Composing graphical models with neural networks like chocolate and peanut butter

https://youtu.be/O7oD_oX-Gio
Composing graphical models with neural networks like chocolate and peanut butter

https://youtu.be/O7oD_oX-Gio

or

Graphical models and exponential families in the age of differentiable programming

David Duvenaud
Alex Wilitchko
Matthew D. Hoffman
Dustin Tran
Scott Linderman
Sandeep Robert Datta
Ryan P. Adams

Matthew J Johnson (mattjj@google.com)
July 22 2019 @ UAI 2019
Goals
Goals

1. **Motivate** why PGMs + DNNs are a **revolution** waiting to happen
Goals

1. **Motivate** why PGMs + DNNs are a revolution waiting to happen

2. **Survey the fundamentals** of PGMs and exponential families so that you have a broad view of the territory
Goals

1. **Motivate** why PGMs + DNNs are a revolution waiting to happen

2. **Survey the fundamentals** of PGMs and exponential families so that you have a broad view of the territory

3. Show how to **unify many models and algorithms** in a framework that lets you **leverage automatic differentiation**
Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen

2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory

3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation

4. Make SVAEs and related PGM + DNN architectures super obvious so that you can invent better ones
Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen
2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory
3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation
4. Make SVAEs and related PGM + DNN architectures super obvious so that you can invent better ones

Non-goals

1. Cover the recent literature on PGMs + DNNs
2. Unpack all the technical details
Goals

1. **Motivate** why PGMs + DNNs are a revolution waiting to happen

2. **Survey the fundamentals** of PGMs and exponential families so that you have a broad view of the territory

3. Show how to **unify many models and algorithms** in a framework that lets you **leverage automatic differentiation**

4. Make **SVAEs** and related PGM + DNN architectures **super obvious** so that you can invent better ones
dart
pause
rear

/b/ /ax/ /n/ /ae/ /n/ /ax/

image manifold
image manifold
depth
video
depth
video

image
manifold
depth video

image manifold
depth video

image manifold
In this section, we describe a model that uses Recurrent LSTM units.

Neural Nets (RNNs) made of LSTM units to do unsupervised learning.

The first source is the current frame. Since derivatives discriminate between containing features.

The optimizer's role is to assign credit to or criticize over sums, the error derivatives don't vanish quickly.

The state of the encoder LSTM after the last input has been read is the representation of the input video. The decoder can be of two kinds – conditional or unconditional. A conditional decoder receives the last generated output. In that case all the information about the input is contained in the video. However, an important question for learning good features is why should this learn good features?

Why should it learn good features?

The decoder can be of two kinds – conditional or unconditional. A conditional decoder receives the last generated output. In that case all the information about the input is contained in the video. However, an important question for learning good features is why should this learn good features?

The state of the encoder LSTM after the last input has been read is the representation of the input video. The decoder can be of two kinds – conditional or unconditional. A conditional decoder receives the last generated output. In that case all the information about the input is contained in the video. However, an important question for learning good features is why should this learn good features?

The state of the encoder LSTM after the last input has been read is the representation of the input video. The decoder can be of two kinds – conditional or unconditional. A conditional decoder receives the last generated output. In that case all the information about the input is contained in the video. However, an important question for learning good features is why should this learn good features?

The state of the encoder LSTM after the last input has been read is the representation of the input video. The decoder can be of two kinds – conditional or unconditional. A conditional decoder receives the last generated output. In that case all the information about the input is contained in the video. However, an important question for learning good features is why should this learn good features?
Recurrent neural networks? [1,2,3]

![Figure 1. LSTM unit](image1)

Probabilistic graphical models? [4,5,6]

![Figure 2. LSTM Autoencoder Model](image2)

supervised learning
Probabilistic graphical models

Deep learning
Probabilistic graphical models  Deep learning

+ structured representations

+ priors and uncertainty

– rigid assumptions may not fit

– feature engineering
Probabilistic graphical models

+ structured representations

+ priors and uncertainty
  - rigid assumptions may not fit
  - feature engineering

Deep learning

+ arbitrary inference queries

+ data and computational efficiency

  - but only in rigid model classes
<table>
<thead>
<tr>
<th>Probabilistic graphical models</th>
<th>Deep learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ structured representations</td>
<td>– neural net “goo”</td>
</tr>
<tr>
<td>+ priors and uncertainty</td>
<td>– difficult parameterization</td>
</tr>
<tr>
<td>– rigid assumptions may not fit</td>
<td>+ flexible, high capacity</td>
</tr>
<tr>
<td>– feature engineering</td>
<td>+ feature learning</td>
</tr>
<tr>
<td>+ arbitrary inference queries</td>
<td></td>
</tr>
<tr>
<td>+ data and computational efficiency</td>
<td></td>
</tr>
<tr>
<td>– but only in rigid model classes</td>
<td></td>
</tr>
<tr>
<td>Probabilistic graphical models</td>
<td>Deep learning</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>+ structured representations</td>
<td>– neural net “goo”</td>
</tr>
<tr>
<td>+ priors and uncertainty</td>
<td>– difficult parameterization</td>
</tr>
<tr>
<td>– rigid assumptions may not fit</td>
<td>+ flexible, high capacity</td>
</tr>
<tr>
<td>– feature engineering</td>
<td>+ feature learning</td>
</tr>
<tr>
<td>+ arbitrary inference queries</td>
<td>– limited inference queries</td>
</tr>
<tr>
<td>+ data and computational efficiency</td>
<td>– data- and compute-hungry</td>
</tr>
<tr>
<td>– but only in rigid model classes</td>
<td>+ recognition networks learn how to do inference</td>
</tr>
</tbody>
</table>
Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen

2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory

3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation

4. Make SVAEs and related PGM + DNN architectures super obvious so that you can invent better ones
Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen

2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory

3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation

4. Make SVAEs and related PGM + DNN architectures super obvious so that you can invent better ones
Graphs
Independence of RVs

\[ X_1 \perp X_3 \mid X_2 \]
Independence of RVs

\[ X_1 \perp X_3 \mid X_2 \]

Algebraic structure in density

\[ p(x) = \frac{1}{Z} \prod_C \psi_C(x_C) \]
Independence of RVs

\[ X_1 \perp X_3 \mid X_2 \]

Algebraic structure in density

\[ p(x) = \frac{1}{Z} \prod_C \psi_C(x_C) \]
\[ G = (V, E) \]

\[ V = \{1, 2, \ldots, n\} \]

\[ E \subseteq \left\{ \{u, v\} : u, v \in V \right\} \]
\[ G = (V, E) \]

\[ V = \{1, 2, \ldots, n\} \]

\[ E \subseteq \{ \{u, v\} : u, v \in V \} \]

\[ V = \{1, 2, 3\} \]

\[ E = \{ \{1, 2\}, \{2, 3\} \} \]
Def \( X = \{X_v\} \) is Markov on \( G \) iff for disjoint \( A, B, C \subseteq V \) when \( C \) disconnects \( A \) from \( B \) we have \( X_A \perp X_B \mid X_C \).
\textbf{Def} \( X = \{X_v\}_{v \in V} \) is Markov on \( G \) iff for disjoint \( A, B, C \subseteq V \) when \( C \) disconnects \( A \) from \( B \) we have \( X_A \perp X_B \mid X_C \).

\textit{disconnects} means no path from \( A \) to \( B \) after removing \( C \).
Def $X = \{X_v\}_{v \in V}$ is Markov on $G$ iff for disjoint $A, B, C \subseteq V$ when $C$ disconnects $A$ from $B$ we have $X_A \perp X_B \mid X_C$.

disconnects means no path from $A$ to $B$ after removing $C$. 

![Diagram of connected nodes 1, 2, 3 with edges 1-2, 2-3, 1-3, 2-1, 3-2]
Def $X = \{X_v\}_{v \in V}$ is Markov on $G$ iff for disjoint $A, B, C \subseteq V$ when $C$ disconnects $A$ from $B$ we have $X_A \perp X_B | X_C$.

disconnects means no path from $A$ to $B$ after removing $C$.

$\implies X_1 \perp X_3 | X_2$

$\implies X_1 \perp X_3 | X_2, X_4$
**Def** $X = \{X_v\}_{v \in V}$ is *Markov on* $G$ iff for disjoint $A, B, C \subseteq V$ when $C$ disconnects $A$ from $B$ we have $X_A \perp X_B \ | \ X_C$.

Disconnects means no path from $A$ to $B$ after removing $C$.

\[\begin{array}{c}
\text{Diagram 1:} \\
1 \quad 2 \quad 3 \\
\quad 1 \quad 2 \quad 3 \quad 4 \\
\end{array}\]

\[\begin{array}{c}
\text{Diagram 2:} \\
1 \quad 2 \quad 3 \\
\quad 1 \quad 2 \quad 3 \quad 4 \\
\end{array}\]
Def \( X = \{ X_v \}_v \subseteq V \) factorizes on \( G \) iff \( p(x) \propto \prod_C \psi_C(x_C) \) where the product is over cliques of \( G \).
\textbf{Def} \( X = \{ X_v \}_{v \in V} \) factorizes on \( G \) iff \( p(x) \propto \prod_C \psi_C(x_C) \) where the product is over cliques of \( G \).

\[
\Rightarrow \quad p(x) \propto \psi_{12}(x_1, x_2)\psi_{23}(x_2, x_3)
\]
**Def** $X = \{X_v\}_{v \in V}$ factorizes on $G$ iff $p(x) \propto \prod_C \psi_C(x_C)$ where the product is over cliques of $G$. 

\[
\begin{array}{c}
\text{Factorization of } p(x) \\
\text{on } G
\end{array}
\]
Def $X = \{X_v\}_{v \in V}$ factorizes on $G$ iff $p(x) \propto \prod_C \psi_C(x_C)$ where the product is over cliques of $G$.

$\implies p(x) \propto \psi_{12}(x_1, x_2)\psi_{23}(x_2, x_3)$

$\implies p(x) \propto \psi_{123}(x_1, x_2, x_3)\psi_{134}(x_1, x_3, x_4)$

$X_1 \perp X_3 \mid X_2 \iff p(x) = \frac{1}{Z} \prod_C \psi_C(x_C)$
Def $X = \{X_v\}_{v \in V}$ is Markov on $G$ iff for disjoint $A, B, C \subseteq V$ when $C$ disconnects $A$ from $B$ we have $X_A \perp X_B | X_C$.

Def $X = \{X_v\}_{v \in V}$ factorizes on $G$ iff $p(x) \propto \prod_C \psi_C(x_C)$ where the product is over cliques of $G$. 

$X_1 \perp X_3 | X_2 \iff p(x) = \frac{1}{Z} \prod_{C} \psi_C(x_C)$
Def $X = \{X_v\}_{v \in V}$ is Markov on $G$ iff for disjoint $A, B, C \subseteq V$ when $C$ disconnects $A$ from $B$ we have $X_A \perp X_B | X_C$.

Def $X = \{X_v\}_{v \in V}$ factorizes on $G$ iff $p(x) \propto \prod_C \psi_C(x_C)$ where the product is over cliques of $G$.

Markov on $G \iff$ factorizes on $G$
Def $X = \{X_v\}_{v \in V}$ is Markov on $G$ iff for disjoint $A, B, C \subseteq V$ when $C$ disconnects $A$ from $B$ we have $X_A \perp X_B \mid X_C$.

Def $X = \{X_v\}_{v \in V}$ factorizes on $G$ iff $p(x) \propto \prod_C \psi_C(x_C)$ where the product is over cliques of $G$.

Markov on $G \quad \Rightarrow \quad$ factorizes on $G$

... but they are the same if $p(x) > 0$. 

$p(x) = \frac{1}{Z} \prod_C \psi_C(x_C)$
A good way to ensure $p(x) > 0$ is to have $p(x) = \exp(-E(x))$.

\[
p(x) = \frac{1}{Z} \prod_C \psi_C(x_C)
\]

\[
E(x) = \log Z + \sum_C \phi_C(x_C)
\]
A good way to ensure \( p(x) > 0 \) is to have \( p(x) = \exp(-E(x)) \).

\[
p(x) = \frac{1}{Z} \prod_C \psi_C(x_C)
\]

\[
E(x; \eta) = \log Z + \sum_C \eta_C \cdot \phi_C(x_C)
\]
Independence of RVs

\[ X_1 \perp X_3 \mid X_2 \]

Algebraic structure in density

\[ p(x) = \prod_{v \in V} p(x_v \mid x_{\text{pa}(v)}) \]
$G = (V, E)$

$V = \{1, 2, \ldots, n\}$

$E \subseteq V \times V$
\[ G = (V, E) \]
\[ V = \{1, 2, \ldots, n\} \]
\[ E \subseteq V \times V \]

\[ V = \{1, 2, 3\} \]
\[ E = \{ (2, 1), (2, 3) \} \]
Def $X = \{X_v\}_{v \in V}$ is Markov on $G$ iff for disjoint $A, B, C \subseteq V$ when $C$ d-separates $A$ from $B$ we have $X_A \perp X_B \mid X_C$.

d-separates means no unblocked undirected path from $A$ to $B$. 

\[ p(x) = \frac{1}{Z} \prod_{e \in E} \psi_G(x_e) \]
Def $X = \{X_v\}_{v \in V}$ is Markov on $G$ iff for disjoint $A, B, C \subseteq V$ when $C$ d-separates $A$ from $B$ we have $X_A \perp X_B \mid X_C$.

d-separates means no unblocked undirected path from $A$ to $B$. 
Def $X = \{X_v\}_{v \in V}$ is Markov on $G$ iff for disjoint $A, B, C \subseteq V$ when $C$ d-separates $A$ from $B$ we have $X_A \perp X_B \mid X_C$.

d-separates means no unblocked undirected path from $A$ to $B$. 

![Diagram showing d-separation concept](image)
Def $X = \{X_v\}_{v \in V}$ is Markov on $G$ iff for disjoint $A, B, C \subseteq V$ when $C$ d-separates $A$ from $B$ we have $X_A \perp X_B | X_C$.

d-separates means no unblocked undirected path from $A$ to $B$. 
Def \( X = \{X_v\}_{v \in V} \) is Markov on \( G \) iff for disjoint \( A, B, C \subseteq V \) when \( C \text{ d-separates} A \) from \( B \) we have \( X_A \perp X_B | X_C \).

d-separates means no unblocked undirected path from \( A \) to \( B \).

\[
\begin{align*}
 & 1 \quad 2 \quad 3 \\
\implies \\
X_1 \perp X_3 | X_2
\end{align*}
\]
Def $X = \{X_v\}_{v \in V}$ is Markov on $G$ iff for disjoint $A, B, C \subseteq V$ when $C$ \underline{d-separates} $A$ from $B$ we have $X_A \perp X_B \mid X_C$.

d-separates means no unblocked undirected path from $A$ to $B$.

\[\begin{array}{c}
1 \\ 2 \\ 3
\end{array}\]
\[\begin{array}{c}
\Rightarrow \\
X_1 \perp X_3 \mid X_2
\end{array}\]

\[\begin{array}{c}
1 \\ 2 \\ 3
\end{array}\]
\[\begin{array}{c}
\Rightarrow \\
X_1 \perp X_3
\end{array}\]
Def $X = \{X_v\}_{v \in V}$ is Markov on $G$ iff for disjoint $A, B, C \subseteq V$ when $C$ \underline{d-separates} $A$ from $B$ we have $X_A \perp X_B \mid X_C$.

\underline{d-separates} means no unblocked undirected path from $A$ to $B$. 

\begin{align*}
\text{\begin{tikzpicture}[->,>=stealth',shorten >=1pt,auto,node distance=1.5cm,thick,scale=0.8]
  \node[fill=white,draw=white,circle] (1) at (0,0) {1};
  \node[fill=white,draw=white,circle] (2) at (1,0) {2};
  \node[fill=white,draw=white,circle] (3) at (2,0) {3};
  \path
    (1) edge (2)
    (2) edge (3)
    (1) edge (3);
\end{tikzpicture}} \implies X_1 \perp X_3 \mid X_2
\end{align*}

\begin{align*}
\text{\begin{tikzpicture}[->,>=stealth',shorten >=1pt,auto,node distance=1.5cm,thick,scale=0.8]
  \node[fill=white,draw=white,circle] (1) at (0,0) {1};
  \node[fill=white,draw=white,circle] (2) at (1,0) {2};
  \node[fill=white,draw=white,circle] (3) at (2,0) {3};
  \path
    (1) edge (2)
    (2) edge (3)
    (1) edge (3);
\end{tikzpicture}} \implies X_1 \perp X_3
\end{align*}

\begin{align*}
\text{\begin{tikzpicture}[->,>=stealth',shorten >=1pt,auto,node distance=1.5cm,thick,scale=0.8]
  \node[fill=white,draw=white,circle] (1) at (0,0) {1};
  \node[fill=white,draw=white,circle] (2) at (1,0) {2};
  \node[fill=white,draw=white,circle] (3) at (2,0) {3};
  \path
    (1) edge (2)
    (2) edge (3)
    (1) edge (3);
\end{tikzpicture}} \cancel{\implies X_1 \perp X_3 \mid X_2}
\end{align*}
Def $X = \{X_v\}_{v \in V}$ factorizes on $G$ iff $p(x) = \prod_{v \in V} p(x_v \mid x_{pa(v)})$. 

$\psi_C(x_C) = \prod C_p(x_C)$
Def \( X = \{X_v\}_{v \in V} \) factorizes on \( G \) iff \( p(x) = \prod_{v \in V} p(x_v \mid x_{pa(v)}) \).

\[
1 \quad \rightarrow \quad 2 \quad \rightarrow \quad 3 \quad \implies \quad p(x) = p(x_2)p(x_1 \mid x_2)p(x_3 \mid x_2)
\]
Def $X = \{X_v\}_{v \in V}$ factorizes on $G$ iff $p(x) = \prod_{v \in V} p(x_v \mid x_{pa(v)}).$

\[
\begin{align*}
\begin{tikzpicture}
  \node[circle,draw,fill=cyan!20] (1) at (0,0) {$1$};
  \node[circle,draw,fill=green!20] (2) at (1,0) {$2$};
  \node[circle,draw,fill=red!20] (3) at (2,0) {$3$};
  \draw[->] (1) to (2);
  \draw[->] (2) to (3);
\end{tikzpicture} & \implies p(x) = p(x_2)p(x_1 \mid x_2)p(x_3 \mid x_2)\\
\begin{tikzpicture}
  \node[circle,draw,fill=cyan!20] (1) at (0,0) {$1$};
  \node[circle,draw,fill=green!20] (2) at (1,0) {$2$};
  \node[circle,draw,fill=red!20] (3) at (2,0) {$3$};
  \draw[->] (1) to (2);
  \draw[->] (2) to (3);
\end{tikzpicture} & \implies p(x) = p(x_1)p(x_3)p(x_2 \mid x_1, x_3)
\end{align*}
\]
Directed models often appear with generative models

- Ancestral sampling
- For Gaussians, like having a Cholesky factorization
Directed models often appear with generative models

- Ancestral sampling
- For Gaussians, like having a Cholesky factorization

Undirected models often appear in inference

- For Gaussians, like solving the linear system
  \[ J\mu = h \]
Conditional random fields (CRFs) are PGMs where potentials depend on exogenous data.

\[ p(y; x) \propto \psi_1(y_1; x_1)\psi_{12}(y_1, y_2)\psi_2(y_2; x_2)\psi_{23}(y_2, y_3)\psi_3(y_3; x_3) \]
Conditional random fields (CRFs) are PGMs where potentials depend on exogenous data.

\[ p(y; x) \propto \psi_1(y_1; x_1)\psi_{12}(y_1, y_2)\psi_2(y_2; x_2)\psi_{23}(y_2, y_3)\psi_3(y_3; x_3) \]

\[ \psi_n(y_n; x_n) = \psi(y_n; f(x_n, \phi)) \] for neural network \( f(\cdot, \phi) \)
Conditional random fields (CRFs) are PGMs where potentials depend on exogenous data.
Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen

2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory

3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation

4. Make SVAEs and related PGM + DNN architectures super obvious so that you can invent better ones
Def An exponential family of densities is
Def An **exponential family of densities** is

\[ p(x) = \exp\left(t(x) + A(\theta)\right) \]

defined by statistic function \( t : \mathcal{X} \rightarrow W \)
Def An *exponential family of densities* is

\[ t(x) \]

defined by statistic function \( t : \mathcal{X} \to \mathbb{W} \) finite-dim real vector space
Def  An *exponential family of densities* is

\[ \text{exp}(\langle \eta, t(x) \rangle) \]

defined by statistic function \( t : \mathcal{X} \rightarrow W \) finite-dim real vector space indexed by natural parameter \( \eta \in \Theta \subseteq W \)
Def An exponential family of densities is

\[ p(x; \eta) = \exp(\langle \eta, t(x) \rangle - A(\eta)) \]

defined by statistic function \( t : \mathcal{X} \to W \) finite-dim real vector space

indexed by natural parameter \( \eta \in \Theta \subseteq W \)

defines a log normalizer \( A : \Theta \to \mathbb{R} \)

\[ A(\eta) \triangleq \log \int_{\mathcal{X}} \exp(\langle \eta, t(x) \rangle) \, dx \]
Def An exponential family of densities is
\[ p(x; \eta) = \exp(\langle \eta, t(x) \rangle - A(\eta)) \]
defined by statistic function \( t : \mathcal{X} \rightarrow \mathbb{W} \) finite-dim real vector space
indexed by natural parameter \( \eta \in \Theta \subseteq W \)
defines a log normalizer \( A : \Theta \rightarrow \mathbb{R} \)
\[ A(\eta) \triangleq \log \int_{\mathcal{X}} \exp(\langle \eta, t(x) \rangle) \, dx \]
\[ \Theta \triangleq \{ \eta \in W : A(\eta) < \infty \} \]
Def An exponential family of densities is

\[ p(x ; \eta) = \exp(\langle \eta, t(x) \rangle - A(\eta)) \]

\[ A(\eta) \triangleq \log \int_{\mathcal{X}} \exp(\langle \eta, t(x) \rangle) \, dx \]
Def An exponential family of densities is

\[ p(x; \eta) = \exp(\langle \eta, t(x) \rangle - \mathcal{A}(\eta)) \]

\[ \mathcal{A}(\eta) \triangleq \log \int_X \exp(\langle \eta, t(x) \rangle) \, dx \]

Claim \( \mathcal{A} \) is convex and \( \nabla \mathcal{A} : \Theta \to \mathcal{M} \subseteq W \) is

\[ \nabla \mathcal{A}(\eta) = \mathbb{E}[t(X)] \]
Def An exponential family of densities is
\[ p(x \mid \eta) = \exp(\langle \eta, t(x) \rangle - A(\eta)) \]
\[ A(\eta) \overset{\Delta}{=} \log \int_{\mathcal{X}} \exp(\langle \eta, t(x) \rangle) \, dx \]

Claim \( A \) is convex and \( \nabla A : \Theta \to \mathcal{M} \subseteq W \) is
\[ \nabla A(\eta) = \mathbb{E}[t(X)] \]

Proof
\[ \nabla A(\eta) = \frac{1}{\int_{\mathcal{X}} \exp(\langle \eta, t(x) \rangle) \, dx} \int_{\mathcal{X}} t(x) \exp(\langle \eta, t(x) \rangle) \, dx \]
\[ = \int_{\mathcal{X}} t(x)p(x \mid \eta) \, dx \]
An exponential family of densities is
\[ p(x; \eta) = \exp(\langle \eta, t(x) \rangle - A(\eta)) \]
\[ A(\eta) \triangleq \log \int_X \exp(\langle \eta, t(x) \rangle) \, dx \]

Claim \( A \) is convex and \( \nabla A : \Theta \to \mathcal{M} \subseteq \mathcal{W} \) is
\[ \nabla A(\eta) = \mathbb{E}[t(X)] \]
\[ \nabla^2 A(\eta) = \mathbb{E}[t(X)t(X)^T] \]
Def An exponential family of densities is

\[ p(x; \eta) = \exp(\langle \eta, t(x) \rangle - A(\eta)) \]

\[ A(\eta) \triangleq \log \int_X \exp(\langle \eta, t(x) \rangle) \, dx \]

Claim \( A \) is convex and \( \nabla A : \Theta \rightarrow \mathcal{M} \subseteq W \) is

\[ \nabla A(\eta) = \mathbb{E}[t(X)] \]

\[ \mathcal{M} \triangleq \{ \mu \in W : \exists p \cdot \mathbb{E}_p[t(X)] = \mu \} \]
Def An exponential family of densities is
\[ p(x; \eta) = \exp(\langle \eta, t(x) \rangle - A(\eta)) \]
\[ A(\eta) \triangleq \log \int_{\mathcal{X}} \exp(\langle \eta, t(x) \rangle) \, dx \]

Claim \( A \) is convex and \( \nabla A : \Theta \to \mathcal{M} \subseteq W \) is
\[ \nabla A(\eta) = \mathbb{E}[t(X)] \]

Claim The KL divergence between two members is
\[ KL(\eta_1, \eta_2) = \langle \eta_1 - \eta_2, \nabla A(\eta_1) \rangle - (A(\eta_1) - A(\eta_2)) \]
**Def** An *exponential family of densities* is

\[ p(x; \eta) = \exp(\langle \eta, t(x) \rangle - A(\eta)) \]

\[ A(\eta) \triangleq \log \int_X \exp(\langle \eta, t(x) \rangle) \, dx \]

**Claim** \( A \) is convex and \( \nabla A : \Theta \to M \subseteq W \) is

\[ \nabla A(\eta) = \mathbb{E}[t(X)] \]

**Claim** The KL divergence between two members is

\[ KL(\eta_1, \eta_2) = \langle \eta_1 - \eta_2, \nabla A(\eta_1) \rangle - (A(\eta_1) - A(\eta_2)) \]

**Def** Say family is *tractable* if \( A \) is easy to evaluate.
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots + \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots + \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]

\[ \psi_n(x_n) = \exp(\eta_n x_n), \quad \psi_{n,n+1}(x_n, x_{n+1}) = \exp(\eta_{n,n+1} x_n x_{n+1}) \]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots) \]

\[ \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]

\[ \psi_n(x_n) = \exp(\eta_n x_n), \quad \psi_{n,n+1}(x_n, x_{n+1}) = \exp(\eta_{n,n+1} x_n x_{n+1}) \]

\[ \phi_n(x_n) = \eta_n x_n, \quad \phi_{n,n+1}(x_n, x_{n+1}) = \eta_{n,n+1} x_n x_{n+1} \]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots \quad \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]

\[ \psi_n(x_n) = \exp(\eta_n x_n) \quad \psi_{n,n+1}(x_n, x_{n+1}) = \exp(\eta_{n,n+1} x_n x_{n+1}) \]

\[ \phi_n(x_n) = \eta_n x_n \quad \phi_{n,n+1}(x_n, x_{n+1}) = \eta_{n,n+1} x_n x_{n+1} \]

\[ t(x) = (\phi_1(x_1), \ldots, \phi_N(x_N), \phi_{1,2}(x_1, x_2), \ldots, \phi_{N-1,N}(x_{N-1}, x_N)) \]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]

\[ \psi_n(x_n) = \exp(\eta_n x_n), \quad \psi_{n,n+1}(x_n, x_{n+1}) = \exp(\eta_{n,n+1} x_n x_{n+1}) \]

\[ \phi_n(x_n) = \eta_n x_n, \quad \phi_{n,n+1}(x_n, x_{n+1}) = \eta_{n,n+1} x_n x_{n+1} \]

\[ t(x) = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \cdots & x_N \end{bmatrix} \]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots + \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]

\[ \psi_n(x_n) = \exp(\eta_n x_n), \quad \psi_{n,n+1}(x_n, x_{n+1}) = \exp(\eta_{n,n+1} x_n x_{n+1}) \]

\[ \phi_n(x_n) = \eta_n x_n, \quad \phi_{n,n+1}(x_n, x_{n+1}) = \eta_{n,n+1} x_n x_{n+1} \]

\[ t(x) = xx^\top \]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots + \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]

\[ \psi_n(x_n) = \exp(\eta_n x_n), \quad \psi_{n, n+1}(x_n, x_{n+1}) = \exp(\eta_{n, n+1} x_n x_{n+1}) \]

\[ \phi_n(x_n) = \eta_n x_n, \quad \phi_{n, n+1}(x_n, x_{n+1}) = \eta_{n, n+1} x_n x_{n+1} \]

\[ t(x) = xx^\top \]

\[ \eta = \begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{array} \]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots \]

\[ \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]

\[ \psi_n(x_n) = \exp(\eta_n x_n), \quad \psi_{n,n+1}(x_n, x_{n+1}) = \exp(\eta_{n,n+1} x_n x_{n+1}) \]

\[ \phi_n(x_n) = \eta_n x_n, \quad \phi_{n,n+1}(x_n, x_{n+1}) = \eta_{n,n+1} x_n x_{n+1} \]

\[ t(x) = xx^T \]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots + \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]

\[ \psi_n(x_n) = \exp(\eta_n x_n), \quad \psi_{n,n+1}(x_n, x_{n+1}) = \exp(\eta_{n,n+1} x_n x_{n+1}) \]

\[ \phi_n(x_n) = \eta_n x_n, \quad \phi_{n,n+1}(x_n, x_{n+1}) = \eta_{n,n+1} x_n x_{n+1} \]

\[ t(x) = x x^\top \]

\[ \eta = \begin{array}{cccccccc}
\end{array} \]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots + \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]

\[ t(x) = xx^T \]

\[ \eta = \begin{bmatrix} \ldots \end{bmatrix} \]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]

\[ t(x) = xx^\top \]

\[ \mathcal{A}(\eta) = \log \sum_{x \in \{0,1\}^N} \exp(\langle \eta, t(x) \rangle) \]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots + \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]

\[ t(x) = xx^T \]

\[ A(\eta) = \log \sum_{x \in \{0,1\}^N} \exp(\langle \eta, t(x) \rangle) \]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots + \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]

\[ t(x) = x x^\top \]

\[ A(\eta) = \log \sum_{x \in \{0,1\}^N} \exp(\langle \eta, t(x) \rangle) \]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots + \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]

\[ t(x) = xx^T \]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]

\[ t(x) = xx^\top \]

\[ A(\eta) = \log \sum_{x\in\{0,1\}^N} \exp(\langle \eta, t(x) \rangle) \]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]

\[ t(x) = xx^T \]

\[ \mathcal{A}(\eta) = \log \sum_{x \in \{0,1\}^N} \exp(\langle \eta, t(x) \rangle) \]
Example: binary hidden Markov model (HMM)

\[
x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\}
\]

\[
p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots \\
\quad \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots)
\]

\[
t(x) = xx^T
\]

\[
\mathcal{A}(\eta) = \log \sum_{x \in \{0, 1\}^N} \exp(\langle \eta, t(x) \rangle)
\]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]

\[ t(x) = xx^\top \]

\[ A(\eta) = \log \sum_{x \in \{0,1\}^N} \exp(\langle \eta, t(x) \rangle) \]

\[ \nabla A(\eta) = (E[x_1], \ldots, E[x_N], E[x_1 x_2], \ldots) \]
Example: binary hidden Markov model (HMM)

\[ x = (x_1, x_2, \ldots, x_N), \quad x_n \in \{0, 1\} \]

\[ p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots + \eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots) \]

\[ t(x) = xx^\top \]

\[ A(\eta) = \log \sum_{x \in \{0,1\}^N} \exp(\langle \eta, t(x) \rangle) \]

\[ \nabla A(\eta) = (\mathbb{E}[x_1], \ldots, \mathbb{E}[x_N], \mathbb{E}[x_1 x_2], \ldots) \]
Example: hidden Markov model (HMM)

```python
def log_normalizer(natparams, data):
    log_pi, log_A, log_B = natparams
    log_alpha = log_pi
    for y in data:
        log_alpha = logsumexp(log_alpha[:, None] + log_A, axis=0) + log_B[:, y]
    return logsumexp(log_alpha)

from autograd import grad
E_stats = grad(log_normalizer)(natparams, data)
```

https://github.com/HIPS/autograd/blob/master/examples/hmm_em.py
What about maximum likelihood?
What about maximum likelihood?

Say \( \{x_n\}_{n=1}^N \) are samples, consider

\[
\frac{1}{N} \log p(x ; \eta) = \langle \eta, \frac{1}{N} \sum_{n=1}^N t(x_n) \rangle - A(\eta)
\]

\[
= \langle \eta, \hat{\mu} \rangle - A(\eta)
\]
What about maximum likelihood?

Say \( \{x_n\}_{n=1}^N \) are samples, consider

\[
\frac{1}{N} \log p(x; \eta) = \langle \eta, \frac{1}{N} \sum_{n=1}^N t(x_n) \rangle - A(\eta)
\]

\[
= \langle \eta, \hat{\mu} \rangle - A(\eta)
\]

So maximum likelihood is the concave problem

\[
A^*(\mu) \triangleq \sup_{\eta \in \Theta} \langle \eta, \mu \rangle - A(\eta)
\]
What about maximum likelihood?

Say \( \{x_n\}_{n=1}^N \) are samples, consider

\[
\frac{1}{N} \log p(x; \eta) = \langle \eta, \frac{1}{N} \sum_{n=1}^N t(x_n) \rangle - A(\eta)
\]

\[
= \langle \eta, \hat{\mu} \rangle - A(\eta)
\]

So maximum likelihood is the concave problem

\[
A^*(\mu) \triangleq \sup_{\eta \in \Theta} \langle \eta, \mu \rangle - A(\eta)
\]

Gradient ascent is

\[
\eta_{k+1} = \eta_k + \alpha_k (\mu - \nabla A(\eta_k))
\]
What about maximum likelihood?

Say \( \{ x_n \}_{n=1}^N \) are samples, consider

\[
\frac{1}{N} \log p(x ; \eta) = \langle \eta, \frac{1}{N} \sum_{n=1}^N t(x_n) \rangle - A(\eta)
\]

\[
= \langle \eta, \hat{\mu} \rangle - A(\eta)
\]

So maximum likelihood is the concave problem

\[
A^*(\mu) \triangleq \sup_{\eta \in \Theta} \langle \eta, \mu \rangle - A(\eta)
\]

Gradient ascent is

\[
\eta_{k+1} = \eta_k + \alpha_k (\mu - \nabla A(\eta_k))
\]

Claim

\[
\nabla A^*(\mu) = \arg \sup_{\eta \in \Theta} \langle \eta, \mu \rangle - A(\eta)
\]
\[ A^*(\mu) = \sup_{\eta \in \Theta} \langle \eta, \mu \rangle - A(\eta) \quad \text{and} \quad A(\eta) = \sup_{\mu \in \mathcal{M}} \langle \eta, \mu \rangle - A^*(\mu) \]

\[ \eta(\mu) = \nabla A^*(\mu) \quad \text{and} \quad \mu(\eta) = \nabla A(\eta) \]
Summary for tractable exponential families

For each tractable exponential family…

1. implement \( t(x) \) and \( A(\eta) \) for exact inference and mean field variational inference

2. implement \( A^*(\mu) \) for maximum likelihood and (variational) expectation-maximization (EM)

3. implement \text{sample} for drawing samples and Gibbs sampling MCMC

Next up: composing tractable families into intractable ones!
\[ \pi = \begin{bmatrix} \pi^{(1)} & \pi^{(2)} & \pi^{(3)} \end{bmatrix} \]

\[ z_{t+1} \sim \pi(z_t) \]
\[\pi = \begin{bmatrix}
\pi^{(1)} \\
\pi^{(2)} \\
\pi^{(3)}
\end{bmatrix}\]

\[z_{t+1} \sim \pi^{(z_t)}\]

\[x_{t+1} = A^{(z_t)} x_t + B^{(z_t)} u_t \quad u_t \sim \mathcal{N}(0, I)\]
\[ y_t \mid x_t \sim \mathcal{N}(Cx_t, \Sigma), \quad \gamma = (C, \Sigma) \]
Compose exp. families, get compositional algorithms?
Compose exp. families, get compositional algorithms?

For \( x = (x_1, x_2) \) consider a negative energy function

\[
\log p(x ; \eta) = \langle \eta, t(x) \rangle + \text{const.}
\]

\[
= \langle \eta_{10}, t_1(x_1) \rangle + \langle \eta_{01}, t_2(x_2) \rangle
+ \langle \eta_{11}, t_1(x_1) \otimes t_2(x_2) \rangle + \text{const.}
\]
Compose exp. families, get compositional algorithms?

For $x = (x_1, x_2)$ consider a negative energy function

$$\log p(x ; \eta) = \langle \eta, t(x) \rangle + \text{const.}$$

$$= \langle \eta_{10}, t_1(x_1) \rangle + \langle \eta_{01}, t_2(x_2) \rangle$$

$$+ \langle \eta_{11}, t_1(x_1) \otimes t_2(x_2) \rangle + \text{const.}.$$ 

$$\log p(x_1 \mid x_2) = \langle \eta_{10} + \eta_{11} \cdot t_2(x_2), t_1(x_1) \rangle + \text{const.}.$$
Compose exp. families, get compositional algorithms?

For $x = (x_1, x_2)$ consider a negative energy function

$$\log p(x ; \eta) = \langle \eta, t(x) \rangle + \text{const.}$$

$$= \langle \eta_{10}, t_1(x_1) \rangle + \langle \eta_{01}, t_2(x_2) \rangle$$
$$+ \langle \eta_{11}, t_1(x_1) \otimes t_2(x_2) \rangle + \text{const.}$$
Compose exp. families, get compositional algorithms?

For $x = (x_1, \ldots, x_M)$ consider a negative energy function

$$
\log p(x ; \eta) = \langle \eta, t(x) \rangle + \text{const.}
$$

$$
= \sum_{\beta \in \beta} \langle \eta_{\beta}, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle
$$

$$
\triangleq g(t_1(x_1), \ldots, t_M(x_M))
$$
Compose exp. families, get compositional algorithms?

For \( x = (x_1, \ldots, x_M) \) consider a negative energy function

\[
\log p(x; \eta) = \langle \eta, t(x) \rangle + \text{const.}
\]

\[
= \sum_{\beta \in \beta} \langle \eta_{\beta}, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle
\]

\[
\triangleq g(t_1(x_1), \ldots, t_M(x_M))
\]

where \( g \) is a multi-affine polynomial
Compose exp. families, get compositional algorithms?

For $x = (x_1, \ldots, x_M)$ consider a negative energy function

$$\log p(x ; \eta) = \langle \eta, t(x) \rangle + \text{const.}$$

$$= \sum_{\beta \in \beta} \langle \eta_\beta, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle$$

$$\triangleq g(t_1(x_1), \ldots, t_M(x_M))$$

where $g$ is a multi-affine polynomial

$\beta \in \beta \subseteq \{0, 1\}^M$ indexes monomial terms
Compose exp. families, get compositional algorithms?

For \( x = (x_1, \ldots, x_M) \) consider a negative energy function

\[
\log p(x ; \eta) = \langle \eta, t(x) \rangle + \text{const.}
\]

\[
= \sum_{\beta \in \beta} \langle \eta_\beta, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle
\]

\[
\triangleq g(t_1(x_1), \ldots, t_M(x_M))
\]

where \( g \) is a multi-affine polynomial

\( \beta \in \beta \subseteq \{0, 1\}^M \) indexes monomial terms

each \( t_m(x_m) \) corresponds to a tractable exponential family
For $x = (x_1, \ldots, x_M)$ consider a negative energy function

$$\log p(x ; \eta) = \langle \eta, t(x) \rangle + \text{const.}$$

$$= \sum_{\beta \in \beta} \langle \eta_\beta, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle$$

$$\triangleq g(t_1(x_1), \ldots, t_M(x_M))$$

where $g$ is a multi-affine polynomial

$\beta \in \beta \subseteq \{0, 1\}^M$ indexes monomial terms

each $t_m(x_m)$ corresponds to a tractable exponential family

But the normalizer $A$ is not tractable!
\[
\log p(x ; \eta) = \langle \eta, t(x) \rangle + \text{const.}
\]
\[
= \sum_{\beta \in \Phi} \langle \eta_\beta, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \ldots \otimes t_M(x_M)^{\beta_M} \rangle
\]
\[
\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}
\]
\[
\log p(x ; \eta) = \langle \eta, t(x) \rangle + \text{const.}
\]
\[
= \sum_{\beta \in \mathcal{B}} \langle \eta_{\beta}, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle
\]
\[
\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}
\]

Can we build algorithms for

1. approximate sampling \( x \sim p(x ; \eta) \) via MCMC?
2. approximate expectations \( \mathbb{E}[t(X)] \) and \( A \)?
3. variational Expectation-Maximization to estimate \( \eta \) that exploit the tractable parts?
\[ \log p(x; \eta) = \langle \eta, t(x) \rangle + \text{const.} \]
\[ = \sum_{\beta \in \beta} \langle \eta_{\beta}, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle \]
\[ \triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.} \]
$$\log p(x ; \eta) = \langle \eta, t(x) \rangle + \text{const.}$$

$$= \sum_{\beta \in \Phi} \langle \eta_\beta, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle$$

$$\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}$$

**Claim** [Gibbs sampling]

$$p(x_m | x_{-m}) = \exp(\langle \eta^*_m, t_m(x_m) \rangle - A_m(\eta^*_m))$$

$$\eta^*_m \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))$$
\[
\log p(x; \eta) = \langle \eta, t(x) \rangle + \text{const.}
\]
\[
= \sum_{\beta \in \beta} \langle \eta_{\beta}, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle
\]
\[
\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}
\]

Claim [Gibbs sampling]
\[
p(x_m | x_{-m}) = \exp(\langle \eta_m^*, t_m(x_m) \rangle - A_m(\eta_m^*))
\]
\[
\eta_m^* \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))
\]

```python
from autograd import grad

def gibbs(g, samplers, niter, x):
    for _ in range(niter):
        for m in range(M):
            x[m] = samplers[m](grad(g)(*x))
    return x
```
\[
\log p(x ; \eta) = \langle \eta, t(x) \rangle + \text{const.}
\]
\[
= \sum_{\beta \in \beta} \langle \eta_\beta, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle
\]
\[
\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}
\]

**Claim** [Gibbs sampling]

\[
p(x_m | x_{-m}) = \exp(\langle \eta_m^*, t_m(x_m) \rangle - A_m(\eta_m^*))
\]
\[
\eta_m^* \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))
\]

```python
from autograd import grad

def gibbs(g, samplers, niter, x):
    for _ in range(niter):
        for m in range(M):
            x[m] = samplers[m](grad(g)(*x))
    return x
```
\[
\log p(x; \eta) = \langle \eta, t(x) \rangle + \text{const.}
\]
\[
= \sum_{\beta \in \mathcal{B}} \langle \eta_{\beta}, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle
\]
\[
\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}
\]

Claim [Gibbs sampling]

\[
p(x_m | x_{-m}) = \exp(\langle \eta_m^*, t_m(x_m) \rangle - A_m(\eta_m^*))
\]
\[
\eta_m^* \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))
\]

```python
from autograd import grad
def gibbs(g, samplers, niter, x):
    for _ in range(niter):
        for m in range(M):
            x[m] = samplers[m](grad(g)(*x))
    return x
```
\[
\log p(x; \eta) = \langle \eta, t(x) \rangle + \text{const.}
\]
\[
= \sum_{\beta \in \beta} \langle \eta_\beta, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \ldots \otimes t_M(x_M)^{\beta_M} \rangle
\]
\[
\Delta g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}
\]

**Claim** [Gibbs sampling]

\[
p(x_m | x_{-m}) = \exp(\langle \eta^*_m, t_m(x_m) \rangle - A_m(\eta^*_m))
\]

\[
\eta^*_m \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))
\]

```python
from autograd import grad

def gibbs(g, samplers, niter, x):
    for _ in range(niter):
        for m in range(M):
            x[m] = samplers[m](grad(g)(*x))
    return x
```
\[
\log p(x ; \eta) = \langle \eta, t(x) \rangle + \text{const.}
\]
\[
= \sum_{\beta \in \beta} \langle \eta_\beta, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle
\]
\[
\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}
\]

**Claim** [Gibbs sampling]

\[
p(x_m | x_{-m}) = \exp(\langle \eta^*_m, t_m(x_m) \rangle - A_m(\eta^*_m))
\]
\[
\eta^*_m \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))
\]

```python
from autograd import grad
def gibbs(g, samplers, niter, x):
    for _ in range(niter):
        for m in range(M):
            x[m] = samplers[m](grad(g)(*x))
    return x
```
\[
\log p(x; \eta) = \left< \eta, t(x) \right> + \text{const.} \\
= \sum_{\beta \in \beta} \left< \eta_\beta, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \ldots \otimes t_M(x_M)^{\beta_M} \right> \\
\Delta g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}
\]

**Claim** [Gibbs sampling]

\[
p(x_m | x_{-m}) = \exp(\left< \eta_m^*, t_m(x_m) \right> - A_m(\eta_m^*))
\]

\[
\eta_m^* \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))
\]

```python
from autograd import grad

def gibbs(g, samplers, niter, x):
    for _ in range(niter):
        for m in range(M):
            x[m] = samplers[m](grad(g)(*x))
    return x
```
\[
\log p(x \mid \eta) = \langle \eta, t(x) \rangle + \text{const.}
\]
\[
= \sum_{\beta \in \mathcal{B}} \langle \eta_{\beta}, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle
\]
\[
\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}
\]

**Claim** [Gibbs sampling]

\[
p(x_m \mid x_{-m}) = \exp(\langle \eta_m^*, t_m(x_m) \rangle - A_m(\eta_m^*))
\]
\[
\eta_m^* \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))
\]

```python
from autograd import grad

def gibbs(g, samplers, niter, x):
    for _ in range(niter):
        for m in range(M):
            x[m] = samplers[m](grad(g)(*x))
    return x
```
\[ \log p(x; \eta) = \langle \eta, t(x) \rangle + \text{const.} \]

\[ = \sum_{\beta \in \beta} \langle \eta_\beta, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle \]

\[ \triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.} \]

**Claim** [Gibbs sampling]

\[ p(x_m | x_{-m}) = \exp(\langle \eta_m^*, t_m(x_m) \rangle - A_m(\eta_m^*)) \]

\[ \eta_m^* \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M)) \]

```python
from autograd import grad

def gibbs(g, samplers, niter, x):
    for _ in range(niter):
        for m in range(M):
            x[m] = samplers[m](grad(g)(*x))
    return x
```
\[
\log p(x; \eta) = \langle \eta, t(x) \rangle + \text{const.}
= \sum_{\beta \in \beta} \langle \eta_\beta, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle
\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}
\]

**Claim** [Gibbs sampling]
\[
p(x_m | x_{-m}) = \exp(\langle \eta^*_m, t_m(x_m) \rangle - A_m(\eta^*_m))
\]
\[
\eta^*_m \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))
\]

```python
from autograd import grad
def gibbs(g, samplers, niter, x):
    for _ in range(niter):
        for m in range(M):
            x[m] = samplers[m](grad(g)(*x))
    return x
```
\[
\log p(x; \eta) = \langle \eta, t(x) \rangle + \text{const.}
\]
\[
= \sum_{\beta \in \beta} \langle \eta_{\beta}, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle
\]
\[
\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}
\]

**Claim** [Gibbs sampling]

\[
p(x_m | x_{-m}) = \exp(\langle \eta^*_m, t_m(x_m) \rangle - A_m(\eta^*_m))
\]
\[
\eta^*_m \triangleq \nabla m g(t_1(x_1), \ldots, t_M(x_M))
\]

```python
from autograd import grad

def gibbs(g, samplers, niter, x):
    for _ in range(niter):
        for m in range(M):
            x[m] = samplers[m](grad(g)(*x))
    return x
```
Consider the variational family

\[ q(x) = \prod_{m=1}^{M} q_m(x_m ; \eta_m) \]

\[ q(x_m ; \eta_m) = \exp(\langle \eta_m, t_m(x_m) \rangle - A_m(\eta_m)) \]
Consider the variational family

\[ q(x) = \prod_{m=1}^{M} q_m(x_m ; \eta_m) \]

\[ q(x_m ; \eta_m) = \exp(\langle \eta_m, t_m(x_m) \rangle - A_m(\eta_m)) \]

That’s a sub-family of \( p(x ; \eta) \) with linear constraints on \( \eta \).
Consider the variational family

\[ q(x) = \prod_{m=1}^{M} q_m(x_m ; \eta_m) \]

\[ q(x_m ; \eta_m) = \exp(\langle \eta_m, t_m(x_m) \rangle - \mathcal{A}_m(\eta_m)) \]

That’s a sub-family of \( p(x ; \eta) \) with linear constraints on \( \eta \).

**Claim** [Variational lower bound]

\[ \mathcal{A}(\eta) \geq \langle \eta, \mathbb{E}_q[t(X)] \rangle - \sum_m \mathcal{A}_m^*(\mu_m) \]

\[ = g(\mu_1, \ldots, \mu_M) - \sum_m \mathcal{A}_m^*(\mu_m) \triangleq \mathcal{L}(\eta) \]
Consider the variational family

\[
q(x) = \prod_{m=1}^{M} q_m(x_m; \eta_m)
\]

\[
q(x_m; \eta_m) = \exp(\langle \eta_m, t_m(x_m) \rangle - A_m(\eta_m))
\]

That’s a sub-family of \( p(x; \eta) \) with linear constraints on \( \eta \).

**Claim** [Variational lower bound]

\[
A(\eta) \geq \langle \eta, \mathbb{E}_q[t(X)] \rangle - \sum_m A^*_m(\mu_m)
\]

\[
= g(\mu_1, \ldots, \mu_M) - \sum_m A^*_m(\mu_m) \triangleq \mathcal{L}(\eta)
\]

**Proof** Use the variational definition of \( A \):

\[
A(\eta) = \sup_{\mu \in \mathcal{M}} \langle \eta, \mu \rangle - A^*(\mu)
\]
Consider the variational family

\[ q(x) = \prod_{m=1}^{M} q_m(x_m ; \eta_m) \]

\[ q(x_m ; \eta_m) = \exp(\langle \eta_m, t_m(x_m) \rangle - A_m(\eta_m)) \]

That’s a sub-family of \( p(x ; \eta) \) with linear constraints on \( \eta \).

\[ g(\nabla A_1(\eta_1), \ldots, \nabla A_M(\eta_M)) - (\sum_m \langle \eta_m, \nabla A_m(\eta_m) \rangle - A_m(\eta_m)) \]
Consider the variational family

\[ q(x) = \prod_{m=1}^{M} q_m(x_m ; \eta_m) \]

\[ q(x_m ; \eta_m) = \exp(\langle \eta_m, t_m(x_m) \rangle - A_m(\eta_m)) \]

That’s a sub-family of \( p(x ; \eta) \) with linear constraints on \( \eta \).

**Claim** [Variational lower bound]

\[ \mathcal{A}(\eta) \geq \langle \eta, \mathbb{E}_q[t(X)] \rangle - \sum_m A_m^*(\mu_m) \]

\[ = g(\mu_1, \ldots, \mu_M) - \sum_m A_m^*(\mu_m) \overset{\Delta}{=} \mathcal{L}(\eta) \]

\[ g(\nabla \mathcal{A}_1(\eta_1), \ldots, \nabla \mathcal{A}_M(\eta_M)) - (\sum_m \langle \eta_m, \nabla A_m(\eta_m) \rangle - A_m(\eta_m)) \]
Consider the variational family

\[
q(x) = \prod_{m=1}^{M} q_m(x_m; \eta_m)
\]

\[
q(x_m; \eta_m) = \exp(\langle \eta_m, t_m(x_m) \rangle - A_m(\eta_m))
\]

**Claim** [Structured mean field]

\[
\arg \max_{\eta_m} \mathcal{L}(\eta_1, \ldots, \eta_M) = \nabla_m g(\mu_1, \ldots, \mu_M)
\]

where \( \mu_{m'} = \nabla A_{m'}(\eta_{m'}) \) for \( m' = 1, \ldots, M \)
Consider the variational family

\[
q(x) = \prod_{m=1}^{M} q_m(x_m ; \eta_m)
\]

\[
q(x_m ; \eta_m) = \exp(\langle \eta_m , t_m(x_m) \rangle - A_m(\eta_m))
\]

**Claim** [Structured mean field]

\[
\arg \max_{\eta_m} \mathcal{L}(\eta_1, \ldots, \eta_M) = \nabla_m g(\mu_1, \ldots, \mu_M)
\]

where \( \mu_{m'} = \nabla A_{m'}(\eta_{m'}) \) for \( m' = 1, \ldots, M \)

```python
def meanfield(g, As, etas):
    def meanfield_sweep(mus):
        for m in range(M):
            mus[m] = grad(As[m])(grad(g, m)(*mus))
        return mus

    mus = [grad(As[m])(etas[m]) for m in range(M)]
    mu_stars = fixed_point(meanfield_sweep, mus)
    return [grad(g, m)(*mu_stars) for m in range(M)]
```
def neg_energy(eta_prior, t_y, t_theta, t_z, t_x):
    t_z_node = markovchain.pair_to_node(t_z)
    t_z_trans = t_z[..., :-1, :, :]
    t_x_init = lds.pair_to_node(t_x[..., 0, :, :])
    t_x_trans = t_x[..., :-1, :, :]
    t_xy = gaussian.stats_product(lds.pair_to_node(t_x), t_y)
    return dot(eta_prior, t_theta) - logZ_theta(eta_prior)
    + np.einsum('i,...i->', t_theta[0], t_z_node[..., 0, :])
    + np.einsum('ij,...tij->', t_theta[1], t_z_trans)
    + np.einsum('kij,k,...ij->', t_theta[2], t_z_node[..., 0, :], t_x_init)
    + np.einsum('kij,tk,...tij->', t_theta[3], t_z_node, t_x)
    + np.einsum('ij,...tij->', t_theta[4], t_xy)
def normal_logpdf(x, loc, scale):
    prec = 1. / scale**2
    return -(np.sum(prec * mu**2) - np.sum(np.log(prec))
    + np.log(2. * np.pi)) * N / 2.

def normal_logpdf(pi, z, mu, tau, x):
    logp = (np.sum((alpha-1) * np.log(x)) - np.sum(gammaln(alpha))
    + np.sum(gammaln(np.sum(alpha, -1))))
    logp += normal_logpdf(mu, 0., 1./np.sqrt(kappa * tau))
    logp += np.sum(one_hot(z, K) * np.log(pi))
    logp += (a-1)*np.log(tau) - b*tau + a*np.log(b) - gammaln(a)
    mu_z = np.dot(one_hot(z, K), mu)
    loglike = normal_logpdf(x, mu_z, 1./np.sqrt(tau))
    return logp + loglike
Domain-specific term graph rewriting implementation

- **Tracer** using Autograd’s API to map Python to term graphs
- **Pattern matcher** to do pattern-directed invocation
  - Python-embedded pattern language
  - Compiled into **continuation-passing matcher combinators** (~300 loc)

```python
pat = (Einsum, Str('formula'), Segment('args1'),
       (Choice(Subtract('op'), Add('op')), Val('x'), Val('y')), Segment('args2'))
```

- **Rewriters** are syntactic graph macros using tracing to get **quasi-quasiquotes**

```python
def rewriter(formula, op, x, y, args1, args2):
    return op(np.einsum(formula, *(args1 + (x,) + args2)),
              np.einsum(formula, *(args1 + (y,) + args2)))

distribute_einsum = Rule(pat, rewriter)  # Rule is a namedtuple
Goals

1. **Motivate** why PGMs + DNNs are a *revolution* waiting to happen

2. **Survey the fundamentals** of PGMs and exponential families so that you have a *broad view of the territory*

3. Show how to **unify many models and algorithms** in a framework that lets you **leverage automatic differentiation**

4. Make **SVAEs** and related PGM + DNN architectures **super obvious** so that you can **invent better ones**
Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen

2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory

3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation

4. Make SVAEs and related PGM + DNN architectures super obvious so that you can invent better ones
$y_t \mid x_t, \gamma \sim \mathcal{N}(\mu(x_t; \gamma), \Sigma(x_t; \gamma))$
\( y_t \mid x_t, \gamma \sim \mathcal{N}(\mu(x_t; \gamma), \Sigma(x_t; \gamma)) \)
\( \theta \)

\[
\begin{align*}
\gamma & \rightarrow z_1 \rightarrow z_2 \rightarrow z_3 \rightarrow z_4 \\
& \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \\
& \rightarrow y_1 \rightarrow y_2 \rightarrow y_3 \rightarrow y_4
\end{align*}
\]

\[
\begin{align*}
\theta & \rightarrow z_n \\
& \rightarrow x_n \\
& \rightarrow y_n
\end{align*}
\]
$p(\theta) \quad p(x | \theta)$

conjugate prior on global variables

exponential family on local variables
conjugate prior on global variables
exponential family on local variables
neural network observation model
Inference?
\[ q^*(x) \triangleq \arg \max_{q(x)} \mathcal{L}[q(\theta)q(x)] \]

Natural gradient SVI for nice exp. fam. PGMs \[1,2\]

\[ p(x \mid \theta) \text{ is a linear dynamical system} \]
\[ p(y \mid x, \theta) \text{ is a linear-Gaussian observation} \]
\[ p(\theta) \text{ is a conjugate prior} \]
\( p(x \mid \theta) \) is a linear dynamical system
\( p(y \mid x, \theta) \) is a linear-Gaussian observation
\( p(\theta) \) is a conjugate prior

\[ q(\theta)q(x) \approx p(\theta, x \mid y) \]
$p(x | \theta)$ is a linear dynamical system
$p(y | x, \theta)$ is a linear-Gaussian observation
$p(\theta)$ is a conjugate prior

\[
\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)} \left[ \log \frac{p(\theta, x, y)}{q(\theta)q(x)} \right]
\]
\( p(x \mid \theta) \) is a linear dynamical system
\( p(y \mid x, \theta) \) is a linear-Gaussian observation
\( p(\theta) \) is a conjugate prior

\[
\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)} \left[ \log \frac{p(\theta, x, y)}{q(\theta)q(x)} \right]
\]

\( \eta^*_x(\eta_\theta) \triangleq \arg \max_{\eta_x} \mathcal{L}(\eta_\theta, \eta_x) \)

\( \mathcal{L}_{SVI}(\eta_\theta) \triangleq \mathcal{L}(\eta_\theta, \eta^*_x(\eta_\theta)) \)
Proposition (natural gradient SVI of Hoffman et al. 2013)

\[ \nabla \mathcal{L}_{\text{SVI}}(\eta_\theta) = \eta_\theta^0 + \mathbb{E}_{q(x)}(t_{xy}(x, y), 1) - \eta_\theta \]
Proposition (natural gradient SVI of Hoffman et al. 2013)

\[ \nabla L_{\text{SVI}}(\eta_\theta) = \nabla_0 L_{\text{SVI}}(\eta_\theta) + \sum_{n=1}^{N} \mathbb{E}_{q(x_n)}(t_{xy}(x_n, y_n), 1) - \eta_\theta \]
Step 1: compute evidence potentials

Step 1: compute evidence potentials

Step 1: compute evidence potentials

Step 1: compute evidence potentials

Step 1: compute evidence potentials

Step 2: run fast message passing

Step 1: compute evidence potentials

Step 2: run fast message passing

Step 1: compute evidence potentials

Step 2: run fast message passing

Step 3: compute natural gradient

arbitrary inference queries
$p(x \mid \theta)$ is a linear dynamical system

$p(y \mid x, \gamma)$ is a neural network decoder

$p(\theta)$ is a conjugate prior
$p(x \mid \theta)$ is a linear dynamical system

$p(y \mid x, \gamma)$ is a neural network decoder

$p(\theta)$ is a conjugate prior

$q(\theta)q(x) \approx p(\theta, x \mid y, \gamma)$
\( p(x \mid \theta) \) is a linear dynamical system
\( p(y \mid x, \gamma) \) is a neural network decoder
\( p(\theta) \) is a conjugate prior

\[
\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)} \left[ \log \frac{p(\theta, x)p(y \mid x, \gamma)}{q(\theta)q(x)} \right]
\]

\[ q(\theta)q(x) \approx p(\theta, x \mid y, \gamma) \]
\( p(x \mid \theta) \) is a linear dynamical system
\( p(y \mid x, \gamma) \) is a neural network decoder
\( p(\theta) \) is a conjugate prior

\[
\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)} \left[ \log \frac{p(\theta, x)p(y \mid x, \gamma)}{q(\theta)q(x)} \right]
\]

\[
\eta_x^*(\eta_\theta) \triangleq \arg \max_{\eta_x} \mathcal{L}(\eta_\theta, \eta_x)
\]
\( p(x \mid \theta) \) is a linear dynamical system
\( p(y \mid x, \gamma) \) is a neural network decoder
\( p(\theta) \) is a conjugate prior,

\[
\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)} \left[ \log \frac{p(\theta, x)p(y \mid x, \gamma)}{q(\theta)q(x)} \right]
\]

\( \eta_x^*(\eta_\theta) \triangleq \arg \max_{\eta_x} \mathcal{L}(\eta_\theta, \eta_x) \)

\[
\mathcal{L}_{\text{SVI}}(\eta_\theta) \triangleq \mathcal{L}(\eta_\theta, \eta_x^*(\eta_\theta))
\]
Variational autoencoders and amortized inference

\[ q^*(x) \triangleq \mathcal{N}(x \mid \mu(y; \phi), \Sigma(y; \phi)) \]

\[ q^*(x_n) \triangleq \mathcal{N}(x_n \mid \mu(y_n; \phi), \Sigma(y_n; \phi)) \]
\[ q^*(x_n) \triangleq \mathcal{N}(x_n \mid \mu(y_n; \phi), \Sigma(y_n; \phi)) \]
\[ q^*(x_n) \triangleq \mathcal{N}(x_n | \mu(y_n; \phi), \Sigma(y_n; \phi)) \]

\[ \mathcal{L}_{VAE}(\eta_\gamma, \phi) \triangleq \mathcal{L}(\eta_\gamma, \eta^*_x(\phi)) \]
\[ \mu_t(y_t; \phi_\mu) \]
\[ J_{t,t}(y_t; \phi_D) \]
\[ J_{t,t+1}(y_t, y_{t+1}; \phi_B) \]

\[ \mu_t(y_t; \phi_\mu) \]
\[ J_{t,t}(y_t; \phi_D) \]
\[ J_{t,t+1}(y_t, y_{t+1}; \phi_B) \]

$$q^*(x) \triangleq \arg \max_{q(x)} \mathcal{L}[q(\theta)q(x)]$$

Natural gradient SVI
\[ q^*(x) \triangleq \arg \max_{q(x)} \mathcal{L}[q(\theta)q(x)] \]

Natural gradient SVI

- expensive for general obs.
\[ q^*(x) \triangleq \arg\max_{q(x)} \mathcal{L}[q(\theta)q(x)] \]

Natural gradient SVI

- expensive for general obs.

+ optimal local factor
Natural gradient SVI

- expensive for general obs.
+ optimal local factor
+ exploits conj. graph structure

\[ q^*(x) \triangleq \arg \max_{q(x)} \mathcal{L}[q(\theta)q(x)] \]
\[ q^*(x) \triangleq \arg \max_{q(x)} \mathcal{L}[q(\theta)q(x)] \]

Natural gradient SVI

- expensive for general obs.
+ optimal local factor
+ exploits conj. graph structure
+ arbitrary inference queries
\[ q^*(x) \triangleq \arg \max_{q(x)} \mathcal{L}[q(\theta)q(x)] \]

Natural gradient SVI

- expensive for general obs.
+ optimal local factor
+ exploits conj. graph structure
+ arbitrary inference queries
+ natural gradients
Natural gradient SVI

$$q^*(x) \triangleright \arg\max_{q(x)} \mathcal{L}[q(\theta) q(x)]$$

Variational autoencoders

$$q^*(x) \triangleright \mathcal{N}(x | \mu(y; \phi), \Sigma(y; \phi))$$

- expensive for general obs.
+ optimal local factor
+ exploits conj. graph structure
+ arbitrary inference queries
+ natural gradients
\[ p \xrightarrow{\text{exploits conj. graph structure}} q \]

Natural gradient SVI

- expensive for general obs.
- optimal local factor
- exploits conj. graph structure
- arbitrary inference queries
- natural gradients

Variational autoencoders

+ fast for general obs.
- suboptimal local inference
- \( \phi \) does all local inference
- limited inference queries
- no cheap natural gradients

\[ q^*(x) \triangleq \arg \max_{q(x)} \mathcal{L}[q(\theta)q(x)] \]

\[ q^*(x) \triangleq \mathcal{N}(x \mid \mu(y; \phi), \Sigma(y; \phi)) \]
\[ q^*(x) \triangleq \arg \max_{q(x)} \mathcal{L}[q(\theta)q(x)] \]

\[ q^*(x) \triangleq \mathcal{N}(x | \mu(y; \phi), \Sigma(y; \phi)) \]

\[ q^*(x) \triangleq ? \]

Natural gradient SVI

Variational autoencoders

Structured VAEs [1]

- expensive for general obs.
  + fast for general obs.
- suboptimal local inference
- \( \phi \) does all local inference
- limited inference queries
- no cheap natural gradients

- optimal local factor
- exploits conj. graph structure
- arbitrary inference queries
- natural gradients

Natural gradient SVI

- expensive for general obs.
+ optimal local factor
+ exploits conj. graph structure
+ arbitrary inference queries
+ natural gradients

Variational autoencoders

+ fast for general obs.
- suboptimal local inference
- $\phi$ does all local inference
- limited inference queries
- no cheap natural gradients

Structured VAEs [1]

+ fast for general obs.
± optimal given conj. evidence
+ exploits conj. graph structure
+ arbitrary inference queries
+ natural gradients on $\eta_\theta$

---

$\begin{align*}
q^*(x) &\triangleq \arg\max_{q(x)} \mathcal{L}[q(\theta)q(x)] \\
q^*(x) &\triangleq \mathcal{N}(x | \mu(y; \phi), \Sigma(y; \phi)) \\
q^*(x) &\triangleq ?
\end{align*}$

Inference: recognition networks output conjugate potentials, then apply fast graphical model inference
\[
\mathcal{L}(\eta_{\theta}, \eta_{x}) \triangleq \mathbb{E}_{q(\theta)q(x)} \left[ \log \frac{p(\theta, x)p(y | x, \gamma)}{q(\theta)q(x)} \right]
\]
\[
\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)} \left[ \log \frac{p(\theta, x)p(y \mid x, \gamma)}{q(\theta)q(x)} \right]
\]

\[\mathbb{E}_{q(\gamma)} \log p(y_t \mid x_t, \gamma)\]
where $\psi(x; y, \phi)$ is a conjugate potential for $p(x | \theta)$. 

\[
\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)} \left[ \log \frac{p(\theta, x)p(y | x, \gamma)}{q(\theta)q(x)} \right]
\]

\[
\hat{\mathcal{L}}(\eta_\theta, \eta_x, \phi) \triangleq \mathbb{E}_{q(\theta)q(x)} \left[ \log \frac{p(\theta, x)\exp(\psi(x; y, \phi))}{q(\theta)q(x)} \right]
\]
\[
\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)} \left[ \log \frac{p(\theta, x)p(y \mid x, \gamma)}{q(\theta)q(x)} \right]
\]

\[
\hat{\mathcal{L}}(\eta_\theta, \eta_x, \phi) \triangleq \mathbb{E}_{q(\theta)q(x)} \left[ \log \frac{p(\theta, x)\exp(\psi(x ; y, \phi))}{q(\theta)q(x)} \right]
\]

where \( \psi(x ; y, \phi) \) is a conjugate potential for \( p(x \mid \theta) \).
\[ \mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)} \left[ \log \frac{p(\theta, x)p(y \mid x, \gamma)}{q(\theta)q(x)} \right] \]

\[ \hat{\mathcal{L}}(\eta_\theta, \eta_x, \phi) \triangleq \mathbb{E}_{q(\theta)q(x)} \left[ \log \frac{p(\theta, x) \exp(\psi(x ; y, \phi))}{q(\theta)q(x)} \right] \]

where \( \psi(x ; y, \phi) \) is a conjugate potential for \( p(x \mid \theta) \).

\[ \eta_x^*(\eta_\theta, \phi) \triangleq \arg \max_{\eta_x} \hat{\mathcal{L}}(\eta_\theta, \eta_x, \phi) \]

\[ \mathcal{L}_{\text{SVAE}}(\eta_\theta, \phi) \triangleq \mathcal{L}(\eta_\theta, \eta_x^*(\eta_\theta, \phi)) \]
Step 1: apply recognition network
Step 1: apply recognition network
Step 1: apply recognition network
Step 1: apply recognition network
Step 1: apply recognition network

Step 2: run fast PGM algorithms
Step 1: apply recognition network

Step 2: run fast PGM algorithms
Step 1: apply recognition network

Step 2: run fast PGM algorithms

Step 3: sample, compute flat grads
Step 1: apply recognition network

Step 2: run fast PGM algorithms

Step 3: sample, compute flat grads
Step 1: apply recognition network

Step 2: run fast PGM algorithms

Step 3: sample, compute flat grads

Step 4: compute natural gradient
Step 1: apply recognition network

Step 2: run fast PGM algorithms

Step 3: sample, compute flat grads

Step 4: compute natural gradient
data space

latent space
arbitrary inference queries*

*see next slide
SVAEs can use any inference network architectures

SVAEs
Application: learn syllable representation of behavior from video.
start rear
fall from rear
fall from rear
grooming
Discovery of Heterozygous Phenotypes in Ror1b Mice

... and high and low doses of each drug

from Alex Wiltschko preprint
Goals
Goals

1. **Motivate** why PGMs + DNNs are a **revolution** waiting to happen
Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen
2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory
Goals

1. **Motivate** why PGMs + DNNs are a revolution waiting to happen

2. **Survey the fundamentals** of PGMs and exponential families so that you have a broad view of the territory

3. Show how to **unify many models and algorithms** in a framework that lets you **leverage automatic differentiation**
Goals

1. **Motivate** why PGMs + DNNs are a revolution waiting to happen

2. **Survey the fundamentals** of PGMs and exponential families so that you have a broad view of the territory

3. Show how to **unify many models and algorithms** in a framework that lets you **leverage automatic differentiation**

4. Make **SVAEs** and related PGM + DNN architectures super obvious so that you can **invent better ones**
Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen
2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory
3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation
4. Make SVAEs and related PGM + DNN architectures super obvious so that you can invent better ones

Non-goals

1. Cover the recent literature on PGMs + DNNs
2. Unpack all the technical details
import jax.numpy as np
from jax import jit, grad, vmap

def predict(params, inputs):
    for W, b in params:
        outputs = np.dot(inputs, W) + b
        inputs = np.tanh(outputs)
    return outputs

def loss(params, batch):
    inputs, targets = batch
    preds = predict(params, inputs)
    return np.sum((preds - targets) ** 2)

gradients_fun = jit(grad(loss))
perexample_grads = jit(vmap(grad(loss), (None, 0)))
What is JAX?

```python
import jax.numpy as np
from jax import jit, grad, vmap

def predict(params, inputs):
    for W, b in params:
        outputs = np.dot(inputs, W) + b
        inputs = np.tanh(outputs)
    return outputs

def loss(params, batch):
    inputs, targets = batch
    preds = predict(params, inputs)
    return np.sum((preds - targets) ** 2)

gradient_fun = jit(grad(loss))
perexample_grads = jit(vmap(grad(loss), (None, 0)))
```
What is JAX?

import jax.numpy as np
from jax import jit, grad, vmap

def predict(params, inputs):
    for W, b in params:
        outputs = np.dot(inputs, W) + b
        inputs = np.tanh(outputs)
    return outputs

def loss(params, batch):
    inputs, targets = batch
    preds = predict(params, inputs)
    return np.sum((preds - targets) ** 2)

gradient_fun = jit(grad(loss))
perexample_grads = jit(vmap(grad(loss), (None, 0)))

JAX is an extensible system for composable function transformations of Python + NumPy code.
Composing graphical models with neural networks like chocolate and peanut butter

https://youtu.be/O7oD_oX-Gio

or

Graphical models and exponential families in the age of differentiable programming

David Duvenaud  Alex Wilitchko  Matthew D. Hoffman  Dustin Tran  Scott Linderman  Sandeep Robert Datta  Ryan P. Adams

Matthew J Johnson (mattjj@google.com)
July 22 2019 @ UAI 2019