An introduction to the structured variational autoencoders (SVAE)

Sharad Vikram

July 31, 2017

UCSD

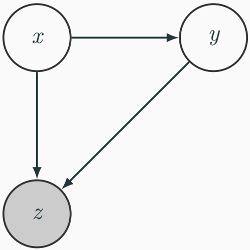
Variational inference

Bayesian inference

In Bayesian inference, we compute the posterior distribution of observations given data.

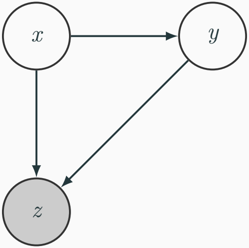
Bayesian inference

In Bayesian inference, we compute the posterior distribution of observations given data.



Bayesian inference

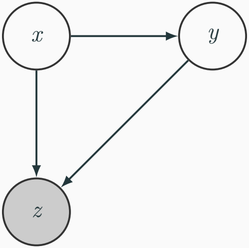
In Bayesian inference, we compute the posterior distribution of observations given data.



We are interested in the posterior distribution $p(x, y|z)$

Bayesian inference

In Bayesian inference, we compute the posterior distribution of observations given data.



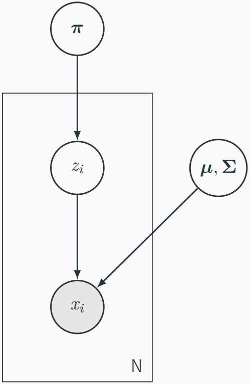
We are interested in the posterior distribution $p(x, y|z)$

This can be computed via Bayes rule:

$$p(x, y|z) = \frac{p(x, y, z)}{p(z)} = \frac{p(x)p(y|x)p(z|x, y)}{\int p(x)p(y|x)p(z|x, y)\,dxdy}$$

Example PGM

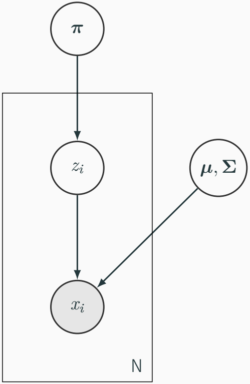
Latent variable model: Gaussian mixture model



$$
\begin{aligned}
\pi &\sim \mathrm{Dir}(\alpha) \\
\mu_k, \Sigma_k &\sim \mathcal{NIW}(\psi, \mu_0, \kappa, \nu) \\
z_i | \pi &\sim \mathrm{Cat}(\pi) \\
x_i | z_i, \boldsymbol{\mu}, \boldsymbol{\Sigma} &\sim \mathcal{N}(\mu_{z_i}, \Sigma_{z_i})
\end{aligned}
$$

Example PGM

Latent variable model: Gaussian mixture model

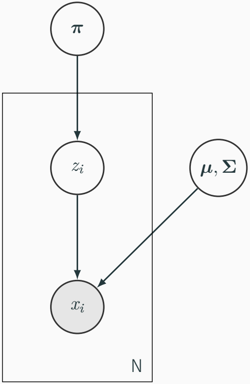


$$\begin{aligned}
\pi &\sim \mathrm{Dir}(\alpha) \\
\mu_k, \Sigma_k &\sim \mathcal{NIW}(\psi, \mu_0, \kappa, \nu) \\
z_i | \pi &\sim \mathrm{Cat}(\pi) \\
x_i | z_i, \boldsymbol{\mu}, \boldsymbol{\Sigma} &\sim \mathcal{N}(\mu_{z_i}, \Sigma_{z_i})
\end{aligned}$$

Local variables: $\{z_i\}_{i=1}^N$

Example PGM

Latent variable model: Gaussian mixture model



$$
\begin{aligned}
\pi &\sim \mathrm{Dir}(\alpha) \\
\mu_k, \Sigma_k &\sim \mathcal{NIW}(\psi, \mu_0, \kappa, \nu) \\
z_i | \pi &\sim \mathrm{Cat}(\pi) \\
x_i | z_i, \boldsymbol{\mu}, \boldsymbol{\Sigma} &\sim \mathcal{N}(\mu_{z_i}, \Sigma_{z_i})
\end{aligned}
$$

Local variables: $\{z_i\}_{i=1}^N$

Global variables: $\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}$

Conjugacy

Conjugate

Two random variables x and y whose distribution is

$$p(x, y) = p(x)p(y|x)$$

are said to be *conjugate* if the posterior $p(x|y)$ is in the same family of distributions as $p(x)$.

Conjugacy

Conjugate

Two random variables x and y whose distribution is

$$p(x, y) = p(x)p(y|x)$$

are said to be *conjugate* if the posterior $p(x|y)$ is in the same family of distributions as $p(x)$.

Examples of conjugate distributions:

- Normal/normal

Conjugacy

Conjugate

Two random variables x and y whose distribution is

$$p(x, y) = p(x)p(y|x)$$

are said to be *conjugate* if the posterior $p(x|y)$ is in the same family of distributions as $p(x)$.

Examples of conjugate distributions:

- Normal/normal
- Normal-inverse-wishart(NIW)/normal

Conjugacy

Conjugate

Two random variables x and y whose distribution is

$$p(x, y) = p(x)p(y|x)$$

are said to be *conjugate* if the posterior $p(x|y)$ is in the same family of distributions as $p(x)$.

Examples of conjugate distributions:

- Normal/normal
- Normal-inverse-wishart(NIW)/normal
- Dirichlet/multinomial

Exponential family

A probability distribution is in the *exponential family* if it can be parametrized in the following way.

Exponential family

A probability distribution is in the *exponential family* if it can be parametrized in the following way.

$$p(x|\theta) = h(x) \exp \left\{ \langle \eta(\theta), t_x(x) \rangle - \log Z(\eta(\theta)) \right\}$$

where

Exponential family

A probability distribution is in the *exponential family* if it can be parametrized in the following way.

$$p(x|\theta) = h(x) \exp\left\{\langle \eta(\theta), t_x(x) \rangle - \log Z(\eta(\theta))\right\}$$

where

- $h(x)$ - base measure

Exponential family

A probability distribution is in the *exponential family* if it can be parametrized in the following way.

$$p(x|\theta) = h(x) \exp \left\{ \langle \eta(\theta), t_x(x) \rangle - \log Z(\eta(\theta)) \right\}$$

where

- $h(x)$ - base measure
- $\eta(\theta)$ - natural parameter function

Exponential family

A probability distribution is in the *exponential family* if it can be parametrized in the following way.

$$p(x|\theta) = h(x) \exp\left\{\langle \eta(\theta), t_x(x) \rangle - \log Z(\eta(\theta))\right\}$$

where

- $h(x)$ - base measure
- $\eta(\theta)$ - natural parameter function
- $t_x(x)$ - sufficient statistic function

Exponential family

A probability distribution is in the *exponential family* if it can be parametrized in the following way.

$$p(x|\theta) = h(x)\exp\left\{\langle \eta(\theta), t_x(x)\rangle - \log Z(\eta(\theta))\right\}$$

where

- $h(x)$ - base measure
- $\eta(\theta)$ - natural parameter function
- $t_x(x)$ - sufficient statistic function
- $\log Z(\eta(\theta))$ - log-partition function

Exponential family

A probability distribution is in the *exponential family* if it can be parametrized in the following way.

$$p(x|\theta) = h(x) \exp\left\{\langle \eta(\theta), t_x(x) \rangle - \log Z(\eta(\theta))\right\}$$

where

- $h(x)$ - base measure
- $\eta(\theta)$ - natural parameter function
- $t_x(x)$ - sufficient statistic function
- $\log Z(\eta(\theta))$ - log-partition function

Examples: Gaussian, Categorical, Dirichlet, inverse-Wishart

Algorithms for inference

Depending on the structure of the graphical model and conjugacy of the variables, computing the posterior distribution can be easy, or very hard!

Algorithms for inference

Depending on the structure of the graphical model and conjugacy of the variables, computing the posterior distribution can be easy, or very hard!

Possible ways:

- Compute posterior analytically

Algorithms for inference

Depending on the structure of the graphical model and conjugacy of the variables, computing the posterior distribution can be easy, or very hard!

Possible ways:

- Compute posterior analytically
- Sampling (MCMC, Gibbs sampling)

Algorithms for inference

Depending on the structure of the graphical model and conjugacy of the variables, computing the posterior distribution can be easy, or very hard!

Possible ways:

- Compute posterior analytically
- Sampling (MCMC, Gibbs sampling)
- Expectation-maximization

Algorithms for inference

Depending on the structure of the graphical model and conjugacy of the variables, computing the posterior distribution can be easy, or very hard!

Possible ways:

- Compute posterior analytically
- Sampling (MCMC, Gibbs sampling)
- Expectation-maximization
- Variational inference

Algorithms for inference

Depending on the structure of the graphical model and conjugacy of the variables, computing the posterior distribution can be easy, or very hard!

Possible ways:

- Compute posterior analytically
- Sampling (MCMC, Gibbs sampling)
- Expectation-maximization
- Variational inference

Variational inference at a high level

For graphical models where we can't compute the posterior analytically, **variational inference** is a viable approach.

Variational inference at a high level

For graphical models where we can't compute the posterior analytically, **variational inference** is a viable approach.

Consider a latent variable model with global variables θ, local variables z and observations x. Our desired posterior is $p(\theta, z|x)$

Variational inference at a high level

For graphical models where we can't compute the posterior analytically, **variational inference** is a viable approach.

Consider a latent variable model with global variables θ, local variables z and observations x. Our desired posterior is $p(\theta, z|x)$

Strategy: convert inference into optimization

Variational inference at a high level

For graphical models where we can't compute the posterior analytically, **variational inference** is a viable approach.

Consider a latent variable model with global variables θ, local variables z and observations x. Our desired posterior is $p(\theta, z|x)$

Strategy: convert inference into optimization

- Instantiate *variational distribution* $q_\phi(\theta, z)$ where ϕ are free parameters

Variational inference at a high level

For graphical models where we can't compute the posterior analytically, **variational inference** is a viable approach.

Consider a latent variable model with global variables θ, local variables z and observations x. Our desired posterior is $p(\theta, z|x)$

Strategy: convert inference into optimization

- Instantiate *variational distribution* $q_\phi(\theta, z)$ where ϕ are free parameters
- Define loss $\mathrm{KL}(q_\phi(\theta, z) \| p(\theta, z|x))$

Variational inference at a high level

For graphical models where we can't compute the posterior analytically, **variational inference** is a viable approach.

Consider a latent variable model with global variables θ, local variables z and observations x. Our desired posterior is $p(\theta, z|x)$

Strategy: convert inference into optimization

- Instantiate *variational distribution* $q_\phi(\theta, z)$ where ϕ are free parameters
- Define loss $\mathrm{KL}(q_\phi(\theta, z) \| p(\theta, z|x))$
- Minimize loss $\phi^* = \operatorname{argmin}_\phi \mathrm{KL}(q_\phi(\theta, z) \| p(\theta, z|x))$

Variational inference at a high level

For graphical models where we can't compute the posterior analytically, **variational inference** is a viable approach.

Consider a latent variable model with global variables θ, local variables z and observations x. Our desired posterior is $p(\theta, z|x)$

Strategy: convert inference into optimization

- Instantiate *variational distribution* $q_\phi(\theta, z)$ where ϕ are free parameters
- Define loss $\mathrm{KL}(q_\phi(\theta, z) \| p(\theta, z|x))$
- Minimize loss $\phi^* = \mathrm{argmin}_\phi \, \mathrm{KL}(q_\phi(\theta, z) \| p(\theta, z|x))$

If $q(\theta, z)$ is sufficiently expressive, it can approximate $p(\theta, z|x)$ quite well.

KL-divergence

Kullback-Leibler (KL) divergence is a measure of how far one probability distribution is from another.

KL-divergence

Kullback-Leibler (KL) divergence is a measure of how far one probability distribution is from another.

For distributions $q(x)$ and $p(x)$,

$$\mathrm{KL}(q(x) \| p(x)) = \int q(x) \log \frac{q(x)}{p(x)} dx = \mathbb{E}_{q(x)} \left[\log \frac{q(x)}{p(x)} \right]$$

KL-divergence

Kullback-Leibler (KL) divergence is a measure of how far one probability distribution is from another.

For distributions $q(x)$ and $p(x)$,

$$\mathrm{KL}(q(x) \| p(x)) = \int q(x) \log \frac{q(x)}{p(x)} dx = \mathbb{E}_{q(x)} \left[\log \frac{q(x)}{p(x)} \right]$$

Properties:

- $\mathrm{KL}(q(x) \| p(x)) = 0$ if $q(x) = p(x)$.
- Asymmetric

Evidence lower bound

In general, we cannot even compute $KL(q(\theta, z) \| p(\theta, z|x))$ because we don't know the posterior $p(\theta, z|x)$.

Evidence lower bound

In general, we cannot even compute $KL(q(\theta, z) \| p(\theta, z|x))$ because we don't know the posterior $p(\theta, z|x)$.

We can rewrite the KL divergence as

$$\begin{aligned}
\mathrm{KL}(q(\theta, z) \| p(\theta, z|x)) &= \int q(\theta, z) \log \frac{q(\theta, z)}{p(\theta, z|x)} \, d\theta, z \\
&= \log p(x) - \mathbb{E}_{q(\theta, z)} \left[\log \frac{p(x, \theta, z)}{q(\theta, z)} \right]
\end{aligned}$$

Evidence lower bound

In general, we cannot even compute $KL(q(\theta, z)\|p(\theta, z|x))$ because we don't know the posterior $p(\theta, z|x)$.

We can rewrite the KL divergence as

$$\begin{aligned}
\mathrm{KL}(q(\theta, z)\|p(\theta, z|x)) &= \int q(\theta, z) \log \frac{q(\theta, z)}{p(\theta, z|x)} \, d\theta, z \\
&= \log p(x) - \mathbb{E}_{q(\theta,z)} \left[\log \frac{p(x, \theta, z)}{q(\theta, z)} \right]
\end{aligned}$$

and maximize the *evidence lower bound* (ELBO)

$$\mathcal{L}[q(\theta, z)] = \mathbb{E}_{q(\theta,z)} \left[\log \frac{p(x, \theta, z)}{q(\theta, z)} \right]$$

Picking $q(\theta, x)$

How do we pick a $q(\theta, x)$?

Picking $q(\theta, x)$

How do we pick a $q(\theta, x)$?

In general, a broader $q(\theta, x)$ is harder to optimize.

Options:

Picking $q(\theta, x)$

How do we pick a $q(\theta, x)$?

In general, a broader $q(\theta, x)$ is harder to optimize.

Options:

- Mean-field $\big(q(\theta, x) = q(\theta)\, q(x)\big)$

Picking $q(\theta, x)$

How do we pick a $q(\theta, x)$?

In general, a broader $q(\theta, x)$ is harder to optimize.

Options:

- Mean-field $\big(q(\theta, x) = q(\theta)\, q(x)\big)$
- Conjugate-exponential

Picking $q(\theta, x)$

How do we pick a $q(\theta, x)$?

In general, a broader $q(\theta, x)$ is harder to optimize.

Options:

- Mean-field $\big(q(\theta, x) = q(\theta)\,q(x)\big)$
- Conjugate-exponential
- Differentiable q

Mean-field variational inference

For a general graphical model with variables $\mathbf{X} = \{x_1, x_2, \ldots\}$ we have joint distribution

$$p(\mathbf{X}) = \prod_i p(x_i|\mathrm{pa}_i)$$

where pa_i are the parents of node x_i in the graph.

Mean-field variational inference

For a general graphical model with variables $\mathbf{X} = \{x_1, x_2, \ldots\}$ we have joint distribution

$$p(\mathbf{X}) = \prod_i p(x_i|\mathrm{pa}_i)$$

where pa_i are the parents of node x_i in the graph.

Let the set $\mathbf{H}$ be all unobserved variables and $\mathbf{V}$ be the observed.

Mean-field variational inference

For a general graphical model with variables $\mathbf{X} = \{x_1, x_2, \ldots\}$ we have joint distribution

$$p(\mathbf{X}) = \prod_i p(x_i | \mathrm{pa}_i)$$

where pa_i are the parents of node x_i in the graph.

Let the set $\mathbf{H}$ be all unobserved variables and $\mathbf{V}$ be the observed.

We are interested in the posterior $p(\mathbf{H}|\mathbf{V})$ and use variational distribution

$$q(\mathbf{H}) = \prod_i q(\mathbf{H}_i)$$

Mean-field variational inference (cont.)

The ELBO is now

$$\begin{aligned}
\mathcal{L}[q(\mathbf{H})] &= \mathbb{E}_{q(\mathbf{H})}\left[\log \frac{p(\mathbf{H}, \mathbf{V})}{q(\mathbf{H})}\right] \\
&= \int \prod_i q(\mathbf{H}_i) \left(\log p(\mathbf{H}, \mathbf{V}) - \log \prod_i q(\mathbf{H}_i)\right) d\mathbf{H} \\
&= \int q(\mathbf{H}_j) \left(\int \log p(\mathbf{H}, \mathbf{V}) \prod_{i \neq j} q(\mathbf{H}_i)\, d\mathbf{H}_i\right) d\mathbf{H}_j \\
&- \int q(\mathbf{H}_j) \log q(\mathbf{H}_j)\, d\mathbf{H}_j + \text{const.} \\
&= \int q(\mathbf{H}_j) \log \tilde{q}(\mathbf{H}_j, \mathbf{V})\, d\mathbf{H}_j - \int q(\mathbf{H}_j) \log q(\mathbf{H}_j)\, d\mathbf{H}_j + \text{const.} \\
&= -\mathrm{KL}(q(\mathbf{H}_j) \| \tilde{q}(\mathbf{H}_j, \mathbf{V})) + \text{const.}
\end{aligned}$$

where

$$\log \tilde{q}(\mathbf{H}_j, \mathbf{V}) = \mathbb{E}_{i \neq j}\left[\log p(\mathbf{H}, \mathbf{V})\right] + \text{const.}$$

Optimizing a single factor

We can isolate a single factor for each hidden node in the graph $\mathbf{H}_j$. This makes optimizing a single variational factor easy!

Optimizing a single factor

We can isolate a single factor for each hidden node in the graph $\mathbf{H}_j$.
This makes optimizing a single variational factor easy!

$$\mathcal{L}[q(\mathbf{H})] = -\mathrm{KL}(q(\mathbf{H}_j) \| \tilde{q}(\mathbf{H}_j, \mathbf{V})) + \mathrm{const.}$$

Optimizing a single factor

We can isolate a single factor for each hidden node in the graph $\mathbf{H}_j$. This makes optimizing a single variational factor easy!

$$\mathcal{L}[q(\mathbf{H})] = -\mathrm{KL}(q(\mathbf{H}_j) \| \tilde{q}(\mathbf{H}_j, \mathbf{V})) + \mathrm{const.}$$

For a single factor $q(\mathbf{H}_j)$, this equation is minimized when $q(\mathbf{H}_j) = \tilde{q}(\mathbf{H}_j, \mathbf{V})$.

Optimizing a single factor

We can isolate a single factor for each hidden node in the graph $\mathbf{H}_j$.
This makes optimizing a single variational factor easy!

$$\mathcal{L}[q(\mathbf{H})] = -\mathrm{KL}(q(\mathbf{H}_j) \| \tilde{q}(\mathbf{H}_j, \mathbf{V})) + \mathrm{const.}$$

For a single factor $q(\mathbf{H}_j)$, this equation is minimized when $q(\mathbf{H}_j) = \tilde{q}(\mathbf{H}_j, \mathbf{V})$.

Furthermore,

$$\log \tilde{q}(\mathbf{H}_j, \mathbf{V}) = \mathbb{E}_{i \neq j} \left[\log p(\mathbf{H}, \mathbf{V})\right] + \mathrm{const.}$$

is a function of only factors other than $q(\mathbf{H}_j)$ and observed data.

Optimizing a single factor

We can isolate a single factor for each hidden node in the graph $\mathbf{H}_j$. This makes optimizing a single variational factor easy!

$$\mathcal{L}[q(\mathbf{H})] = -\mathrm{KL}(q(\mathbf{H}_j) \| \tilde{q}(\mathbf{H}_j, \mathbf{V})) + \mathrm{const.}$$

For a single factor $q(\mathbf{H}_j)$, this equation is minimized when $q(\mathbf{H}_j) = \tilde{q}(\mathbf{H}_j, \mathbf{V})$.

Furthermore,

$$\log \tilde{q}(\mathbf{H}_j, \mathbf{V}) = \mathbb{E}_{i \neq j}\left[\log p(\mathbf{H}, \mathbf{V})\right] + \mathrm{const.}$$

is a function of only factors other than $q(\mathbf{H}_j)$ and observed data.

Mean-field variational inference

Until converged, for each factor $q(\mathbf{H}_j)$, hold factors $q(\mathbf{H}_{i \neq j})$ constant and set $q(\mathbf{H}_j) = \tilde{q}(\mathbf{H}_j, \mathbf{V})$.

Conjugate-exponential graphical models

If our model is *conjugate-exponential*, where every node belongs in the exponential family of distributions, and is conjugate w.r.t. its parents,

$$\log \tilde{q}(\mathbf{H}_j, \mathbf{V}) = \mathbb{E}_{i \neq j} \left[\log p(\mathbf{H}, \mathbf{V}) \right] + \text{const.}$$

can be calculated in closed form!

Conjugate-exponential graphical models

If our model is *conjugate-exponential*, where every node belongs in the exponential family of distributions, and is conjugate w.r.t. its parents,

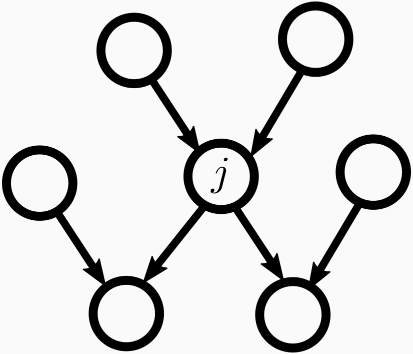
$$\log \tilde{q}(\mathbf{H}_j, \mathbf{V}) = \mathbb{E}_{i \neq j} [\log p(\mathbf{H}, \mathbf{V})] + \text{const.}$$

can be calculated in closed form!

This gives rise to an efficient coordinate-ascent algorithm that can be extended with natural gradients (stochastic variational inference).

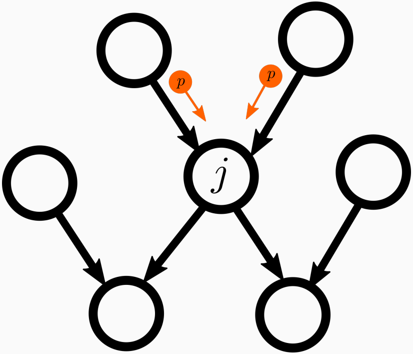
Variational message passing

An update for node j relies on only its Markov blanket.



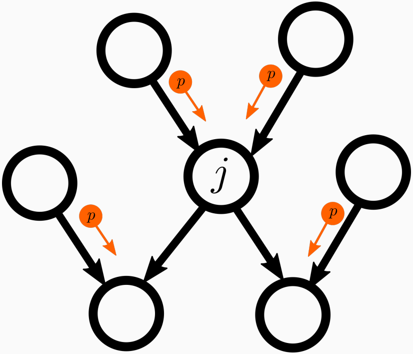
Variational message passing

An update for node j relies on only its Markov blanket.



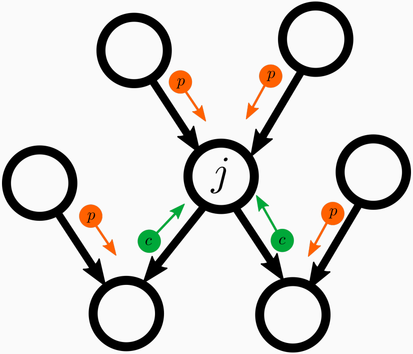
Variational message passing

An update for node j relies on only its Markov blanket.



Variational message passing

An update for node j relies on only its Markov blanket.



Messages

Remember that η_j is the natural parameter of the distribution $q(\mathbf{H}_j)$.

Messages

Remember that η_j is the natural parameter of the distribution $q(\mathbf{H}_j)$. The message from a parent Y to a child X is

$$m_{Y \to X} = f(\eta_Y)$$

Messages

Remember that η_j is the natural parameter of the distribution $q(\mathbf{H}_j)$. The message from a parent Y to a child X is

$$m_{Y\to X} = f(\eta_Y)$$

The message from a child X to a parent Y is

$$m_{X\to Y} = g(\eta_X, \{m_{i\to X}\}_{i\in \mathrm{cp}_Y})$$

Messages

Remember that η_j is the natural parameter of the distribution $q(\mathbf{H}_j)$.
The message from a parent Y to a child X is

$$m_{Y \to X} = f(\eta_Y)$$

The message from a child X to a parent Y is

$$m_{X \to Y} = g(\eta_X, \{m_{i \to X}\}_{i \in \mathrm{cp}_Y})$$

In conjugate-exponential PGMs, messages can be computed in closed-form.

Summary of VMP

Setup: Conjugate-exponential graphical model

Summary of VMP

Setup: Conjugate-exponential graphical model

Problem: Compute posterior $p(\mathbf{H}|\mathbf{V})$, approximated with $q(\mathbf{H}) = \prod_j q(\mathbf{H}_j)$

Summary of VMP

Setup: Conjugate-exponential graphical model

Problem: Compute posterior $p(\mathbf{H}|\mathbf{V})$, approximated with $q(\mathbf{H}) = \prod_j q(\mathbf{H}_j)$

Solution: Until converged, for each hidden node $\mathbf{H}_j$:

Summary of VMP

Setup: Conjugate-exponential graphical model

Problem: Compute posterior $p(\mathbf{H}|\mathbf{V})$, approximated with $q(\mathbf{H}) = \prod_j q(\mathbf{H}_j)$

Solution: Until converged, for each hidden node $\mathbf{H}_j$:

1. Collect messages from children and parents

Summary of VMP

Setup: Conjugate-exponential graphical model

Problem: Compute posterior $p(\mathbf{H}|\mathbf{V})$, approximated with $q(\mathbf{H}) = \prod_j q(\mathbf{H}_j)$

Solution: Until converged, for each hidden node $\mathbf{H}_j$:

1. Collect messages from children and parents
2. Compute updated distribution parameters from messages

Summary of VMP

Setup: Conjugate-exponential graphical model

Problem: Compute posterior $p(\mathbf{H}|\mathbf{V})$, approximated with $q(\mathbf{H}) = \prod_j q(\mathbf{H}_j)$

Solution: Until converged, for each hidden node $\mathbf{H}_j$:

1. Collect messages from children and parents
2. Compute updated distribution parameters from messages

Benefits: efficient, simple, can incorporate mini-batches (stochastic variational inference)

Summary of VMP

Setup: Conjugate-exponential graphical model

Problem: Compute posterior $p(\mathbf{H}|\mathbf{V})$, approximated with $q(\mathbf{H}) = \prod_j q(\mathbf{H}_j)$

Solution: Until converged, for each hidden node $\mathbf{H}_j$:

1. Collect messages from children and parents
2. Compute updated distribution parameters from messages

Benefits: efficient, simple, can incorporate mini-batches (stochastic variational inference)

Drawbacks: can be underexpressive (conjugate-exponential requirement)

Demo

Monte-Carlo approach

The previous approach was *restrictive*! Coordinate ascent only works on conjugate-exponential models with the mean-field assumption.

Monte-Carlo approach

The previous approach was *restrictive*! Coordinate ascent only works on conjugate-exponential models with the mean-field assumption.

Assume a non-conjugate, non-exponential model and a differentiable, sampleable $q(\theta, z)$.

Monte-Carlo approach

The previous approach was *restrictive*! Coordinate ascent only works on conjugate-exponential models with the mean-field assumption.

Assume a non-conjugate, non-exponential model and a differentiable, sampleable $q(\theta, z)$.

$$\mathcal{L}[q(\theta, z)] = \mathbb{E}_{q(\theta,z)}\left[\log \frac{p(x, \theta, z)}{q(\theta, z)}\right]$$

Monte-Carlo approach

The previous approach was *restrictive*! Coordinate ascent only works on conjugate-exponential models with the mean-field assumption.

Assume a non-conjugate, non-exponential model and a differentiable, sampleable $q(\theta, z)$.

$$\mathcal{L}[q(\theta, z)] = \mathbb{E}_{q(\theta,z)}\left[\log \frac{p(x, \theta, z)}{q(\theta, z)}\right]$$

Perform Monte-Carlo estimate:

$$\hat{\mathcal{L}}[q(\theta, z)] = \sum_{l=1}^{L} q(\theta^{(l)}, z^{(l)})\left[\log \frac{p(x, \theta^{(l)}, z^{(l)})}{q(\theta^{(l)}, z^{(l)})}\right]$$

Gradient-based approaches

1. Start with Monte-Carlo loss

$$\hat{\mathcal{L}}[q(\theta, z)] = \frac{1}{S} \sum_{s=1}^{S} q(\theta^{(s)}, z^{(s)}) \left[\log \frac{p(x, \theta^{(s)}, z^{(s)})}{q(\theta^{(s)}, z^{(s)})} \right]$$

2. Compute gradient

$$\nabla_\phi \hat{\mathcal{L}}[q_\phi(\theta, z)] = \frac{1}{S} \sum_{s=1}^{S} \nabla_\phi q(\theta^{(s)}, z^{(s)}) \left[\log \frac{p(x, \theta^{(s)}, z^{(s)})}{q(\theta^{(s)}, z^{(s)})} \right]$$

3. Perform gradient ascent

Issues with gradient-based approaches

Core problem: high-variance gradients

Issues with gradient-based approaches

Core problem: high-variance gradients

How do we address this?

- Rao-Blackwellization (replace $\mathbb{E}[f(X, Y)]$ with $\mathbb{E}[f(X, Y)|X]$)[1]
- Reparametrization trick [2]

Variational autoencoder

The VAE is a generative model for data [2].

$$z_i \sim \mathcal{N}(0, I)$$
$$x_i \sim \mathcal{N}(\mu_\gamma(z_i), \Sigma_\gamma(z_i))$$

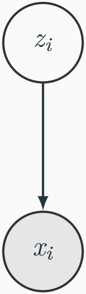
where μ_γ and Σ_γ are neural networks parameterized by γ.

Variational autoencoder

The VAE is a generative model for data [2].

$$z_i \sim \mathcal{N}(0, I)$$
$$x_i \sim \mathcal{N}(\mu_\gamma(z_i), \Sigma_\gamma(z_i))$$

where μ_γ and Σ_γ are neural networks parameterized by γ.

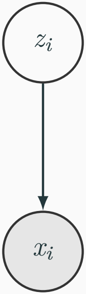


Variational autoencoder

The VAE is a generative model for data [2].

$$z_i \sim \mathcal{N}(0, I)$$
$$x_i \sim \mathcal{N}(\mu_\gamma(z_i), \Sigma_\gamma(z_i))$$

where μ_γ and Σ_γ are neural networks parameterized by γ.



How do we do inference?

Variational autoencoder

Basic strategy: gradient-based variational inference

Variational autoencoder

Basic strategy: gradient-based variational inference

- Pick $q_\phi(z|x)$ to be a *neural network* parameterized by ϕ (both differentiable and sampleable)

Variational autoencoder

Basic strategy: gradient-based variational inference

- Pick $q_\phi(z|x)$ to be a *neural network* parameterized by ϕ (both differentiable and sampleable)

Let $r_\phi(x)$ be a neural network (with weights ϕ)that outputs the parameters to a distribution, for example Gaussian.

$$q_\phi(z|x) = \mathcal{N}(r_\phi(x))$$

Variational autoencoder

Basic strategy: gradient-based variational inference

- Pick $q_\phi(z|x)$ to be a *neural network* parameterized by ϕ (both differentiable and sampleable)

Let $r_\phi(x)$ be a neural network (with weights ϕ)that outputs the parameters to a distribution, for example Gaussian.

$$q_\phi(z|x) = \mathcal{N}(r_\phi(x))$$

- Use the reparametrization trick to lower variance of gradients

Variational autoencoder

Basic strategy: gradient-based variational inference

- Pick $q_\phi(z|x)$ to be a *neural network* parameterized by ϕ (both differentiable and sampleable)

Let $r_\phi(x)$ be a neural network (with weights ϕ)that outputs the parameters to a distribution, for example Gaussian.

$$q_\phi(z|x) = \mathcal{N}(r_\phi(x))$$

- Use the reparametrization trick to lower variance of gradients

We learn the weights for the two neural networks $(\mu_\gamma(z), \Sigma_\gamma(z))$ and $r_\phi(x)$.

Variational autoencoder

Basic strategy: gradient-based variational inference

- Pick $q_\phi(z|x)$ to be a *neural network* parameterized by ϕ (both differentiable and sampleable)

Let $r_\phi(x)$ be a neural network (with weights ϕ)that outputs the parameters to a distribution, for example Gaussian.

$$q_\phi(z|x) = \mathcal{N}(r_\phi(x))$$

- Use the reparametrization trick to lower variance of gradients

We learn the weights for the two neural networks $(\mu_\gamma(z), \Sigma_\gamma(z))$ and $r_\phi(x)$.

$$\mathcal{L}[q(z|x)] = \mathbb{E}_{q(z|x)}\left[\log p(x,z) - \log q(z|x)\right]$$

Reparametrization trick

What is the reparametrization trick?

Reparametrization trick

What is the reparametrization trick? We want to sample $z \sim q(z|x)$ and do so it in a roundabout way.

$$\epsilon \sim p(\epsilon)$$
$$z = f(r_\phi(x), \epsilon)$$

where ϵ is noise sampled from a simple distribution (e.g. Gaussian) and f is a function that depends on the type of distribution.

Reparametrization trick

What is the reparametrization trick? We want to sample $z \sim q(z|x)$ and do so it in a roundabout way.

$$\epsilon \sim p(\epsilon)$$
$$z = f(r_\phi(x), \epsilon)$$

where ϵ is noise sampled from a simple distribution (e.g. Gaussian) and f is a function that depends on the type of distribution.

Example: sampling from a univariate Gaussian $q_\phi(z|x)$

Reparametrization trick

What is the reparametrization trick? We want to sample $z \sim q(z|x)$ and do so it in a roundabout way.

$$\epsilon \sim p(\epsilon)$$
$$z = f(r_\phi(x), \epsilon)$$

where ϵ is noise sampled from a simple distribution (e.g. Gaussian) and f is a function that depends on the type of distribution.

Example: sampling from a univariate Gaussian $q_\phi(z|x)$

- Compute $(\mu, \sigma^2) = r(x)$

Reparametrization trick

What is the reparametrization trick? We want to sample $z \sim q(z|x)$ and do so it in a roundabout way.

$$\epsilon \sim p(\epsilon)$$
$$z = f(r_\phi(x), \epsilon)$$

where ϵ is noise sampled from a simple distribution (e.g. Gaussian) and f is a function that depends on the type of distribution.

Example: sampling from a univariate Gaussian $q_\phi(z|x)$

- Compute $(\mu, \sigma^2) = r(x)$
- Sample $\epsilon \sim N(0, 1))$

Reparametrization trick

What is the reparametrization trick? We want to sample $z \sim q(z|x)$ and do so it in a roundabout way.

$$\epsilon \sim p(\epsilon)$$
$$z = f(r_\phi(x), \epsilon)$$

where ϵ is noise sampled from a simple distribution (e.g. Gaussian) and f is a function that depends on the type of distribution.

Example: sampling from a univariate Gaussian $q_\phi(z|x)$

- Compute $(\mu, \sigma^2) = r(x)$
- Sample $\epsilon \sim N(0, 1))$
- Return $z = \mu + \epsilon\sigma$

Reparametrization trick

What is the reparametrization trick? We want to sample $z \sim q(z|x)$ and do so it in a roundabout way.

$$\epsilon \sim p(\epsilon)$$
$$z = f(r_\phi(x), \epsilon)$$

where ϵ is noise sampled from a simple distribution (e.g. Gaussian) and f is a function that depends on the type of distribution.

Example: sampling from a univariate Gaussian $q_\phi(z|x)$

- Compute $(\mu, \sigma^2) = r(x)$
- Sample $\epsilon \sim N(0, 1))$
- Return $z = \mu + \epsilon\sigma$

This is very effective at decreasing the variance of the gradient.

Structured variational autoencoder

What is an SVAE?

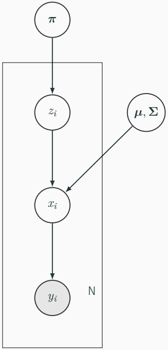
There are two ways of thinking about it

- A conjugate-exponential graphical model augmented with a neural network observation model
- A VAE augmented with also a graphical because of a neural-network observation model

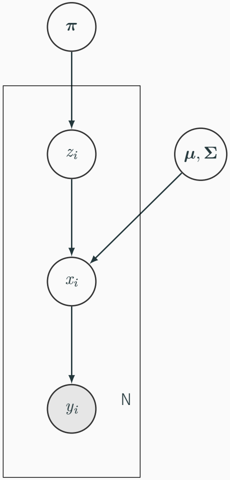
What is an SVAE?

There are two ways of thinking about it

- A conjugate-exponential graphical model augmented with a neural network observation model
- A VAE augmented with also a graphical because of a neural-network observation model

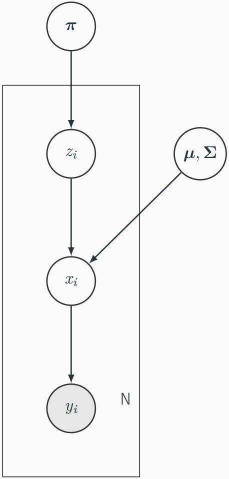


SVAE GMM



$$
\begin{aligned}
\pi &\sim \mathrm{Dir}(\alpha) \\
\mu_k, \Sigma_k &\sim \mathcal{NIW}(\psi, \mu_0, \kappa, \nu) \\
z_i | \pi &\sim \mathrm{Cat}(\pi) \\
x_i | z_i, \boldsymbol{\mu}, \boldsymbol{\Sigma} &\sim \mathcal{N}(\mu_{z_i}, \Sigma_{z_i}) \\
y_i | x_i &\sim \mathcal{N}(\mu_\gamma(x_i), \Sigma(x_i))
\end{aligned}
$$

SVAE GMM



$$
\begin{aligned}
\pi &\sim \mathrm{Dir}(\alpha) \\
\mu_k, \Sigma_k &\sim \mathcal{NIW}(\psi, \mu_0, \kappa, \nu) \\
z_i | \pi &\sim \mathrm{Cat}(\pi) \\
x_i | z_i, \boldsymbol{\mu}, \boldsymbol{\Sigma} &\sim \mathcal{N}(\mu_{z_i}, \Sigma_{z_i}) \\
y_i | x_i &\sim \mathcal{N}(\mu_\gamma(x_i), \Sigma(x_i))
\end{aligned}
$$

How do we do inference?

Inference in SVAE

Inference in SVAE is a hybrid of gradient-based methods and coordinate-ascent.

Inference in SVAE

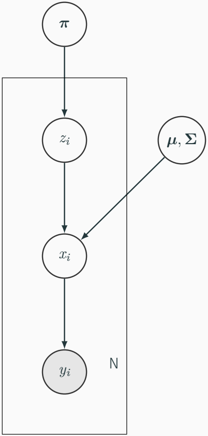
Inference in SVAE is a hybrid of gradient-based methods and coordinate-ascent.

How do we merge these two different approaches?

Inference in SVAE

Inference in SVAE is a hybrid of gradient-based methods and coordinate-ascent.

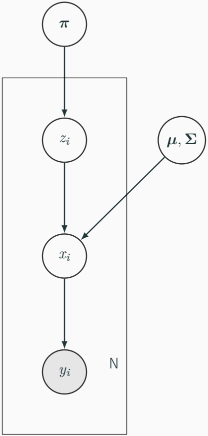
How do we merge these two different approaches?



Inference in SVAE

Inference in SVAE is a hybrid of gradient-based methods and coordinate-ascent.

How do we merge these two different approaches?

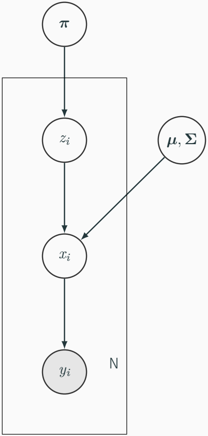


- Learn $q(z_i), q(x_i), q(\pi), q(\mu, \Sigma)$ with stochastic VMP using a neural network $r_\phi(y) = m_{y_i \to x_i}$

Inference in SVAE

Inference in SVAE is a hybrid of gradient-based methods and coordinate-ascent.

How do we merge these two different approaches?



- Learn $q(z_i), q(x_i), q(\pi), q(\mu, \Sigma)$ with stochastic VMP using a neural network $r_\phi(y) = m_{y_i \to x_i}$
- Learn weights of neural networks $(\mu_\gamma, \Sigma_\gamma), r_\phi$ using the gradient of the ELBO

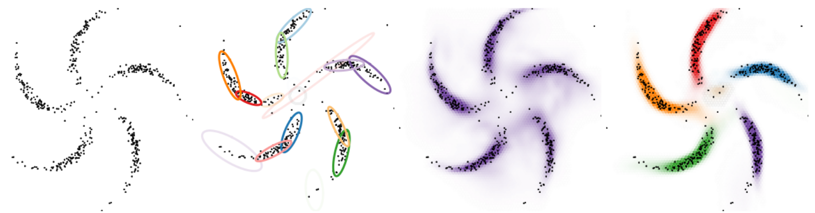
Conclusion

Why SVAE?

The SVAE naturally applies to scenarios where there is already a tractable model.

Why SVAE?

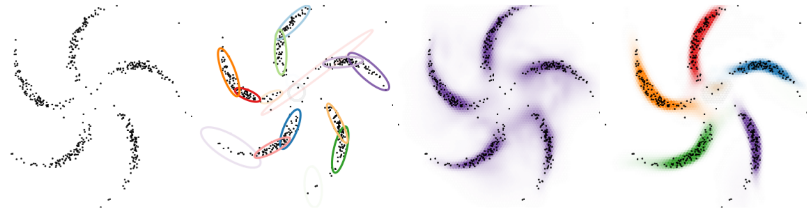
The SVAE naturally applies to scenarios where there is already a tractable model.



(a) Data (b) GMM (c) Density net (VAE) (d) GMM SVAE

Why SVAE?

The SVAE naturally applies to scenarios where there is already a tractable model.



(a) Data (b) GMM (c) Density net (VAE) (d) GMM SVAE

In this scenario, the SVAE enables modeling non-Gaussian cluster shapes [3].

Current work

One idea I am currently working on is using the SVAE to learn latent models to be used in reinforcement learning[1].

[1] Joint work with Marvin Zhang from UC Berkeley

Current work

One idea I am currently working on is using the SVAE to learn latent models to be used in reinforcement learning[1].

General approach:

- Learn a latent LDS
- Use MPC to control an agent in the latent space

[1] Joint work with Marvin Zhang from UC Berkeley

Current work

One idea I am currently working on is using the SVAE to learn latent models to be used in reinforcement learning[1].

General approach:

- Learn a latent LDS
- Use MPC to control an agent in the latent space

Benefits of this approach:

[1]Joint work with Marvin Zhang from UC Berkeley

Current work

One idea I am currently working on is using the SVAE to learn latent models to be used in reinforcement learning[1].

General approach:

- Learn a latent LDS
- Use MPC to control an agent in the latent space

Benefits of this approach:

- We can learn simple dynamics even with camera data

[1] Joint work with Marvin Zhang from UC Berkeley

Current work

One idea I am currently working on is using the SVAE to learn latent models to be used in reinforcement learning[1].

General approach:

- Learn a latent LDS
- Use MPC to control an agent in the latent space

Benefits of this approach:

- We can learn simple dynamics even with camera data
- Model-based RL tends to be more sample efficient

[1] Joint work with Marvin Zhang from UC Berkeley

Current work

One idea I am currently working on is using the SVAE to learn latent models to be used in reinforcement learning[1].

General approach:

- Learn a latent LDS
- Use MPC to control an agent in the latent space

Benefits of this approach:

- We can learn simple dynamics even with camera data
- Model-based RL tends to be more sample efficient

[1] Joint work with Marvin Zhang from UC Berkeley

Current work

One idea I am currently working on is using the SVAE to learn latent models to be used in reinforcement learning[1].

General approach:

- Learn a latent LDS
- Use MPC to control an agent in the latent space

Benefits of this approach:

- We can learn simple dynamics even with camera data
- Model-based RL tends to be more sample efficient

Demo

[1] Joint work with Marvin Zhang from UC Berkeley

Questions?

References

[1] Rajesh Ranganath, Sean Gerrish, and David M. Blei. Black Box Variational Inference. dec 2013.

[2] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. dec 2013.

[3] M. Johnson, D. Duvenaud, A. Wiltschko, S. Datta, and R. Adams. Composing graphical models with neural networks for structured representations and fast inference. In *NIPS*, 2016.