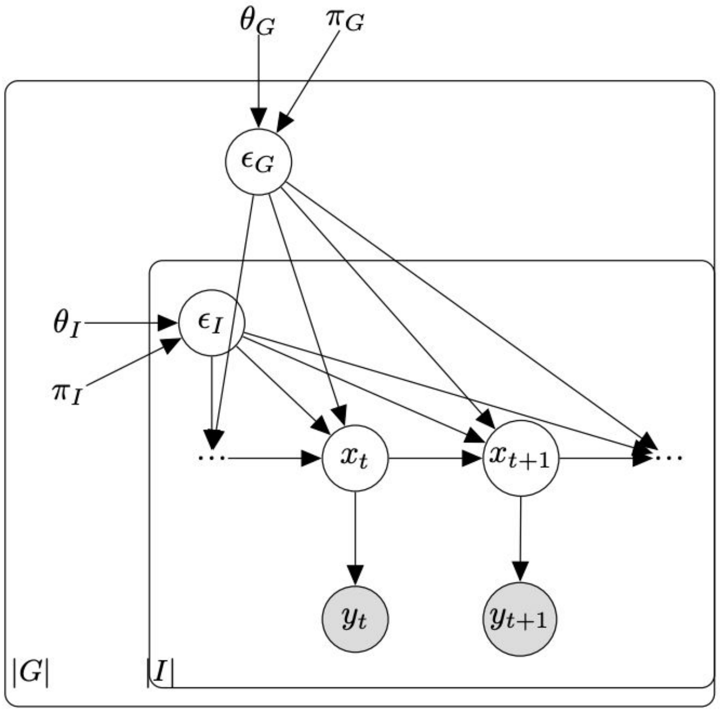


We model group foraging behavior of a colony of harbour seals using GPS data

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



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

Experiment 3: word sentiment from weak supervision

An example sentence from the Sentihood dataset:

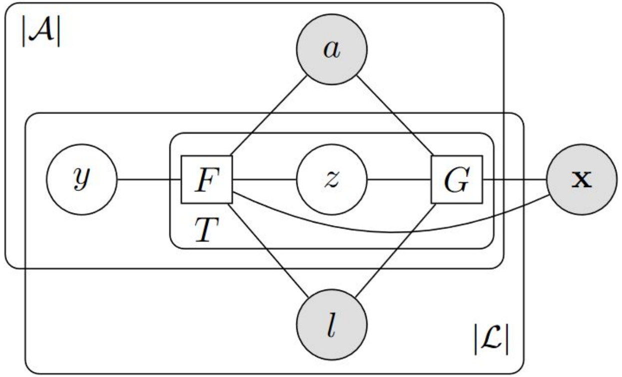
*"Other places to look at in South London are **Streatham** (good range of shops and restaurants, maybe a bit far out of central London but you get more for your money) **Brixton** (good transport links, trendy, can be a bit edgy) **Clapham** (good transport, good restaurants/pubs, can feel a bit dull,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

A synthetic example with Sentihood-style annotations:

Sentence	**Labels**
location1 is very safe and **location2** is too far	(location1,safety,Positive) (location1,transit-location,None) (location2,safety,None) (location2,transit-location,Negative)

(Saeidi et al 2016)

Experiment 3: word sentiment from weak supervision



	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

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

```
Z, hy = pyro.ops.contract.einsum("ntz,ntyz,ny->n,ny", F, G, P_Y, plates="t")
```

<Your experiment here>

Find tutorials, examples, and more online at

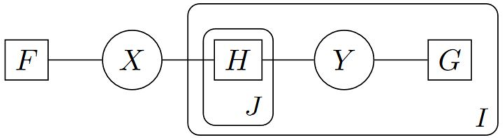
pyro.ai

Install Pyro and get started today!

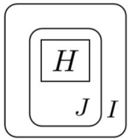
```
pip install -U pyro-ppl
```

Algorithm: Tensor variable elimination

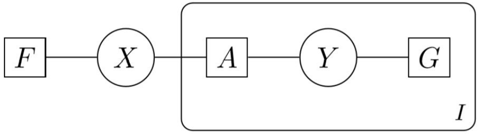
```
while any factors in graph G have plates:

  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)
```



$$\{\} < \{\, I \,\} < \{\, I, J \,\}$$

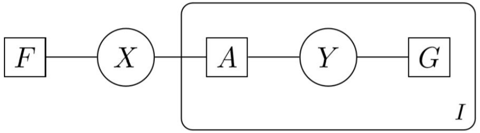


$$A_{ixy_i} = \prod_j H_{ijxy_i}$$

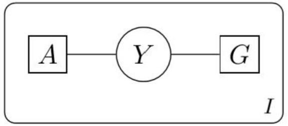


Algorithm: Tensor variable elimination

```
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)
```



$\{\,\} < \{\,I\,\}$



$$B_x = \prod_i \sum_{y_i} A_{ixy_i} G_{iy_i}$$