

# Deep Generative Models

Philip Schulz and Wilker Aziz

<https://github.com/philschulz/VITutorial>

# Generative Models

First Attempt: Log-linear Models

Second Attempt: Wake-Sleep

This is how we do: Variational Autoencoders

# Generative Models

First Attempt: Log-linear Models

Second Attempt: Wake-Sleep

This is how we do: Variational Autoencoders

# Recap: Generative Models

Joint distribution over observed data  $x$  and latent variables  $Z$ .

$$p(x, z|\alpha) = \overbrace{p(x|z, \alpha)}^{\text{likelihood}} \underbrace{p(z|\alpha)}_{\text{prior}}$$

The likelihood and prior are often standard distributions (Gaussian, Bernoulli) with simple dependence on conditioning information.

# Recap: Variational Inference

## Objective

$$\max_{q(z)} \mathbb{E} [\log p(x, z)] + \mathbb{H}(q(z))$$

- ▶ The ELBO is a lower bound on  $\log p(x)$
- ▶ Mean field assumption:  $q(z) = \prod_{i=1}^N q(z_i)$

# Generative Models

## First Attempt: Log-linear Models

## Second Attempt: Wake-Sleep

## This is how we do: Variational Autoencoders

# Feature-rich Generative Models

Let us assume that  $z$  has internal structure (features). How can we exploit that?

## First Idea

Make  $p(x|z, \alpha)$  a log-linear model.

- ▶ Only discrete data
- ▶ Trainable with EM if we can efficiently enumerate  $\mathcal{X}$  and  $\mathcal{Z}$ .

# Log-linear Model

Let us treat  $z$  as observed.

$$p(x|z, \alpha = w) = \frac{\exp(w^\top f(x, z))}{\sum_{x \in \mathcal{X}} \exp(w^\top f(x, z))}$$



# Log-linear Model

Let us treat  $z$  as observed.

$$p(x|z, \alpha = w) = \frac{\exp(w^\top f(x, z))}{\sum_{x \in \mathcal{X}} \exp(w^\top f(x, z))}$$

## Weight Gradient

$$\frac{d}{dw} \log p(x|z, w) = f(x, z) - \mathbb{E}[f(X, z)|z, w]$$

# Log-linear Model

Let us treat  $z$  as observed.

$$p(x|z, \alpha = \mathbf{w}) = \frac{\exp(\mathbf{w}^\top f(x, z))}{\sum_{x \in \mathcal{X}} \exp(\mathbf{w}^\top f(x, z))}$$

## Weight Gradient

$$\frac{d}{d\mathbf{w}} \log p(x|z, \mathbf{w}) = f(x, z) - \mathbb{E}[f(X, z)|z, \mathbf{w}]$$

Updates need to be performed iteratively.

# Log-linear model with latent variables

Now let us treat  $z$  as latent.

# Log-linear model with latent variables

Now let us treat  $z$  as latent.

Model

$$p(x, z|w) = \underbrace{\frac{\exp(w^\top f(x, z))}{\sum_{x \in \mathcal{X}} \exp(w^\top f(x, z))}}_{p(x|z, w)} \times \underbrace{p(z)}_{\text{arbitrary}}$$

# Log-linear model with latent variables

## Posterior

$$p(z|x, w)$$

# Log-linear model with latent variables

## Posterior

$$p(z|x, w) = \frac{p(x, z|w)}{p(x|w)}$$

# Log-linear model with latent variables

## Posterior

$$p(z|x, w) = \frac{p(x, z|w)}{p(x|w)} = \frac{p(x, z|w)}{\sum_z p(x, z|w)} =$$

# Log-linear model with latent variables

## Posterior

$$\begin{aligned} p(z|x, w) &= \frac{p(x, z|w)}{p(x|w)} = \frac{p(x, z|w)}{\sum_z p(x, z|w)} = \\ &= \frac{\frac{\exp(w^\top f(x, z))}{\sum_{x \in \mathcal{X}} \exp(w^\top f(x, z))} \times p(z)}{\sum_z \frac{\exp(w^\top f(x, z))}{\sum_{x \in \mathcal{X}} \exp(w^\top f(x, z))} \times p(z)} \end{aligned}$$



# Log-linear model with latent variables

Weight Gradient (treat  $p(z|x, w)$  as fixed)

$$\frac{d}{dw} \mathbb{E}_{p(z|x, w)} [\log p(x, z|w)] =$$

# Log-linear model with latent variables

Weight Gradient (treat  $p(z|x, w)$  as fixed)

$$\begin{aligned}\frac{d}{dw} \mathbb{E}_{p(z|x, w)} [\log p(x, z|w)] &= \\ \frac{d}{dw} \sum_z p(z|x, w) \log p(x, z|w) &= \end{aligned}$$

# Log-linear model with latent variables

Weight Gradient (treat  $p(z|x, w)$  as fixed)

$$\begin{aligned}\frac{d}{dw} \mathbb{E}_{p(z|x, w)} [\log p(x, z|w)] &= \\ \frac{d}{dw} \sum_z p(z|x, w) \log p(x, z|w) &= \\ \sum_z p(z|x, w) \frac{d}{dw} \log p(x, z|w) &= \end{aligned}$$

# Log-linear model with latent variables

Weight Gradient (treat  $p(z|x, w)$  as fixed)

$$\begin{aligned}\frac{d}{dw} \mathbb{E}_{p(z|x, w)} [\log p(x, z|w)] &= \\ \frac{d}{dw} \sum_z p(z|x, w) \log p(x, z|w) &= \\ \sum_z p(z|x, w) \frac{d}{dw} \log p(x, z|w) &= \\ \sum_z p(z|x, w) \left( \frac{d}{dw} \log p(x|z, w) + \frac{d}{dw} p(z) \right)\end{aligned}$$

# Log-linear model with latent variables

Weight Gradient (treat  $p(z|x, w)$  as fixed)

$$\frac{d}{dw} \mathbb{E}_{p(z|x, w)} [\log p(x, z|w)] =$$

$$\frac{d}{dw} \sum_z p(z|x, w) \log p(x, z|w) =$$

$$\sum_z p(z|x, w) \frac{d}{dw} \log p(x, z|w) =$$

$$\sum_z p(z|x, w) \left( \overbrace{\frac{d}{dw} \log p(x, z|w)}^{\text{We've already solved this!}} + \overbrace{\frac{d}{dw} p(z)}^0 \right)$$

# Log-linear model with latent variables

Weight Gradient (treat  $p(z|x, w)$  as fixed)

$$\frac{d}{dw} \mathbb{E}_{p(z|x, w)} [\log p(x, z|w)] =$$
$$\mathbb{E}_{p(z|x, w)} [f(x, Z)|x, w] - \mathbb{E}_{p(z|x, w)} [\mathbb{E} [(f(X, Z)|Z, w)]]$$

# Log-linear model with latent variables

Weight Gradient (treat  $p(z|x, w)$  as fixed)

$$\frac{d}{dw} \mathbb{E}_{p(z|x, w)} [\log p(x, z|w)] =$$

$$\mathbb{E}_{p(z|x, w)} [f(x, Z)|x, w] - \mathbb{E}_{p(z|x, w)} [\mathbb{E} [(f(X, Z)|Z, w)]]$$

Procedurally

$$E\_count(x, z) - \{ E\_count(x, z) \times \mathbb{E} [X|z, w] \}$$

# EM

E-step  $p(z|x, w) = \frac{p(x, z|w)}{\sum_z p(x, z|w)}$  in  $\mathcal{O}(|\mathcal{X}| \times |\mathcal{Z}|)$

M-step Iteratively optimise  $w$  to match  $\text{E\_count}(x, z)$   
with  $\text{E\_count}(x, z) \times \mathbb{E}[X|z, w]$

## Restrictions

- ▶ Only log-linear models
- ▶ Scales badly



# Generative Models

First Attempt: Log-linear Models

Second Attempt: Wake-Sleep

This is how we do: Variational Autoencoders

# Wake-sleep Algorithm

- ▶ Generalise latent variables to Neural Networks
- ▶ Train generative neural model
- ▶ Use variational inference! (kind of)

# Wake-sleep Architecture

2 Neural Networks:

# Wake-sleep Architecture

2 Neural Networks:

- ▶ A generation network to model the data (the one we want to optimise) – parameters:  $\theta$

# Wake-sleep Architecture

## 2 Neural Networks:

- ▶ A generation network to model the data (the one we want to optimise) – parameters:  $\theta$
- ▶ An inference (recognition) network (to model the latent variable) – parameters:  $\lambda$

# Wake-sleep Architecture

## 2 Neural Networks:

- ▶ A generation network to model the data (the one we want to optimise) – parameters:  $\theta$
- ▶ An inference (recognition) network (to model the latent variable) – parameters:  $\lambda$
- ▶ Original setting: binary hidden units

# Wake-sleep Architecture

## 2 Neural Networks:

- ▶ A generation network to model the data (the one we want to optimise) – parameters:  $\theta$
- ▶ An inference (recognition) network (to model the latent variable) – parameters:  $\lambda$
- ▶ Original setting: binary hidden units
- ▶ Training is performed in a “hard EM” fashion

# Wake-sleep Training

## Wake Phase

- ▶ Use inference network to sample hidden unit setting  $z$  from  $q(z|x, \lambda)$
- ▶ Update generation parameters  $\theta$  to maximize likelihood of data given latent state  $p(x|z, \theta)$



# Wake-sleep Training

## Wake Phase

- ▶ Use inference network to sample hidden unit setting  $z$  from  $q(z|x, \lambda)$
- ▶ Update generation parameters  $\theta$  to maximize likelihood of data given latent state  $p(x|z, \theta)$

## Sleep Phase

- ▶ Produce dream sample  $\tilde{x}$  from random hidden unit  $z$
- ▶ Update inference parameters  $\lambda$  to maximize probability of latent state  $q(z|\tilde{x}, \lambda)$

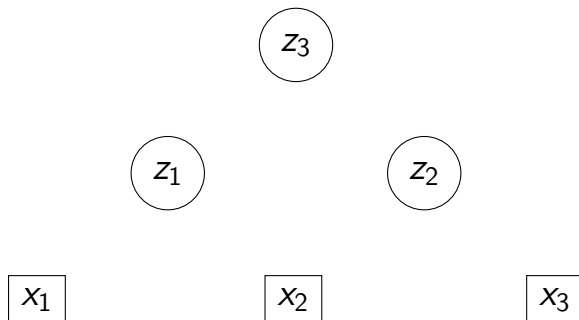
# Wake Phase Objective

Assumes latent state  $z$  to be fixed random draws from  $q(z|x, \lambda)$ .

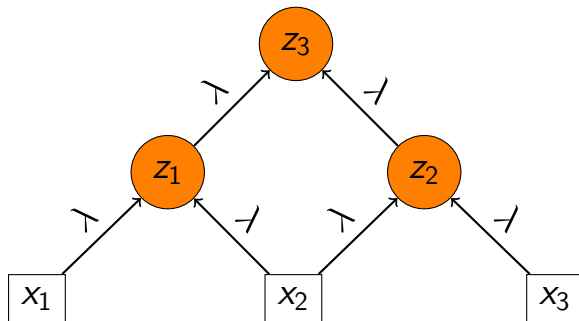
$$\max_{\theta} \log p(x|z, \theta)$$

This is simply supervised learning with imputed latent data!

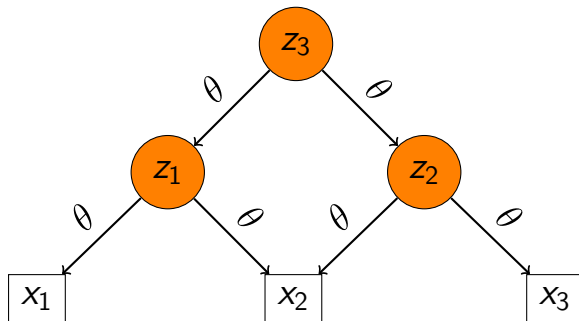
# Wake Phase Sampling



# Wake Phase Sampling



# Wake Phase Update

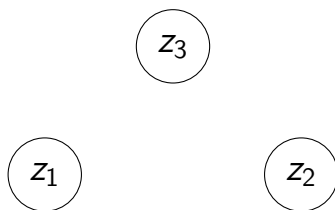


# Sleep Phase Objective

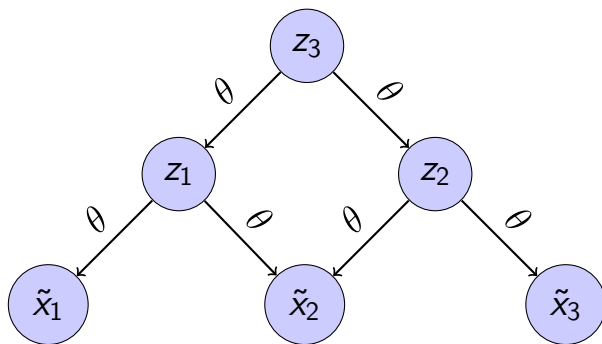
Assumes fake data  $\tilde{x}$  and latent variables  $z$  to be fixed random draw from  $p(x, z|\theta)$ .

$$\min_{\lambda} \mathbb{E}_{q(z|\tilde{x}, \lambda)} [\log p(\tilde{x}, z|\theta)] + \mathbb{H}(q(z|\tilde{x}, \lambda))$$

# Sleep Phase Sampling

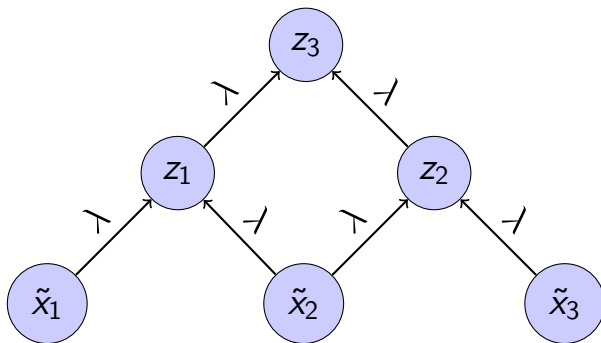


# Sleep Phase Sampling





# Sleep Phase Update



# Wake-sleep Algorithm

## Advantages

- ▶ Simple layer-wise updates
- ▶ Amortised inference: all latent variables are inferred from the same weights  $\lambda$

# Wake-sleep Algorithm

## Advantages

- ▶ Simple layer-wise updates
- ▶ Amortised inference: all latent variables are inferred from the same weights  $\lambda$

## Drawbacks

- ▶ Inference and generative networks are trained on different objectives
- ▶ Inference weights  $\lambda$  are updated on fake data  $\tilde{x}$
- ▶ Generative weights are bad initially, giving wrong signal to the updates of  $\lambda$

## Generative Models

First Attempt: Log-linear Models

Second Attempt: Wake-Sleep

This is how we do: Variational Autoencoders

# Generative Model with NN Likelihood

## Goal

Define model  $p(x, z|\theta) = p(x|z, \theta)p(z)$  where the likelihood  $p(x|z, \theta)$  is given by a neural network.  
(We fix  $p(z)$  for simplicity.)

# Generative Model with NN Likelihood

## Goal

Define model  $p(x, z|\theta) = p(x|z, \theta)p(z)$  where the likelihood  $p(x|z, \theta)$  is given by a neural network.  
(We fix  $p(z)$  for simplicity.)

## Problem

$p(x) = \int p(x|z, \theta)p(z)dz$  is hard to compute.

# Generative Model with NN Likelihood

## Goal

Define model  $p(x, z|\theta) = p(x|z, \theta)p(z)$  where the likelihood  $p(x|z, \theta)$  is given by a neural network.  
(We fix  $p(z)$  for simplicity.)

## Problem

$p(x) = \int \underbrace{p(x|z, \theta)}_{\text{highly non-linear!}} p(z) dz$  is hard to compute.

# Generative Model with NN Likelihood

Solution: VI

$$\log p(x) \geq \overbrace{\mathbb{E}_{q(z|x, \lambda)} [\log p(x, z|\theta)]}^{\text{ELBO}} + \mathbb{H}(q(z|x, \lambda))$$



# Generative Model with NN Likelihood

Solution: VI

$$\begin{aligned}\log p(x) &\geq \overbrace{\mathbb{E}_{q(z|x, \lambda)} [\log p(x, z|\theta)] + \mathbb{H}(q(z|x, \lambda))}^{\text{ELBO}} \\ &= \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta) + \log p(z)] + \mathbb{H}(q(z|x, \lambda))\end{aligned}$$

# Generative Model with NN Likelihood

## Solution: VI

$$\begin{aligned}\log p(x) &\geq \overbrace{\mathbb{E}_{q(z|x, \lambda)} [\log p(x, z|\theta)] + \mathbb{H}(q(z|x, \lambda))}^{\text{ELBO}} \\ &= \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta) + \log p(z)] + \mathbb{H}(q(z|x, \lambda)) \\ &= \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta)] - \text{KL}(q(z|x, \lambda) \parallel p(z))\end{aligned}$$

# Generative Model with NN Likelihood

## Solution: VI

$$\begin{aligned}
 \log p(x) &\geq \overbrace{\mathbb{E}_{q(z|x, \lambda)} [\log p(x, z|\theta)] + \mathbb{H}(q(z|x, \lambda))}^{\text{ELBO}} \\
 &= \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta) + \log p(z)] + \mathbb{H}(q(z|x, \lambda)) \\
 &= \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta)] - \underbrace{\text{KL}(q(z|x, \lambda) || p(z))}_{\substack{\text{assume analytical} \\ \text{(true for exponential families)}}}
 \end{aligned}$$

# Generative Model with NN Likelihood

## Solution: VI

$$\begin{aligned}
 \log p(x) &\geq \overbrace{\mathbb{E}_{q(z|x, \lambda)} [\log p(x, z|\theta)] + \mathbb{H}(q(z|x, \lambda))}^{\text{ELBO}} \\
 &= \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta) + \log p(z)] + \mathbb{H}(q(z|x, \lambda)) \\
 &= \underbrace{\mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta)]}_{\text{approximate by sampling}} - \underbrace{\text{KL}(q(z|x, \lambda) \parallel p(z))}_{\substack{\text{assume analytical} \\ \text{(true for exponential families)}}}
 \end{aligned}$$

# Generation Network Gradient

$$\frac{d}{d\theta} \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta)] - \overbrace{\text{KL}(q(z|x, \lambda) \parallel p(z))}^{\text{constant}}$$

# Generation Network Gradient

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

# Generation Network Gradient

$$\begin{aligned}
 & \frac{d}{d\theta} \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta)] - \overbrace{\text{KL}(q(z|x, \lambda) || p(z))}^{\text{constant}} \\
 &= \mathbb{E}_{q(z|x, \lambda)} \left[ \frac{d}{d\theta} \log p(x|z, \theta) \right] \\
 &\approx^{\text{MC}} \frac{1}{S} \sum_{i=1}^S \frac{d}{d\theta} \log p(x|z_i, \theta)
 \end{aligned}$$

# Generation Network Gradient

$$\begin{aligned}
 & \frac{d}{d\theta} \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta)] - \overbrace{\text{KL}(q(z|x, \lambda) || p(z))}^{\text{constant}} \\
 &= \mathbb{E}_{q(z|x, \lambda)} \left[ \frac{d}{d\theta} \log p(x|z, \theta) \right] \\
 &\approx \frac{1}{S} \sum_{i=1}^S \frac{d}{d\theta} \log p(x|z_i, \theta)
 \end{aligned}$$

Note:  $q(z|x, \lambda)$  does not depend on  $\theta$ .



# Inference Network Gradient

$$\frac{d}{d\lambda} \left[ \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta)] - \text{KL} (q(z|x, \lambda) || p(z)) \right]$$

# Inference Network Gradient

$$\begin{aligned} & \frac{d}{d\lambda} \left[ \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta)] - \text{KL} (q(z|x, \lambda) \parallel p(z)) \right] \\ &= \frac{d}{d\lambda} \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta)] - \underbrace{\frac{d}{d\lambda} \text{KL} (q(z|x, \lambda) \parallel p(z))}_{\text{analytical computation}} \end{aligned}$$

# Inference Network Gradient

$$\begin{aligned} & \frac{d}{d\lambda} \left[ \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta)] - \text{KL} (q(z|x, \lambda) \parallel p(z)) \right] \\ &= \frac{d}{d\lambda} \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta)] - \underbrace{\frac{d}{d\lambda} \text{KL} (q(z|x, \lambda) \parallel p(z))}_{\text{analytical computation}} \end{aligned}$$

The first term again requires approximation by sampling

# Inference Network Gradient

$$\begin{aligned} & \frac{d}{d\lambda} \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta)] \\ &= \frac{d}{d\lambda} \int q(z|x, \lambda) \log p(x|z, \theta) dz \end{aligned}$$

# Inference Network Gradient

$$\begin{aligned} & \frac{d}{d\lambda} \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta)] \\ &= \frac{d}{d\lambda} \int q(z|x, \lambda) \log p(x|z, \theta) dz \end{aligned}$$

MC estimator non-differentiable

# Inference Network Gradient

$$\begin{aligned} & \frac{d}{d\lambda} \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta)] \\ &= \frac{d}{d\lambda} \int q(z|x, \lambda) \log p(x|z, \theta) dz \end{aligned}$$

## MC estimator non-differentiable

- ▶ Sampling  $z$  neglects  $\frac{d}{d\lambda} q(z|x, \lambda)$

# Inference Network Gradient

$$\begin{aligned} & \frac{d}{d\lambda} \mathbb{E}_{q(z|x, \lambda)} [\log p(x|z, \theta)] \\ &= \frac{d}{d\lambda} \int q(z|x, \lambda) \log p(x|z, \theta) dz \end{aligned}$$

## MC estimator non-differentiable

- ▶ Sampling  $z$  neglects  $\frac{d}{d\lambda} q(z|x, \lambda)$
- ▶ Differentiating  $q(z|x, \lambda)$  breaks the expectation

# Inference Network Gradient

## Reparametrisation trick

Find a transformation  $h : z \mapsto \epsilon$  such that  $\epsilon$  does not depend on  $\lambda$ .

- ▶  $h(z, \lambda)$  needs to be invertible
- ▶  $h(z, \lambda)$  needs to be differentiable



# Inference Network Gradient

## Reparametrisation trick

Find a transformation  $h : z \mapsto \epsilon$  such that  $\epsilon$  does not depend on  $\lambda$ .

- ▶  $h(z, \lambda)$  needs to be invertible
- ▶  $h(z, \lambda)$  needs to be differentiable
- ▶  $h(z, \lambda) = \epsilon$
- ▶  $h^{-1}(\epsilon, \lambda) = z$

# Gaussian Transformation

## Affine property

$$Ax + b \sim \mathcal{N}(\mu + b, A\Sigma A^T) \text{ for } x \sim \mathcal{N}(\mu, \Sigma)$$

# Gaussian Transformation

## Affine property

$$Ax + b \sim \mathcal{N}(\mu + b, A\Sigma A^T) \text{ for } x \sim \mathcal{N}(\mu, \Sigma)$$

## Special case

$$Ax + b \sim \mathcal{N}(b, AA^T) \text{ for } x \sim \mathcal{N}(0, I)$$

# Gaussian Transformation

## Affine property

$$Ax + b \sim \mathcal{N}(\mu + b, A\Sigma A^T) \text{ for } x \sim \mathcal{N}(\mu, \Sigma)$$

## Special case

$$Ax + b \sim \mathcal{N}(b, AA^T) \text{ for } x \sim \mathcal{N}(0, I)$$

## Gaussian transformation

$$h(z, \lambda) = \frac{z - \mu(x, \lambda)}{\sigma(x, \lambda)} = \epsilon \sim \mathcal{N}(0, I)$$

$$h^{-1}(\epsilon, \lambda) = \mu(x, \lambda) + \sigma(x, \lambda) \odot \epsilon \quad \epsilon \sim \mathcal{N}(0, I)$$

# Inference Network Gradient

$$= \frac{d}{d\lambda} \int q(z|x, \lambda) \log p(x|z, \theta) dz$$

# Inference Network Gradient

$$\begin{aligned} &= \frac{d}{d\lambda} \int q(z|x, \lambda) \log p(x|z, \theta) dz \\ &= \frac{d}{d\lambda} \int q(\epsilon) \log \left( p(x | \overbrace{h^{-1}(\epsilon, \lambda)}^{=z}, \theta) \right) d\epsilon \end{aligned}$$

# Inference Network Gradient

$$\begin{aligned}
 &= \frac{d}{d\lambda} \int q(z|x, \lambda) \log p(x|z, \theta) dz \\
 &= \frac{d}{d\lambda} \int q(\epsilon) \log \left( p(x | \overbrace{h^{-1}(\epsilon, \lambda)}^{=z}, \theta) \right) d\epsilon \\
 &= \int q(\epsilon) \frac{d}{d\lambda} \left[ \log p(x | \overbrace{h^{-1}(\epsilon, \lambda)}^{=z}, \theta) \right] d\epsilon
 \end{aligned}$$

# Inference Network Gradient

$$= \int q(\epsilon) \frac{d}{dz} \log p(x | \overbrace{h^{-1}(\epsilon, \lambda)}^{=z}, \theta) \times \frac{d}{d\lambda} h^{-1}(\epsilon, \lambda) d\epsilon$$



# Inference Network Gradient

$$\begin{aligned}
 &= \int q(\epsilon) \frac{d}{dz} \log p(x | \overbrace{h^{-1}(\epsilon, \lambda)}^{=z}, \theta) \times \frac{d}{d\lambda} h^{-1}(\epsilon, \lambda) d\epsilon \\
 &= \mathbb{E}_{p(\epsilon)} \left[ \frac{d}{dz} \log p(x | \overbrace{h^{-1}(\epsilon, \lambda)}^{=z}, \theta) \times \frac{d}{d\lambda} h^{-1}(\epsilon, \lambda) \right]
 \end{aligned}$$

# Inference Network Gradient

$$\begin{aligned}
 &= \int q(\epsilon) \frac{d}{dz} \log p(x | \overbrace{h^{-1}(\epsilon, \lambda)}^{=z}, \theta) \times \frac{d}{d\lambda} h^{-1}(\epsilon, \lambda) d\epsilon \\
 &= \mathbb{E}_{p(\epsilon)} \left[ \frac{d}{dz} \log p(x | \overbrace{h^{-1}(\epsilon, \lambda)}^{=z}, \theta) \times \frac{d}{d\lambda} h^{-1}(\epsilon, \lambda) \right] \\
 &\stackrel{\text{MC}}{\approx} \frac{1}{S} \sum_{i=1}^S \frac{d}{dz} \log p(x | \overbrace{h^{-1}(\epsilon, \lambda)}^{=z}, \theta) \times \frac{d}{d\lambda} h^{-1}(\epsilon, \lambda)
 \end{aligned}$$

# Derivatives of Gaussian transformation

Recall:

$$h^{-1}(\epsilon, \lambda) = \mu(x, \lambda) + \sigma(x, \lambda) \odot \epsilon .$$

# Derivatives of Gaussian transformation

Recall:

$$h^{-1}(\epsilon, \lambda) = \mu(x, \lambda) + \sigma(x, \lambda) \odot \epsilon .$$

This gives us 2 gradient paths.

# Derivatives of Gaussian transformation

Recall:

$$h^{-1}(\epsilon, \lambda) = \mu(x, \lambda) + \sigma(x, \lambda) \odot \epsilon .$$

This gives us 2 gradient paths.

$$\frac{d}{d\mu(x, \lambda)} = \frac{d}{d\mu(x, \lambda)} [\mu(x, \lambda) + \sigma(x, \lambda) \odot \epsilon] = 1$$

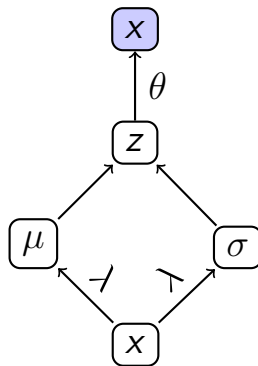
$$\frac{d}{d\sigma(x, \lambda)} = \frac{d}{d\sigma(x, \lambda)} [\mu(x, \lambda) + \sigma(x, \lambda) \odot \epsilon] = \epsilon$$

# Gaussian KL

Analytical computation of  $-\text{KL}(q(z|x, \lambda) || p(z))$ :

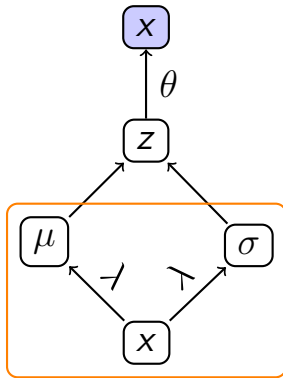
$$-\frac{1}{2} \sum_{i=1}^N (1 + \log(\sigma_i^2) - \mu_i - \sigma_i^2)$$

# Computation Graph



# Computation Graph

inference model

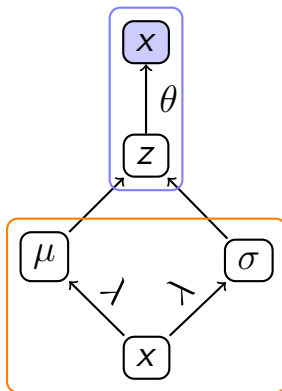




# Computation Graph

generation model

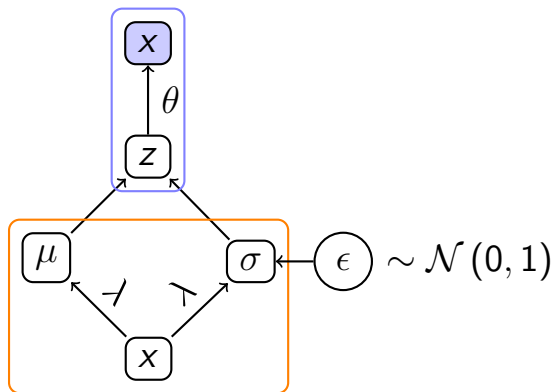
inference model



# Computation Graph

generation model

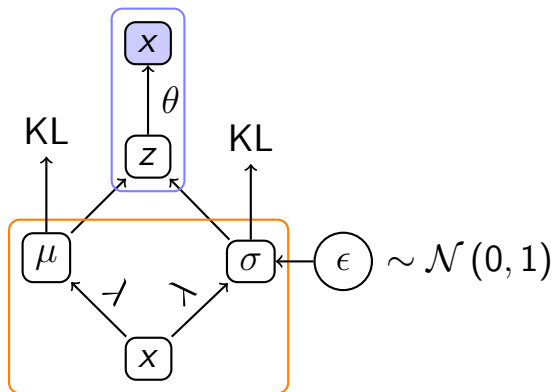
inference model



# Computation Graph

generation model

inference model



# Example

- ▶ Data: binary mnist
- ▶ Likelihood: product of Bernoullis
  - ▶ Let  $\phi = \sigma(\text{NN}(z))$
  - ▶  $\prod_{i=1}^N p(x_i|\phi) = \prod_{i=1}^N \phi^{x_i} \times (1 - \phi)^{1-x_i}$
- ▶ Prior over  $z$ :  $\mathcal{N}(0, 1)$
- ▶  $q(z|x, \lambda) = \mathcal{N}(\mu(x, \lambda), \sigma(x, \lambda)^2)$
- ▶  $\mu(x, \lambda) = \text{NN}_\mu(x; \lambda)$
- ▶  $\sigma(x, \lambda) = \text{NN}_\sigma(x; \lambda)$

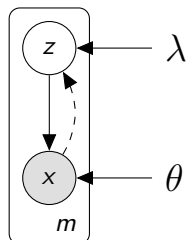
# Example

- ▶ Data: binary mnist
- ▶ Likelihood: product of Bernoullis
  - ▶ Let  $\phi = \sigma(\text{NN}(z))$
  - ▶  $\prod_{i=1}^N p(x_i|\phi) = \prod_{i=1}^N \phi^{x_i} \times (1 - \phi)^{1-x_i}$
- ▶ Prior over  $z$ :  $\mathcal{N}(0, 1)$
- ▶  $q(z|x, \lambda) = \mathcal{N}(\mu(x, \lambda), \sigma(x, \lambda)^2)$
- ▶  $\mu(x, \lambda) = \text{NN}_\mu(x; \lambda)$
- ▶  $\sigma(x, \lambda) = \text{NN}_\sigma(x; \lambda)$

## Mean Field assumption

Variational approximation factorises over latent dimensions.

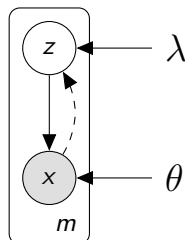
# Graphical Model



- approximate posterior

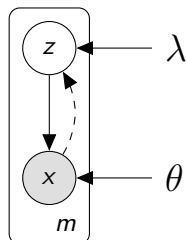
$$q(z|x, \lambda) = \mathcal{N}(\mu(x, \lambda), \sigma(x, \lambda)^2)$$

# Graphical Model



- ▶ approximate posterior  
 $q(z|x, \lambda) = \mathcal{N}(\mu(x, \lambda), \sigma(x, \lambda)^2)$
- ▶ where
  - ▶  $\mu(x, \lambda) = \text{NN}_{\mu}(x; \lambda)$   
 e.g.  $\mu(x, \lambda) = W^{(u)}x + b^{(u)}$

# Graphical Model



- ▶ approximate posterior

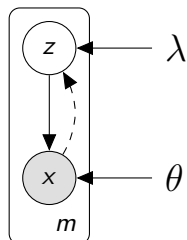
$$q(z|x, \lambda) = \mathcal{N}(\mu(x, \lambda), \sigma(x, \lambda)^2)$$

- ▶ where

- ▶  $\mu(x, \lambda) = \text{NN}_{\mu}(x; \lambda)$   
e.g.  $\mu(x, \lambda) = W^{(u)}x + b^{(u)}$
- ▶  $\sigma(x, \lambda) = \exp(\text{NN}_{\sigma}(x; \lambda))$   
e.g.  $\sigma(x, \lambda) = \log(1 + \exp(W^{(v)}x + b^{(v)}))$



# Graphical Model



- ▶ approximate posterior
 
$$q(z|x, \lambda) = \mathcal{N}(\mu(x, \lambda), \sigma(x, \lambda)^2)$$
- ▶ where
  - ▶  $\mu(x, \lambda) = \text{NN}_{\mu}(x; \lambda)$   
e.g.  $\mu(x, \lambda) = W^{(u)}x + b^{(u)}$
  - ▶  $\sigma(x, \lambda) = \exp(\text{NN}_{\sigma}(x; \lambda))$   
e.g.  $\sigma(x, \lambda) = \log(1 + \exp(W^{(v)}x + b^{(v)}))$
  - ▶  $\lambda = (W^{(u)}, W^{(v)}, b^{(u)}, b^{(v)})$

# Variational Autoencoder

## Advantages

- ▶ Backprop training
- ▶ Easy to implement
- ▶ Posterior inference possible
- ▶ One objective for both NNs

# Variational Autoencoder

## Advantages

- ▶ Backprop training
- ▶ Easy to implement
- ▶ Posterior inference possible
- ▶ One objective for both NNs

## Drawbacks

- ▶ Discrete latent variables are difficult
- ▶ Optimisation may be difficult with several latent variables

# Summary

- ▶ When  $|\mathcal{X}|$  and  $|\mathcal{Z}|$  are not too large, we can do EM with features
- ▶ Otherwise use VI with simple approximation
- ▶ Wake-Sleep: train inference and generation networks with separate objectives
- ▶ VAE: train both networks with same objective
- ▶ Reparametrisation
  - ▶ Transform parameter-free variable  $\epsilon$  into latent value  $z$
  - ▶ Update parameters with stochastic gradient estimates

# Literature I

Taylor Berg-Kirkpatrick, Alexandre Bouchard-Côté, John DeNero, and Dan Klein. Painless unsupervised learning with features. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 582–590, 2010. URL <http://www.aclweb.org/anthology/N10-1083>.

# Literature II

- G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal.  
The wake-sleep algorithm for unsupervised neural  
networks. *Science*, 268:1158–1161, 1995. URL  
[http://www.gatsby.ucl.ac.uk/~dayan/  
papers/hdfn95.pdf](http://www.gatsby.ucl.ac.uk/~dayan/papers/hdfn95.pdf).
- Diederik P. Kingma and Max Welling.  
Auto-Encoding Variational Bayes. 2013. URL  
<http://arxiv.org/abs/1312.6114>.

# Literature III

Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M. Blei. Automatic differentiation variational inference. *Journal of Machine Learning Research*, 18(14):1–45, 2017.

URL

<http://jmlr.org/papers/v18/16-107.html>.

Danilo J. Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*,

# Literature IV

pages 1278–1286, 2014. URL

<http://jmlr.org/proceedings/papers/v32/rezende14.pdf>.

Michalis Titsias and Miguel Lázaro-Gredilla. Doubly stochastic variational bayes for non-conjugate inference. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1971–1979, 2014. URL

<http://jmlr.org/proceedings/papers/v32/titsias14.pdf>.