
# Lecture 6: Variational Auto-Encoders

Advanced Topics in Machine Learning

Dr. Tom Rainforth

January 31st, 2020

rainforth@stats.ox.ac.uk

# Lecture 6 Outline

In this lecture we will show how we can combine ideas from variational inference with deep learning to produce methods for learning powerful **deep generative models**

Particular topics:

- Latent variable models
- Learning models
- Deep latent variable models
- Model learning with the ELBO
- Variational Auto–Encoders

# Latent Variable Models

# What is a Latent Variable?

- The term latent stems from the Latin to lie hidden
- A **latent variable** is a thus any variable we don't observe (think unobserved nodes in a DAG)
- In other words, any parameter in a Bayesian model is a technically a latent variable
- However, the term is slightly overloaded: people typically use it to specifically refer to a parameter than has a direct relationship with a **single** datapoint
    - This is in contrast to a **global parameter** which effects all the datapoints

# What is a Latent Variable Model?

- A **latent variable model** (LVM) is a generative model where each datapoint has a corresponding latent variable
- For data $\mathcal{D} = \{\boldsymbol{x}_n\}_{n=1}^N$, the generative model for any latent variable model takes the form

$$p(\theta, \boldsymbol{z}_{1:N}, \mathcal{D}) = p(\theta)p(\boldsymbol{z}_{1:N}|\theta)\prod_{n=1}^{N} p(\boldsymbol{x}_n|\boldsymbol{z}_n, \theta) \tag{1}$$

  where $\theta$ are our global variables (which are sometimes fixed) and each $\boldsymbol{z}_n$ is a latent variable associated with datapoint $\boldsymbol{x}_n$
- The underlying assumption behind LVMs is that each datapoint can be explained by its latent variable in conjunction with the global parameters
- Often, the latent variable is much lower dimensional or simpler than the datapoint. It may also have interpretable meaning.

# Example: Gaussian Mixture Model

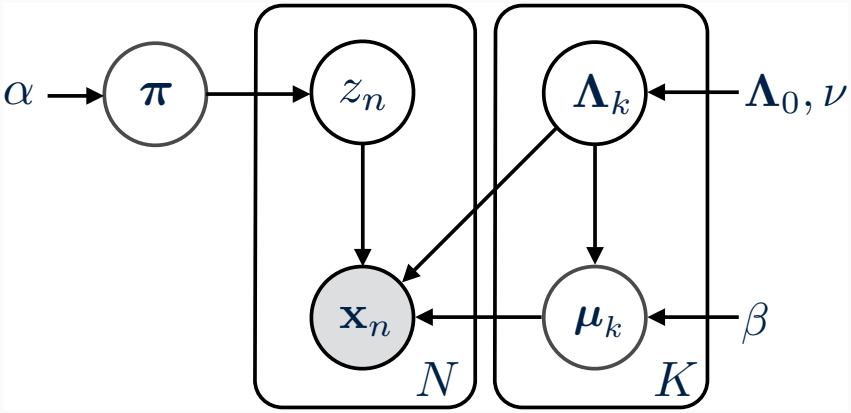
Prior:

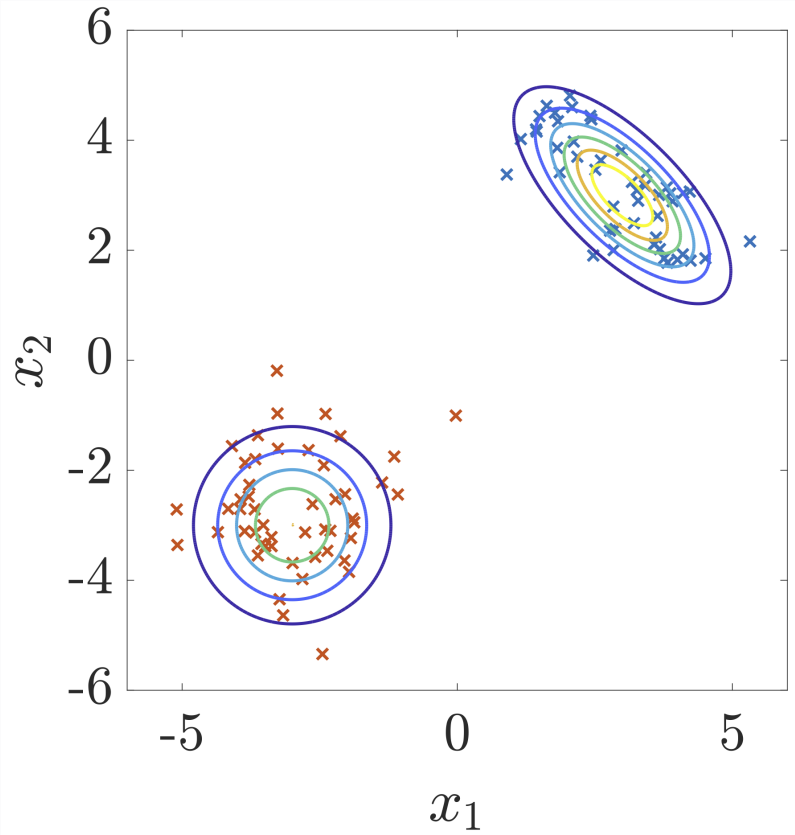
$$\begin{aligned}
\pi &\sim \mathrm{Dirichlet}(\alpha) \\
\Lambda_k &\sim \mathrm{Wishart}(\Lambda_0, \nu) \quad \forall k \in \{1, \ldots, K\} \\
\mu_k \big| \Lambda_k &\sim \mathcal{N}\big(\mathbf{0}, (\beta\Lambda_k)^{-1}\big) \quad \forall k \in \{1, \ldots, K\} \\
\boldsymbol{z}_n \big| \pi &\sim \mathrm{Categorical}(\pi) \quad \forall n \in \{1, \ldots, N\}
\end{aligned}$$

Likelihood:

$$\begin{aligned}
p(\mathcal{D}|\theta, \boldsymbol{z}_{1:N}) = p(\pi; \alpha) &\left( \prod_{n=1}^{N} p(\boldsymbol{z}_n|\pi) p(\boldsymbol{x}_n|\mu_{\boldsymbol{z}_n}, \Lambda_{\boldsymbol{z}_n}) \right) \\
&\times \left( \prod_{k=1}^{K} p(\Lambda_k; \Lambda_0, \nu) p(\mu_k|\Lambda_k; \beta) \right)
\end{aligned}$$

# Example: Gaussian Mixture Model




Global parameters:

$$\theta = \{\pi, \mu_{1:K}, \Lambda_{1:K}\}$$

Latent variables $z_{1:N}$

# Example: Latent Dirichlet Allocation for Topic Modeling

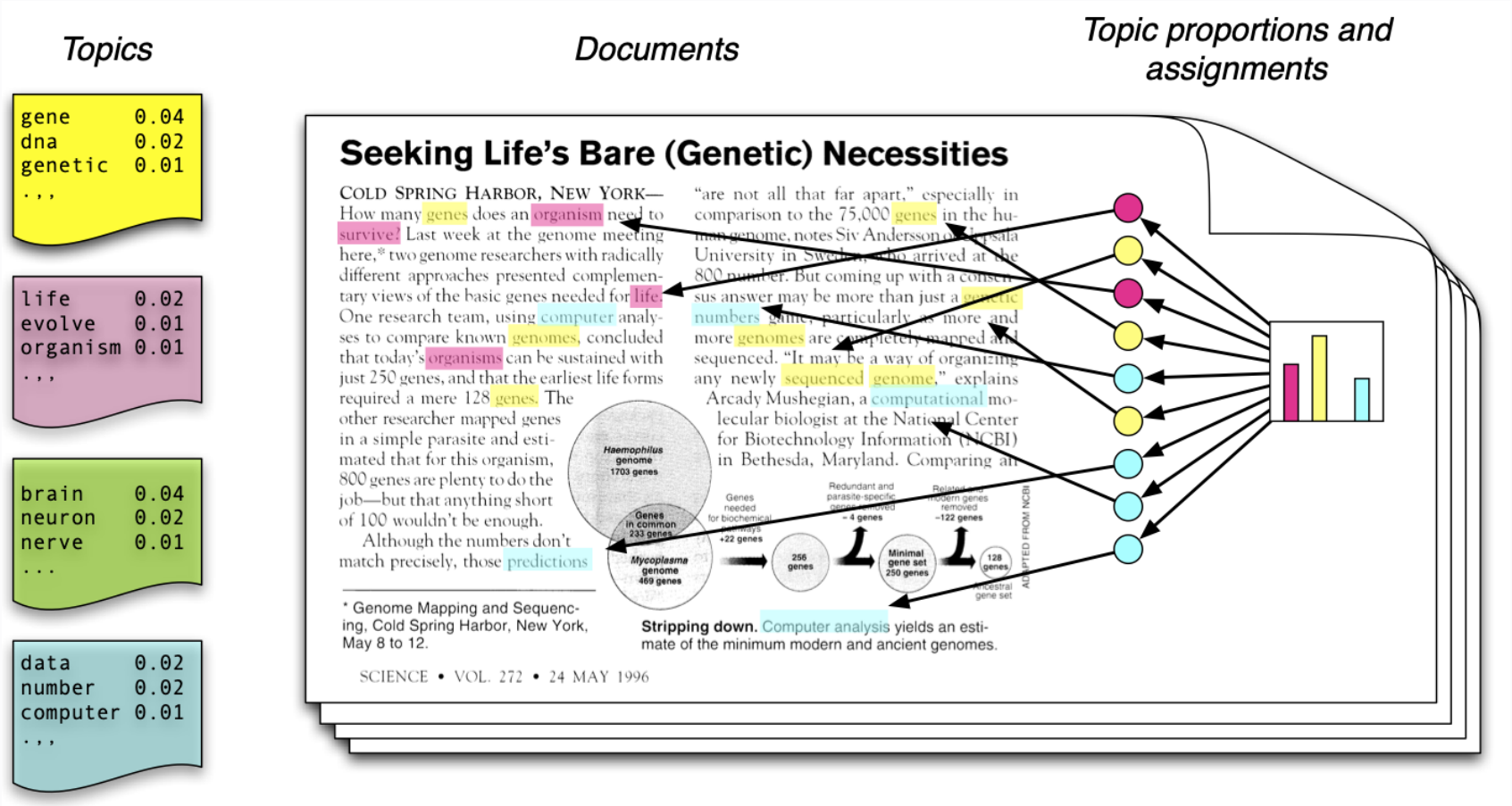
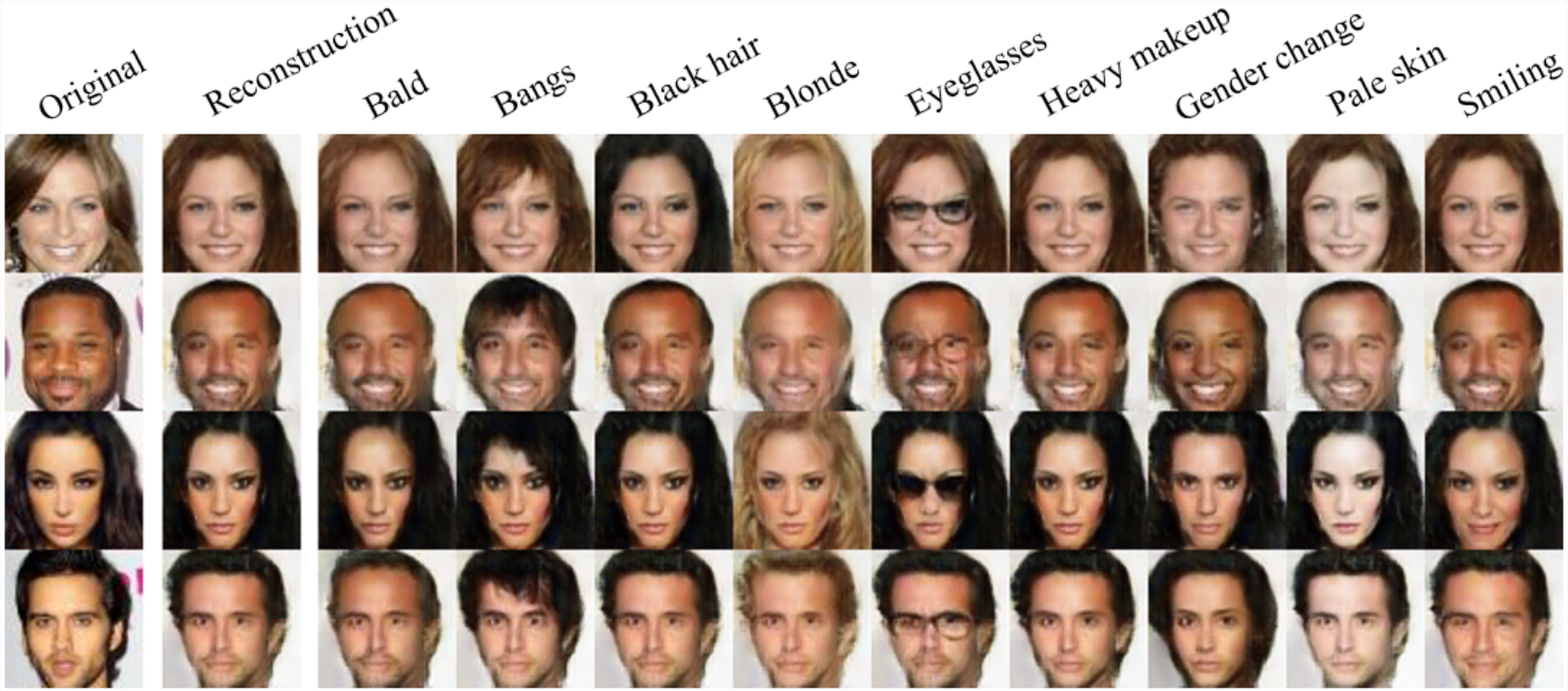



Image Credit: David Blei

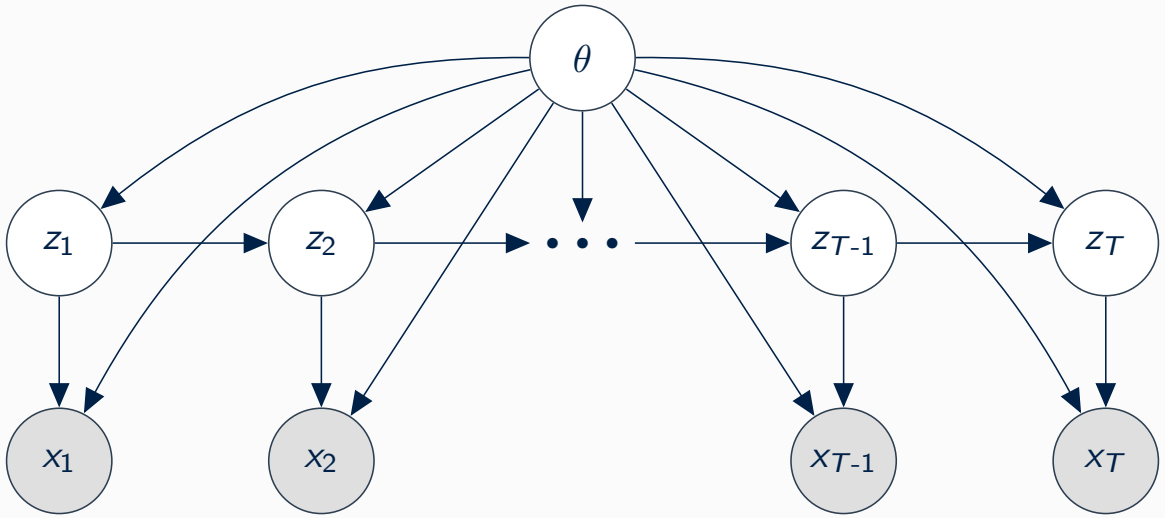
# Example: Images of Faces




[1] G Perarnau et al. "Invertible conditional gans for image editing". In: *arXiv preprint arXiv:1611.06355* (2016).

# Example: Hidden Markov Models

Not all LVM models have conditionally independent latents given $\theta$

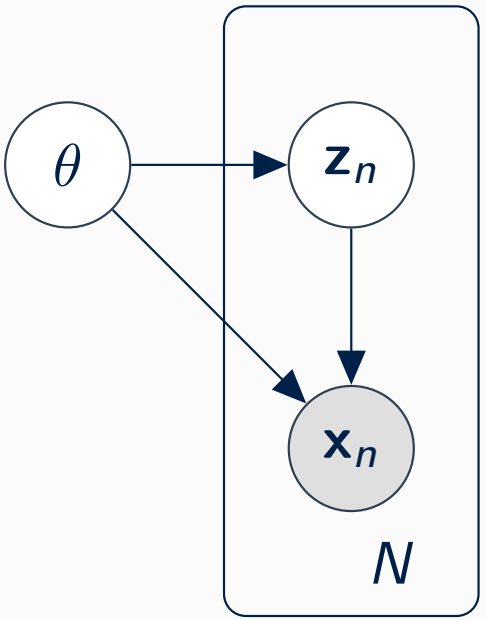



- Consider tracking a lion's movement.
- The activity they are performing is a latent state, e.g. sleeping, hunting
- Consecutive latent states are not independent: they stay in states for a while, eating tends to follow hunting, etc

# Factorized Latent Variable Models

- If we assume that our data is i.i.d. given $\theta$, this implies that all the latent variables are also conditionally independent given $\theta$
- This is a reasonable assumption whenever their is no natural ordering to the data
- The resulting modal is known as a **factorized** LVM and has joint distribution

$$p(\theta, \boldsymbol{z}_{1:N}, \mathcal{D}) = p(\theta) \prod_{n=1}^{N} p(\boldsymbol{z}_n|\theta) p(\boldsymbol{x}_n|\boldsymbol{z}_n, \theta)$$

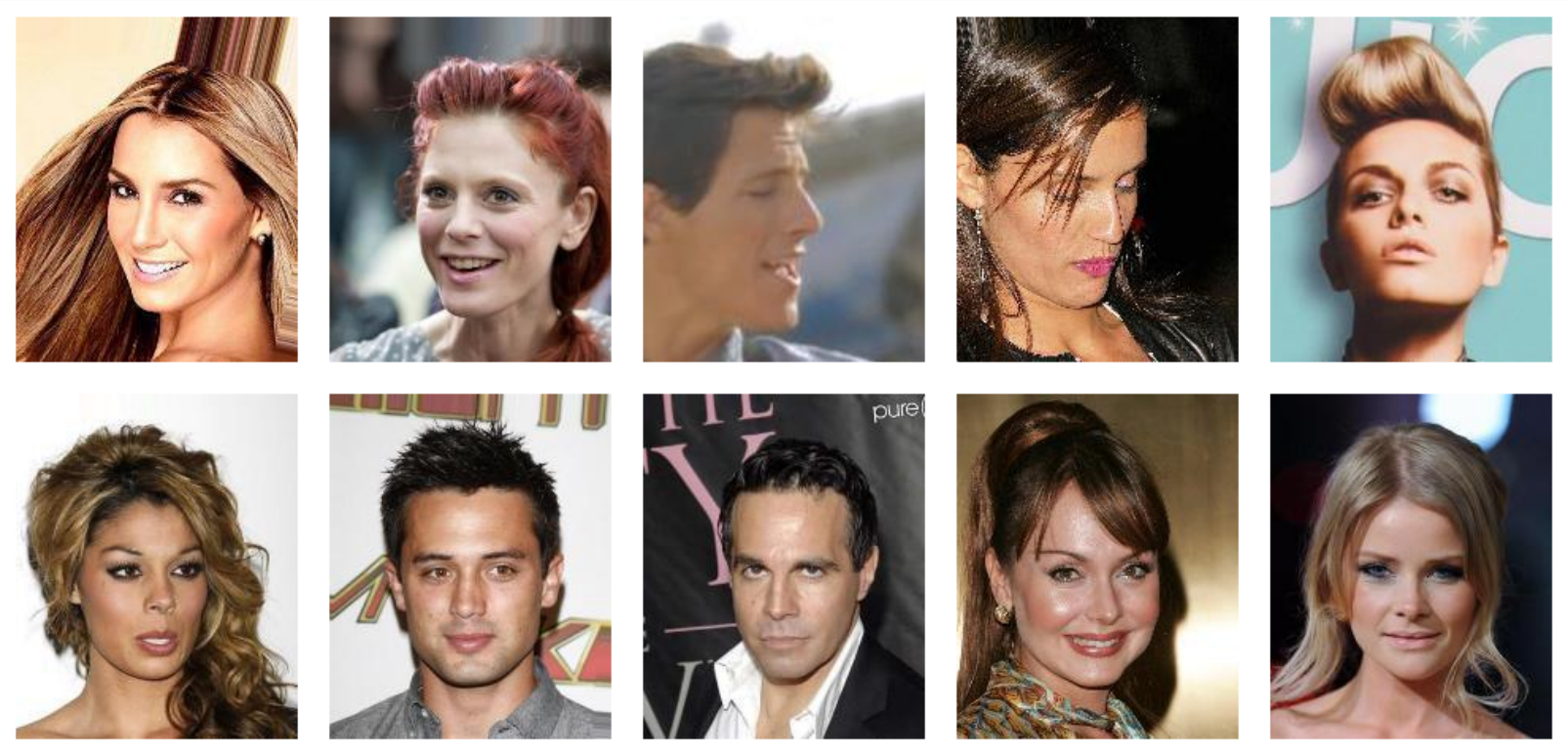



- For the rest of the lecture, our focus will be on these factorized LVMs

# Learning Models

# What If We Can't Manually Construct a Generative Model?

How could we manually construct a LVM for faces?



This would be almost impossible even with a huge about of prior expertise and time to carefully construct the model

[1] Perarnau et al., "Invertible conditional gans for image editing".

# What If We Can't Manually Construct a Generative Model?

- If we have little data and can't manually write a good generative model we give up
- But if we have a lot of data, we can **learn** a generative model instead
- We can do this my defining a flexible class of models and then optimizing to find the best model within this class

# Recap: What Makes a Good Model?

The marginal likelihood, or evidence, $p(\mathcal{D}|m)$ gives a means of measuring how good a model is.

The model with highest evidence is the one that is powerful enough to explain that data but not anything more complicated.

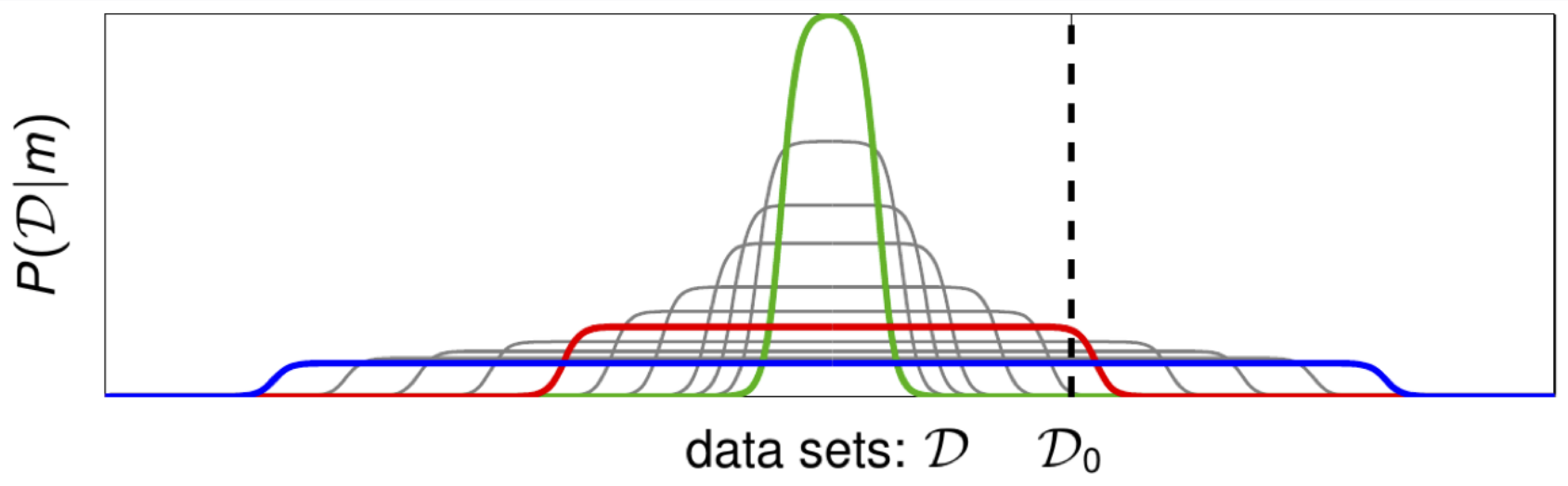
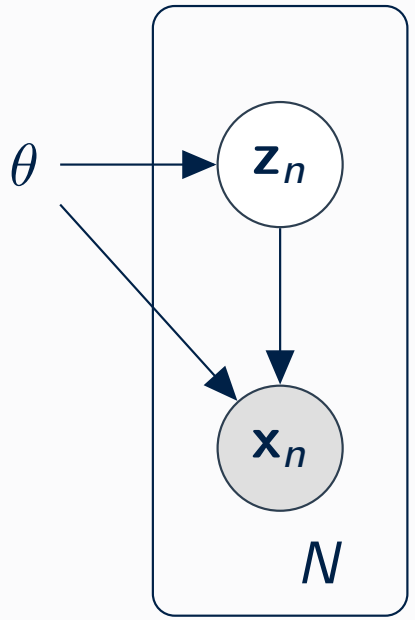



Image credit: Maneesh Sahani

# Type–II Maximum Likelihood

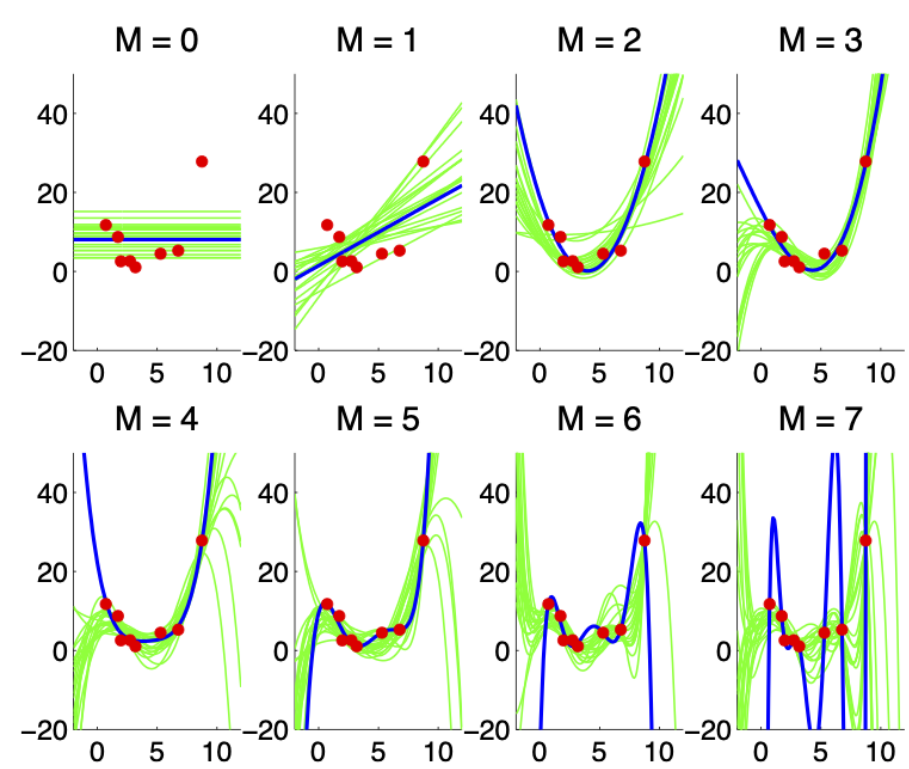
- If we have a parametrized model, we can try to directly optimize the marginal likelihood to learn a model
- This is know as **type–II maximum likelihood**
- For our factorized LVM models, we will presume $\theta$ is fixed for any particular model
- As such, we can perform model learning by optimizing for $\theta$:

$$\begin{aligned}\theta^* &= \arg\max_{\theta \in \vartheta} p_\theta(\mathcal{D}) \\ &= \arg\max_{\theta \in \vartheta} \mathbb{E}_{p_\theta(\boldsymbol{z}_{1:N})} \left[ p_\theta(\mathcal{D} | \boldsymbol{z}_{1:N}) \right]\end{aligned}$$



# Example: Choosing the Degree of Polynomial

We have already come across an example of type–II maximum likelihood: choosing the degree of the polynomial in Bayesian polynomial regression (note this is not a factorized LVM though)



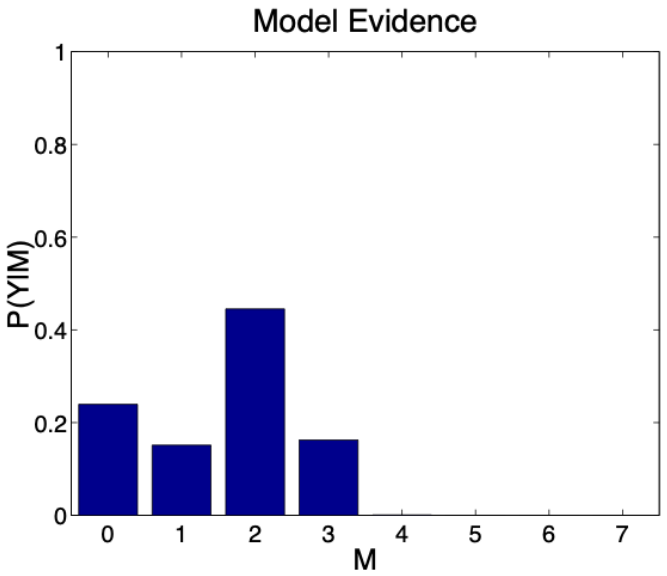




Image credit: Maneesh Sahani

# Type–II Maximum Likelihood for Factorized LVMs

For factorized LVMs, we can simplify the form of our optimization by working with the log evidence and exploiting the independences

$$
\begin{aligned}
\theta^* &= \arg\max_{\theta \in \vartheta} \; p_\theta(\mathcal{D}) \\
&= \arg\max_{\theta \in \vartheta} \; \log p_\theta(\mathcal{D}) \\
&= \arg\max_{\theta \in \vartheta} \; \log \left( \prod_{n=1}^{N} p_\theta(\boldsymbol{x}_n) \right) \\
&= \arg\max_{\theta \in \vartheta} \; \sum_{n=1}^{N} \log p_\theta(\boldsymbol{x}_n) \\
&= \arg\max_{\theta \in \vartheta} \; \sum_{n=1}^{N} \log \left( \mathbb{E}_{p_\theta(\boldsymbol{z}_n)} \left[ p_\theta(\boldsymbol{x}_n | \boldsymbol{z}_n) \right] \right)
\end{aligned}
$$

# Alternative View: Divergence to The True Distribution

As the data is i.i.d. in a factorized LVM, we can think of the generative model as being defined using only a single arbitrary datapoint, i.e. our model is

$$p_\theta(\boldsymbol{z})p_\theta(\boldsymbol{x}|\boldsymbol{z}) \tag{2}$$

We can now think of type-II maximum likelihood as minimizing a Monte Carlo estimate of the KL divergence from the true data generating distribution $p_{\text{true}}(\boldsymbol{x})$ to the marginal of our model $p_\theta(\boldsymbol{x})$

# Alternative View: Divergence to The True Distribution

$$
\begin{aligned}
\theta^* &= \arg\min_{\theta \in \vartheta} \; \mathrm{KL}(p_{\text{true}}(\boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{x})) \\
&= \arg\min_{\theta \in \vartheta} \; \mathbb{E}_{p_{\text{true}}(\boldsymbol{x})}\left[\log p_{\text{true}}(\boldsymbol{x})\right] - \mathbb{E}_{p_{\text{true}}(\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x})\right] \\
&= \arg\max_{\theta \in \vartheta} \mathbb{E}_{p_{\text{true}}(\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x})\right] \\
&\approx \arg\max_{\theta \in \vartheta} \frac{1}{N} \sum_{n=1}^{N} \log p_\theta(\boldsymbol{x}_n) \quad \text{where} \quad \boldsymbol{x}_n \sim p_{\text{true}}(\boldsymbol{x}) \\
&= \arg\max_{\theta \in \vartheta} \sum_{n=1}^{N} \log p_\theta(\boldsymbol{x}_n) \quad \text{where} \quad \boldsymbol{x}_n \sim p_{\text{true}}(\boldsymbol{x}) \\
&= \arg\max_{\theta \in \vartheta} p_\theta(\mathcal{D})
\end{aligned}
$$

# Deep Latent Variable Models

# Deep Latent Variable Models

- A **deep latent variable model** is simply a LVM where the likelihood $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is based on a flexible deep neural network
- Typically $p_\theta(\boldsymbol{z})$ is a fixed distribution, e.g. $p_\theta(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; 0, I)$
  - We will drop the $\theta$ subscript
- Most deep LVMs operate in an **unsupervised** manner: we have no labels associated with our datapoints $\boldsymbol{x}_n$
- Note that $\boldsymbol{z}_n$ and $\boldsymbol{x}_n$ do not themselves need to be factorized: the factorization is over different datapoints, not dimensions

# Motivation

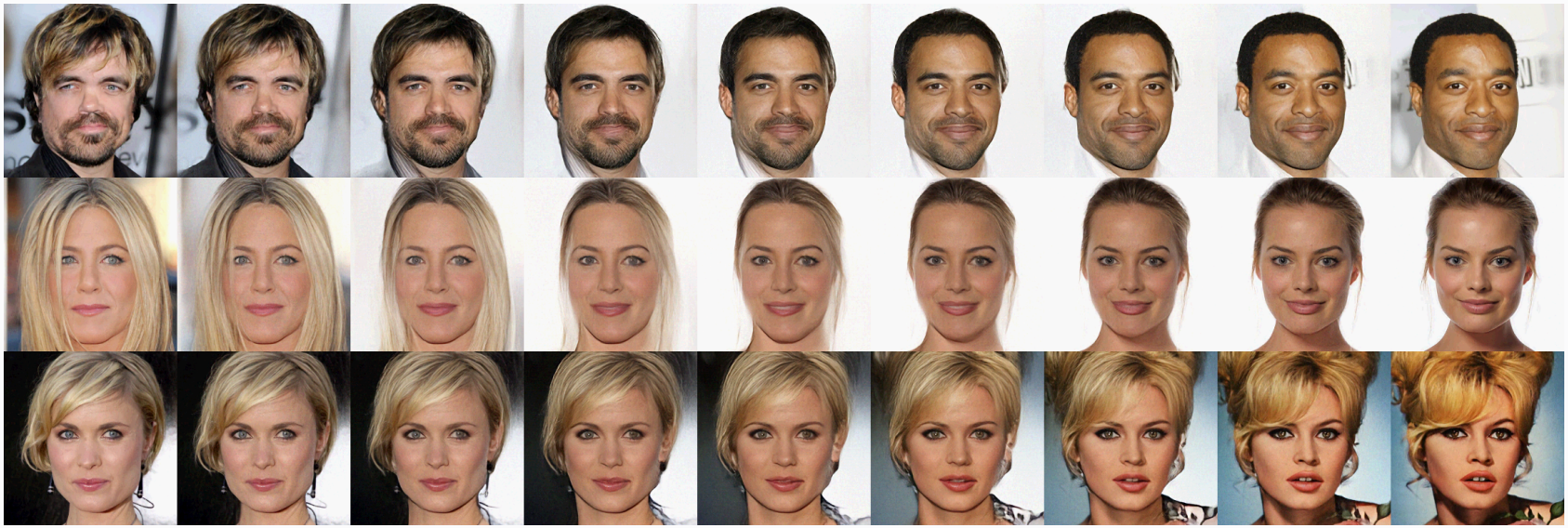
- Learn rich and powerful generative models for high–dimensional data for which handcrafted models are inappropriate
- Such data is ubiquitous, but applications in computer vision (i.e. image based data) is particularly prominent

# Example Applications: Generating New Data

These are not real faces: they are samples from a learned model!

[2] D P Kingma and P Dhariwal. "Glow: Generative flow with invertible 1x1 convolutions". In: *NeurIPS*. 2018.

# Example Applications: Manipulating Images



[2] Kingma and Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions".

# Example Applications: Deep Fakes

Which image is real?

Images credit: Stefano Ermon and Aditya Grover

# Example Applications: Deep Fakes (2)

Neither!



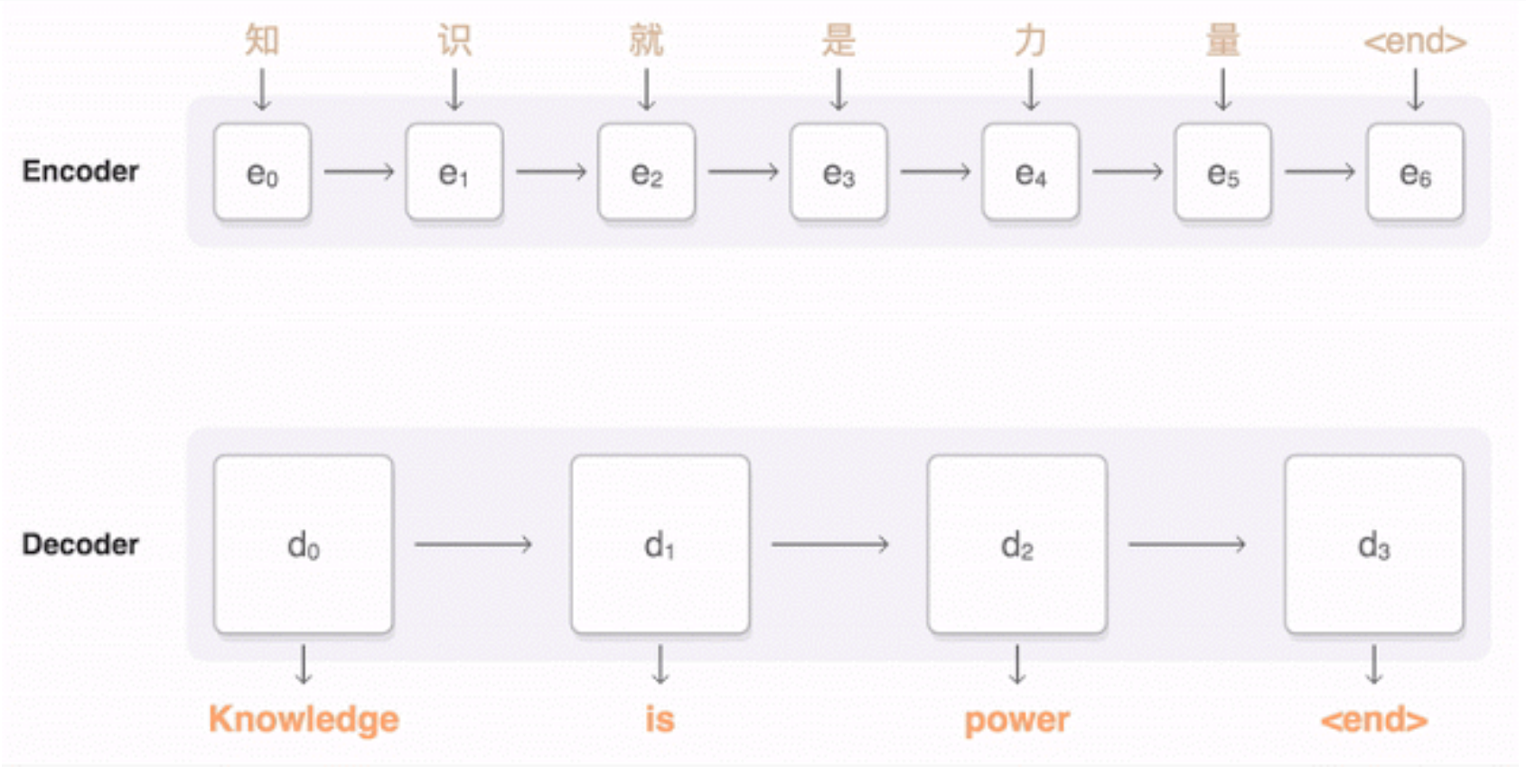



No glasses!

No smile!

Images credit: Stefano Ermon and Aditya Grover

# Example Applications: Machine Translation




Images credit: Google AI research blog

# Example Applications: Text Generation

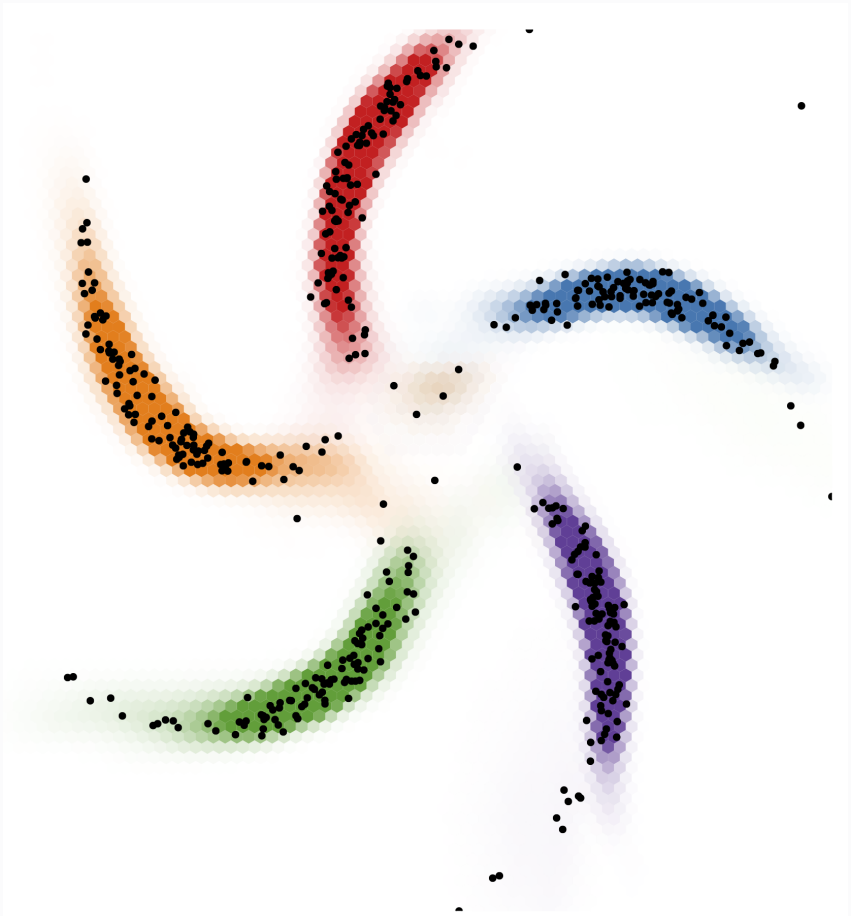
https://talktotransformer.com

# Mathematical Example: Spirals Data

Fixed parameters: $m_{1:K}$, $S_{1:K}$
Generative model for single datapoint:

$$z_1 \sim \textsc{Categorical}(1, 2, 3, 4, 5)$$
$$z_2 \sim \mathcal{N}(z_2; m_{z_1}, S_{z_1})$$
$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x}; \mu_\theta(z_1, z_2), \Sigma_\theta(z_1, z_2))$$

where $\mu_\theta$ and $\Sigma_\theta$ are deep neural networks taking in $(z_1, z_2)$ and returning a mean and covariance for an individual datapoint



# Model Learning as Gradient–Based Optimization

- Recap: we want to learn the model by maximizing

$$\log p_\theta(\mathcal{D}) = \sum_{n=1}^{N} \log \left( \mathbb{E}_{p(\boldsymbol{z}_n)} [p_\theta(\boldsymbol{x}_n | \boldsymbol{z}_n)] \right)$$

- Given we are using neural networks, we obviously want to do this with gradient–based methods
- We thus need the gradient:

$$\nabla_\theta \log p_\theta(\mathcal{D}) = \sum_{n=1}^{N} \nabla_\theta \log \left( \mathbb{E}_{p(\boldsymbol{z}_n)} [p_\theta(\boldsymbol{x}_n | \boldsymbol{z}_n)] \right)$$

# Recap from Previous Course: Stochastic Gradient Ascent

We can maximize a function $f(\theta)$ using approximate gradients $\hat{\nabla}_\theta f(\theta)$ to carry out **stochastic gradient ascent** (SGA) updates:

$$\theta_{t+1} = \theta_t + \rho_t \hat{\nabla}_\theta f(\theta) \tag{3}$$

provided that $\mathbb{E}[\hat{\nabla}_\theta f(\theta)] = \nabla f(\theta)$ (i.e. our gradient estimate is unbiased) and our **step sizes** $\rho_t$ diminish according to the Robbins-Monro conditions.

In other words:

- We don't need the gradient, we only need an unbiased estimate of it
- Our step sizes need to get smaller over time

# Problem 1: Lack of an Unbiased Estimator

- There are unfortunately a lot of problems if we want to apply SGA to $\nabla_\theta \log p_\theta(\mathcal{D})$
- Perhaps to most significant is that we cannot generate the required unbiased gradient estimates
- By Jensen's inequality $\mathbb{E}[\log \hat{Z}] = \log \mathbb{E}[\hat{Z}]$ if and only if the variance of $\hat{Z}$ is zero
- Thus if we take a Monte Carlo estimate to construct our gradients

$$\nabla_\theta \log p_\theta(\mathcal{D}) = \sum_{n=1}^{N} \nabla_\theta \log \left( \mathbb{E}_{p(\boldsymbol{z}_n)}[p_\theta(\boldsymbol{x}_n|\boldsymbol{z}_n)] \right)$$

$$\approx \sum_{n=1}^{N} \nabla_\theta \log \left( \frac{1}{M} \sum_{m=1}^{M} p_\theta(\boldsymbol{x}_n|\hat{\boldsymbol{z}}_{m,n}) \right) \quad \text{where} \quad \hat{\boldsymbol{z}}_{m,n} \overset{i.i.d.}{\sim} p(\boldsymbol{z}_n)$$

we get biased gradients that do not converge to what we want

# Problem 2: High Variance

- A second major issue is that actually providing reliable estimates for $p_\theta(\boldsymbol{x})$ or its gradients is insurmountably challenging
- The Monte Carlo estimated we considered on the last slide

$$\mathbb{E}_{p(\boldsymbol{z}_n)}[p_\theta(\boldsymbol{x}_n|\boldsymbol{z}_n)] \approx \frac{1}{M}\sum_{m=1}^{M} p_\theta(\boldsymbol{x}_n|\hat{\boldsymbol{z}}_{m,n})$$

  is effectively an importance sampling estimate that uses the prior as a proposal
- We know from the last lecture that this will not work well when $\boldsymbol{z}$ is high–dimensional which will usually be the case
- We could try a different proposal, but this will still fall foul of the curse of dimensionality

# Problem 3: Large Dataset

- A third problem is that we will typically have a large number of datapoints and it so it is not practical to update $\theta$ using all of them at once
- We can get around this using a **mini-batch** of datapoints
- This involves only using a subset of the datapoint to make each update:

$$\hat{\nabla}_\theta \log p_\theta(\mathcal{D}) = \frac{N}{\|B\|} \sum_{n \in B} \nabla_\theta \log p_\theta(\boldsymbol{x}_n) \tag{4}$$

  where $B \subset \{1, \ldots, N\}$ is our mini-batch
- As as $\mathbb{E}[\hat{\nabla}_\theta \log p_\theta(\mathcal{D})] = \nabla_\theta \log p_\theta(\mathcal{D})$, this does not introduce bias and so is fine for SGA
- But we need to be careful that how we combat the first two problems is compatible with doing our updates this way

# Model Learning with the ELBO

# Recap: The ELBO

Let's revisit the ELBO from the last lecture with a slight change in notation to match our current discussions, using the i.i.d. nature of our data to define it for a single datapoint:

$$\begin{aligned}\mathcal{L}(\theta, \phi, \boldsymbol{x}) &= \mathbb{E}_{q_\phi(\boldsymbol{z})}\left[\log \frac{p_\theta(\boldsymbol{z}, \boldsymbol{x})}{q_\phi(\boldsymbol{z})}\right] \\ &= \log p_\theta(\boldsymbol{x}) - \mathrm{KL}(q_\phi(\boldsymbol{z}) \,\|\, p_\theta(\boldsymbol{z}|\boldsymbol{x})) \\ &\leq \log p_\theta(\boldsymbol{x})\end{aligned}$$

This is now a function of both our model parameters $\theta$ and our variational parameters $\phi$

# The ELBO as a Proxy for the log Evidence

Given we could not optimize $\log p_\theta(\mathcal{D})$ directly, could we work with the lower bound

$$\mathcal{L}(\theta, \phi, \mathcal{D}) := \sum_{n=1}^{N} \mathcal{L}(\theta, \phi, \boldsymbol{x}_n) \leq \log p_\theta(\mathcal{D})$$

instead?

Let's for now presume that $\phi$ is fixed. We could try the following optimization:

$$\theta^* = \arg\max_{\theta \in \vartheta} \sum_{n=1}^{N} \mathcal{L}(\theta, \phi, \boldsymbol{x}_n) \tag{5}$$

$$= \arg\max_{\theta \in \vartheta} \sum_{n=1}^{N} \mathbb{E}_{q_\phi(\boldsymbol{z}_n)} \left[ \log \frac{p_\theta(\boldsymbol{z}_n, \boldsymbol{x}_n)}{q_\phi(\boldsymbol{z}_n)} \right] \tag{6}$$

# The ELBO as a Proxy for the log Evidence (2)

This objective has a critical difference to the log evidence: we can construct unbiased gradient updates for it and run SGA

Namely we have

$$\begin{aligned}\nabla_\theta \mathcal{L}(\theta, \phi, \mathcal{D}) &= \sum_{n=1}^{N} \nabla_\theta \, \mathbb{E}_{q_\phi(\boldsymbol{z}_n)} \left[ \log \frac{p_\theta(\boldsymbol{z}_n, \boldsymbol{x}_n)}{q_\phi(\boldsymbol{z}_n)} \right] \\ &= \sum_{n=1}^{N} \mathbb{E}_{q_\phi(\boldsymbol{z}_n)} \left[ \nabla_\theta \log \frac{p_\theta(\boldsymbol{z}_n, \boldsymbol{x}_n)}{q_\phi(\boldsymbol{z}_n)} \right] \\ &\approx \hat{\nabla}_\theta \mathcal{L}(\theta, \phi, \mathcal{D}) := \frac{N}{\|B\|} \sum_{n \in B} \nabla_\theta \log \frac{p_\theta(\hat{\boldsymbol{z}}_n, \boldsymbol{x}_n)}{q_\phi(\hat{\boldsymbol{z}}_n)}\end{aligned}$$

where each $\hat{\boldsymbol{z}}_n \sim q_\phi(\boldsymbol{z}_n)$ and $\mathbb{E}[\hat{\nabla}_\theta \mathcal{L}(\theta, \phi, \mathcal{D})] = \nabla_\theta \mathcal{L}(\theta, \phi, \mathcal{D})$ such that it is a valid SGA gradient!

# Will This Give Us Anything Sensible Though?

Will this give us anything sensible though? Let's break it down using the other form of the ELBO

$$\nabla_\theta \mathcal{L}(\theta, \phi, \boldsymbol{x}) = \nabla_\theta \log p_\theta(\boldsymbol{x}) - \nabla_\theta \, \mathrm{KL}(q_\phi(\boldsymbol{z}) \,\|\, p_\theta(\boldsymbol{z}|\boldsymbol{x}))$$

We thus see we have two competing terms, optimizing the ELBO whats to:

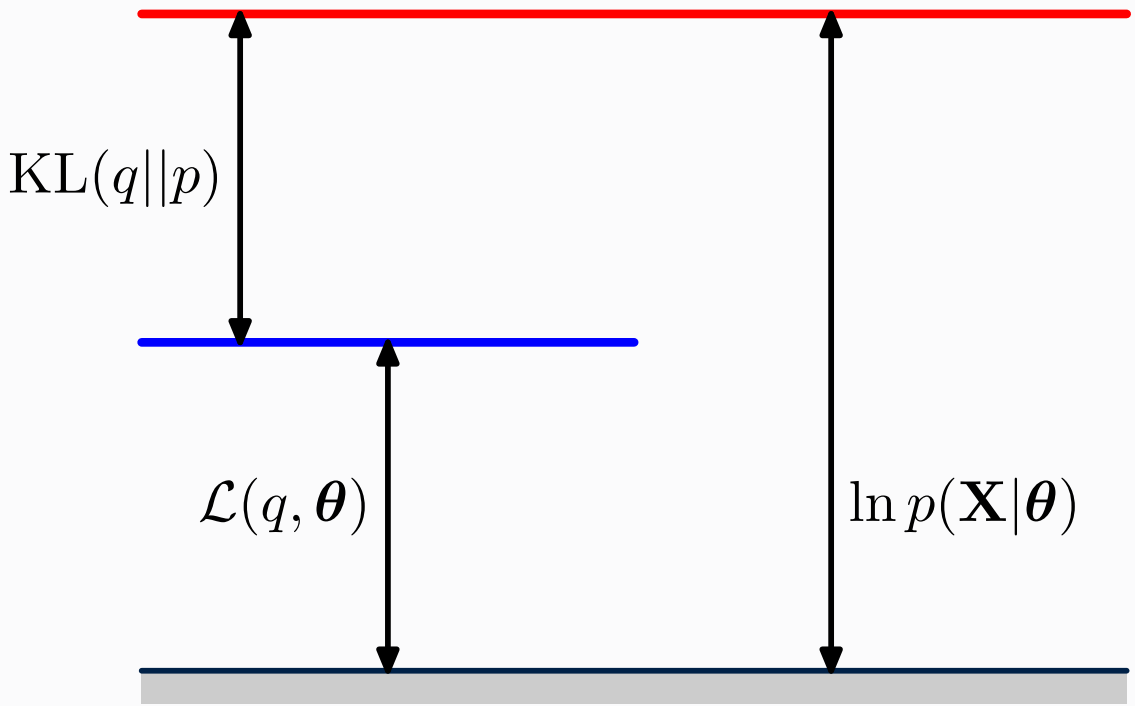
- Increase the evidence $\log p_\theta(\boldsymbol{x})$
- Reduce the divergence between $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ and $q_\phi(\boldsymbol{z})$ (remembering we are assuming the later is fixed for now)

As the KL is bounded, there is only so much we can gain by reducing it: we should expect at least some improvements in the true log evidence

# Variational EM

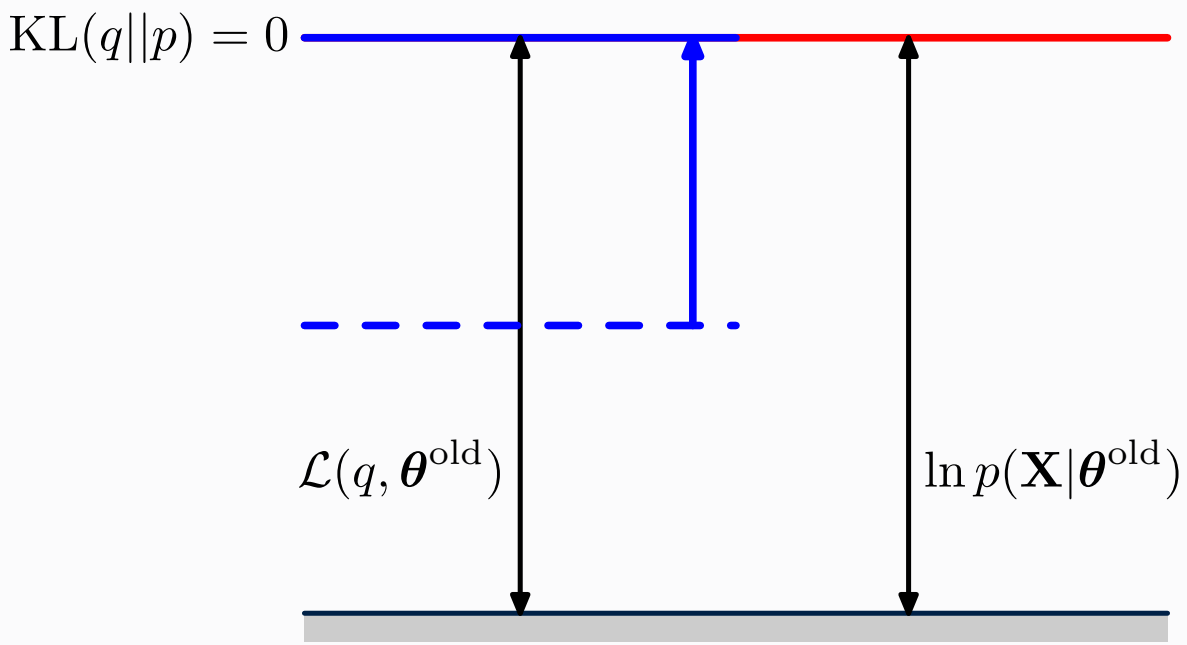
- Nonetheless, if $q_\phi(\boldsymbol{z})$ is kept fixed,this KL term is going to be restrictive to what we learn: we can only learn models whose posterior is close to $q_\phi(\boldsymbol{z})$
- We can alleviate this issue by instead alternating between updating our model and finding a new variational approximation, i.e. cycle between
  1. Update $\theta$ using SGA with fixed $\phi$
  2. Update $\phi$ by running variational inference with fixed $\theta$
- This is known as (stochastic) **variational expectation maximization** (EM)
- Note that both steps improve the ELBO

# Variational EM (2)



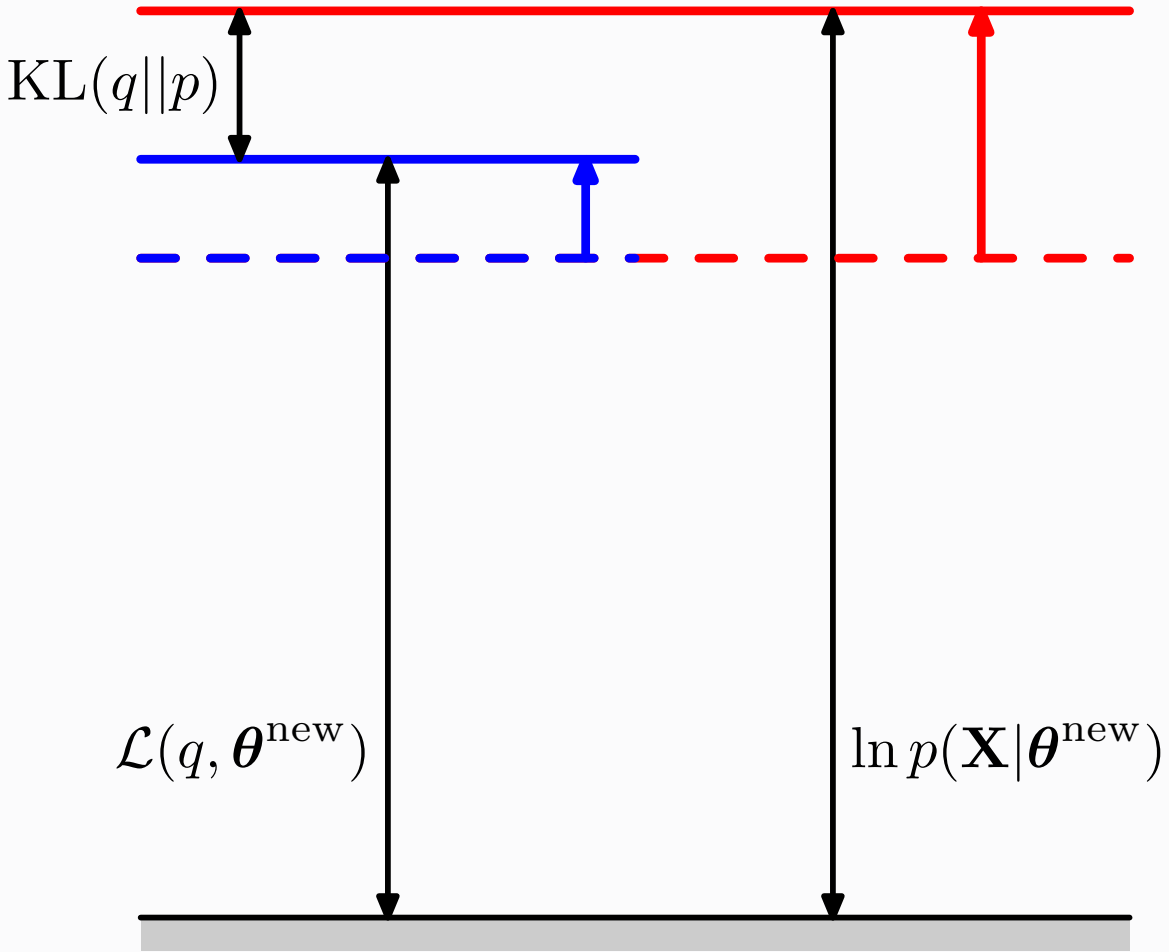

Images credit: Bishop Section 9.4

# Variational EM (2)



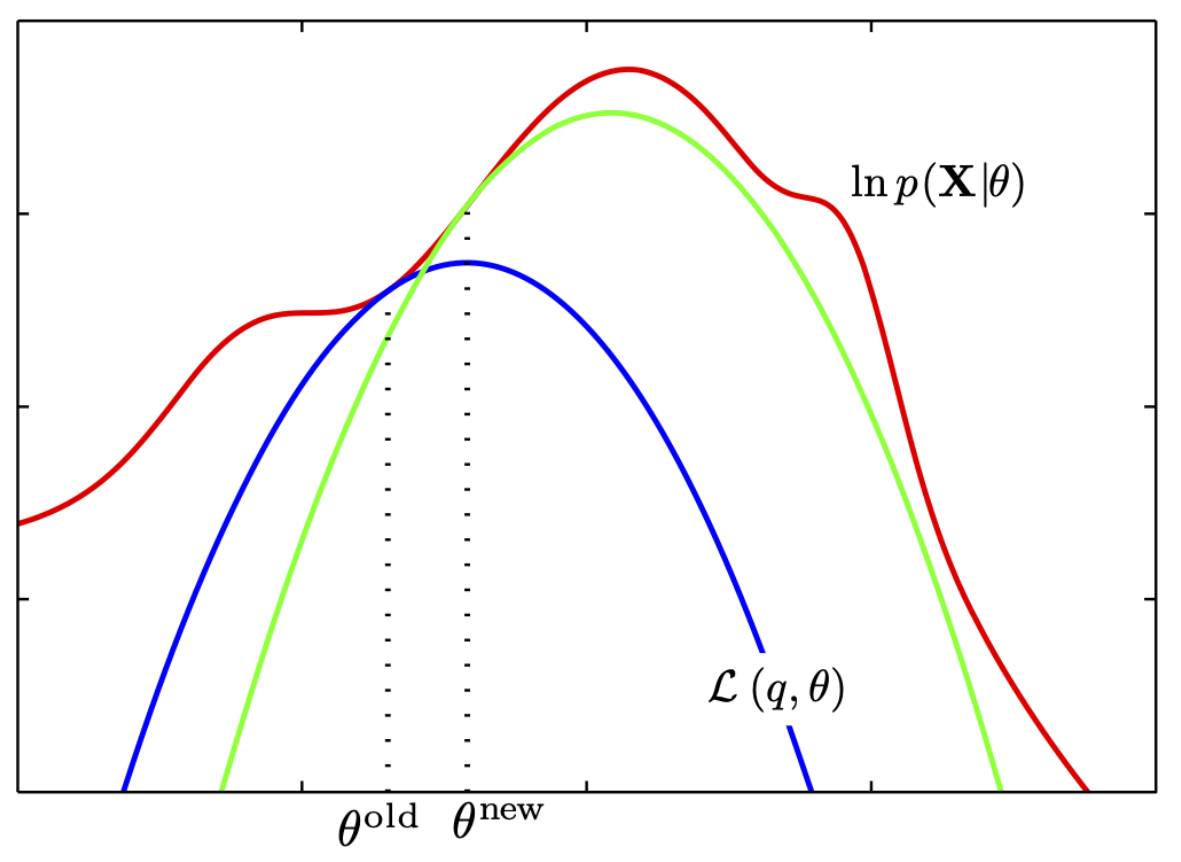

Images credit: Bishop Section 9.4

# Variational EM (2)




Images credit: Bishop Section 9.4

# Variational EM (2)




Images credit: Bishop Section 9.4

# The ELBO Through Importance Sampling Eyes

Working with the ELBO has solved our problem of needing unbiased estimates of the gradient to perform SGA.

What about our second problem though: that our estimates where very high variance?

Let's consider using $q_\phi(\boldsymbol{z})$ to estimate the marginal likelihood versus using it in the ELBO:

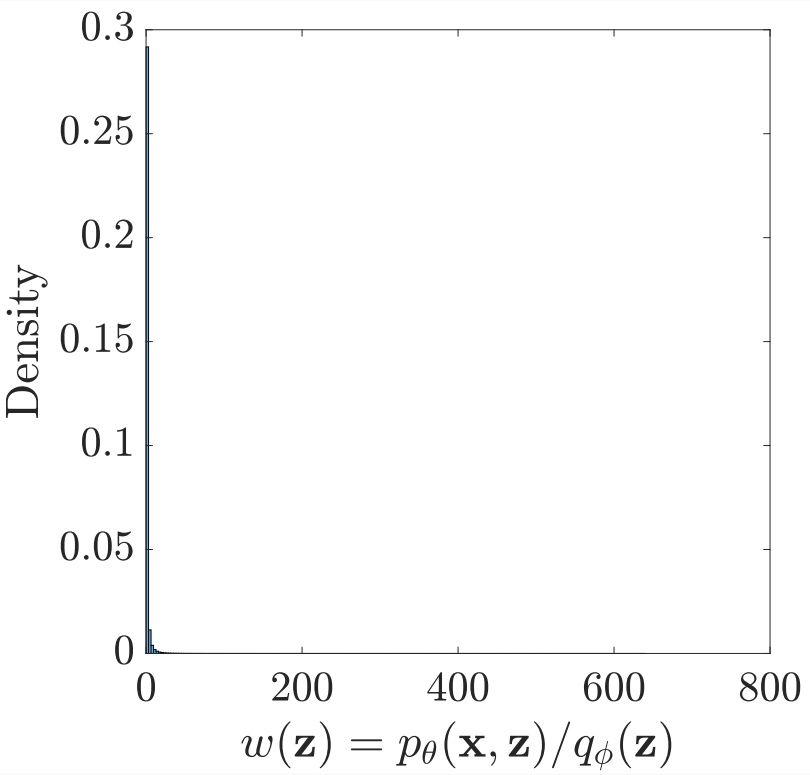
$$\mathbb{E}_{p(\boldsymbol{z})}\left[p(\boldsymbol{x}|\boldsymbol{z})\right] = \mathbb{E}_{q_\phi(\boldsymbol{z})}\left[\frac{p(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z})}\right] \quad \text{versus} \quad \mathbb{E}_{q_\phi(\boldsymbol{z})}\left[\log\left(\frac{p(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z})}\right)\right]$$

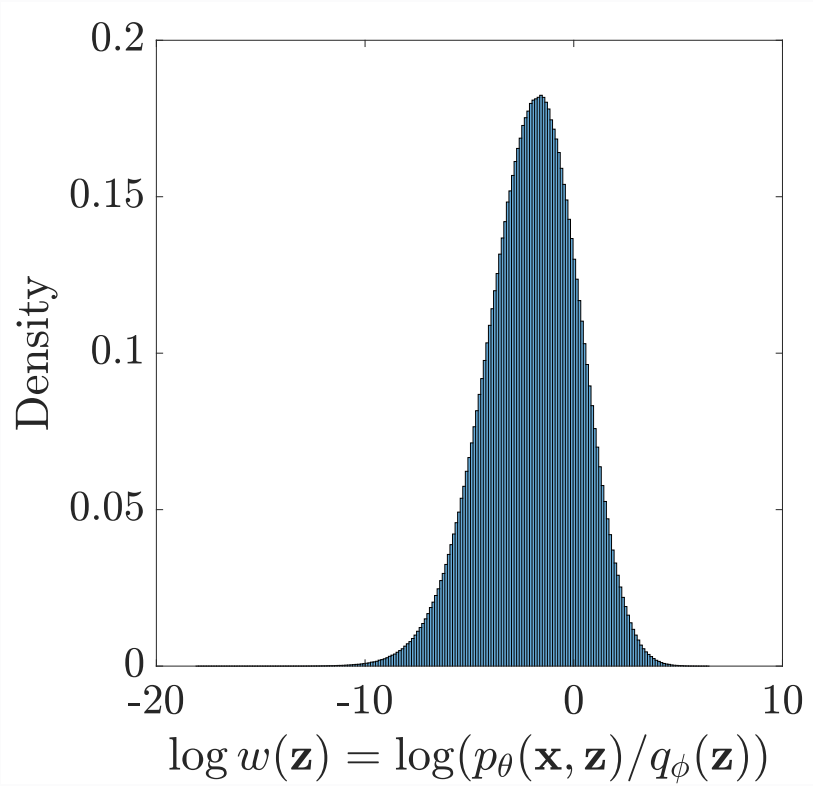
The ELBO is like working with log weights instead of the normal weights

This means its has much lower relative variance

# A Closer Look at the Weights

As a simple example, let $q_\phi(\phi)(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; 2, 1.1^2)$ and $p_\theta(\boldsymbol{x}, \boldsymbol{z}) \propto \mathcal{N}(\boldsymbol{z}; 0, 1)$. The distribution of the weights and log weights is as follows:



# Have we Solved our Problems?

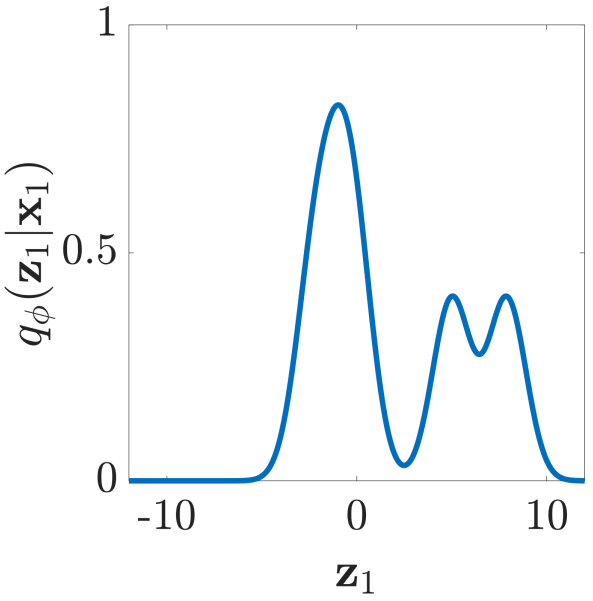
- Problem 1: having an unbiased SGA gradient estimator ✓
- Problem 2: having low variance estimates ✓
- Problem 3: Dealing with large datasets ×

# Dealing with Large Datasets

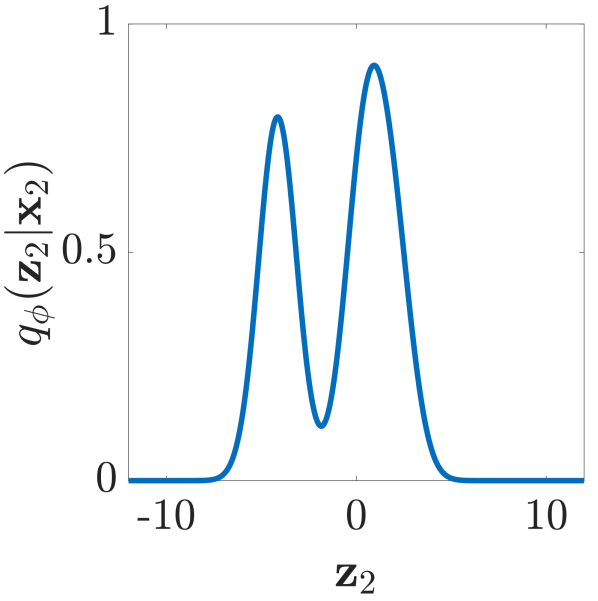
- Our current setup does not allow updates for $q_\phi(\boldsymbol{z})$ that are both scalable and effective
- If we use a single $q_\phi(\boldsymbol{z})$ shared across datapoints, then it can't simultaneously be a good fit for all of them
- We could instead use a different $\phi_{\boldsymbol{n}}$ for each datapoint, but then we either have to regularly cycle through our whole dataset to update them, or they won't get properly updated as $\theta$ changes
- Solution: learning a mapping from datapoints to $\phi$ instead, i.e. use a variational distribution of the form $q_\phi(\boldsymbol{z}|\boldsymbol{x})$
- This is known as an **amortized proposal**

# Amortized Inference

$q_\phi(\mathbf{z}|\mathbf{x})$




$q_\phi(\mathbf{z}|\mathbf{x})$



# Amortized Inference (2)

- More concretely, in amortized inference $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is a **mapping** from datapoints to proposal parameters
- This parameterized mapping takes the form of a deep neural network
- $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is often known as an **inference network**
- One common choice is to use a parameterized Gaussian:

$$q_\phi(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \mu_\phi(\boldsymbol{x}), \Sigma_\phi(\boldsymbol{x}))$$

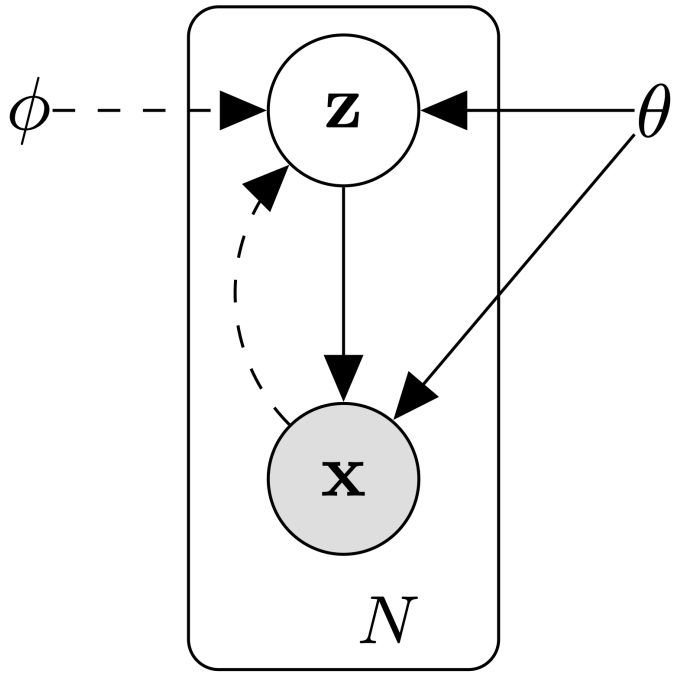
  where $\mu_\phi$ and $\Sigma_\phi$ are deep neural networks
- The upshot of this is that learning $\phi$ now corresponds to learning a mapping rather than a particular variational approximation

# Variational Auto–Encoders

# Variational Auto–Encoders

$q_\phi(z|x)$

Inference Network




$p_\theta(x|z)$

Generative Network

Maximize

$$\mathcal{L}(\theta, \phi, \mathcal{D}) := \sum_{n=1}^{N} \mathbb{E}_{q_\phi(z_n|x_n)} \left[ \log \left( \frac{p_\theta(x_n, z_n)}{q_\phi(z_n|x_n)} \right) \right] \tag{7}$$

with respect to both $\theta$ and $\phi$

# A Complication: Gradients for the Inference Network

As before, we naturally wish to maximize $\mathcal{L}(\theta, \phi, \mathcal{D})$ using SGA

However, a complication arises with this when we try and take derivatives with respect to $\phi$: these also effect the distribution the expectation is taken with respect to

$$\nabla_\phi \mathbb{E}_{q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)}\left[\log\left(\frac{p_\theta(\boldsymbol{x}_n, \boldsymbol{z}_n)}{q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)}\right)\right] = \mathbb{E}_{q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)}\left[\nabla_\phi \log\left(\frac{p_\theta(\boldsymbol{x}_n, \boldsymbol{z}_n)}{q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)}\right)\right] + \int \log\left(\frac{p_\theta(\boldsymbol{x}_n, \boldsymbol{z}_n)}{q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)}\right) \nabla_\phi q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n) d\boldsymbol{z}_n$$

This second term is not an expectation so we have extra work to do to estimate it

# The Score Function Estimator

One simple way to deal with this is to note that $\nabla x = x \nabla \log x$, such that we have

$$\nabla_\phi \, \mathbb{E}_{q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)} \left[ \log \left( \frac{p_\theta(\boldsymbol{x}_n, \boldsymbol{z}_n)}{q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)} \right) \right] =$$

$$\mathbb{E}_{q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)} \left[ \nabla_\phi \log \left( \frac{p_\theta(\boldsymbol{x}_n, \boldsymbol{z}_n)}{q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)} \right) + \log \left( \frac{p_\theta(\boldsymbol{x}_n, \boldsymbol{z}_n)}{q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)} \right) \nabla_\phi \log q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n) \right]$$

This is known as the **score function** estimator or the **reinforce** estimator.

Unfortunately, it tends to have very high variance and so is rarely used in practice.

# The Reparameterization Trick

An alternative approach which is lower variance but which can cause issues if there are discrete variables is to **reparameterize** $\boldsymbol{z}$

The key idea is to express $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ in the form $\varepsilon \sim q(\epsilon)$, $\boldsymbol{z} = g(\epsilon, \boldsymbol{x}, \phi)$, such that $q(\epsilon)$ is a simple, fixed, distribution and $g$ is a deterministic mapping that ensures $\boldsymbol{z}$ has distribution $q_\phi(\boldsymbol{z}|\boldsymbol{x})$

We can then express the expectation in terms of $\epsilon$ and thus move the gradient inside as follows

$$\nabla_\phi \, \mathbb{E}_{q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)} \left[ \log \left( \frac{p_\theta(\boldsymbol{x}_n, \boldsymbol{z}_n)}{q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)} \right) \right] =$$

$$\mathbb{E}_{q(\epsilon)} \left[ \nabla_\phi \log \left( \frac{p_\theta(\boldsymbol{x}_n, g(\epsilon_n, \boldsymbol{x}_n, \phi))}{q_\phi(g(\epsilon_n, \boldsymbol{x}_n, \phi)|\boldsymbol{x}_n)} \right) \right]$$

# Putting it All Together: the VAE Algorithm

The VAE training algorithm cycles through the following steps (presuming a reparameterization approach)

1. Sample a mini-batch of datapoints $B \subseteq \{1, \ldots, N\}$
2. Construct a gradient estimate of the ELBO using $B$

$$\hat{\nabla}_{\theta,\phi}\mathcal{L}(\theta, \phi, \mathcal{D}) = \frac{N}{\|B\|} \sum_{n \in B} \hat{\nabla}_{\theta,\phi} \log \frac{p_\theta(\hat{\boldsymbol{z}}_n, \boldsymbol{x}_n)}{q_\phi(\hat{\boldsymbol{z}}_n | \boldsymbol{x}_n)} \tag{8}$$

   where each $\hat{\boldsymbol{z}}_n = g(\varepsilon_n, \boldsymbol{x}_n, \phi)$ and $\varepsilon_n \sim q(\varepsilon)$
3. Update the parameters using these gradient estimates and step sizes $\rho_t$ and $\eta_t$

$$\theta \leftarrow \theta + \rho_t \hat{\nabla}_\theta \mathcal{L}(\theta, \phi, \mathcal{D}) \tag{9}$$

$$\phi \leftarrow \phi + \eta_t \hat{\nabla}_\phi \mathcal{L}(\theta, \phi, \mathcal{D}) \tag{10}$$

4. Repeat until convergence

# The Auto-Encoder View of VAEs

We can also view the VAE as a stochastic auto–encoder where the inference network=encoder and the generative network=decoder:

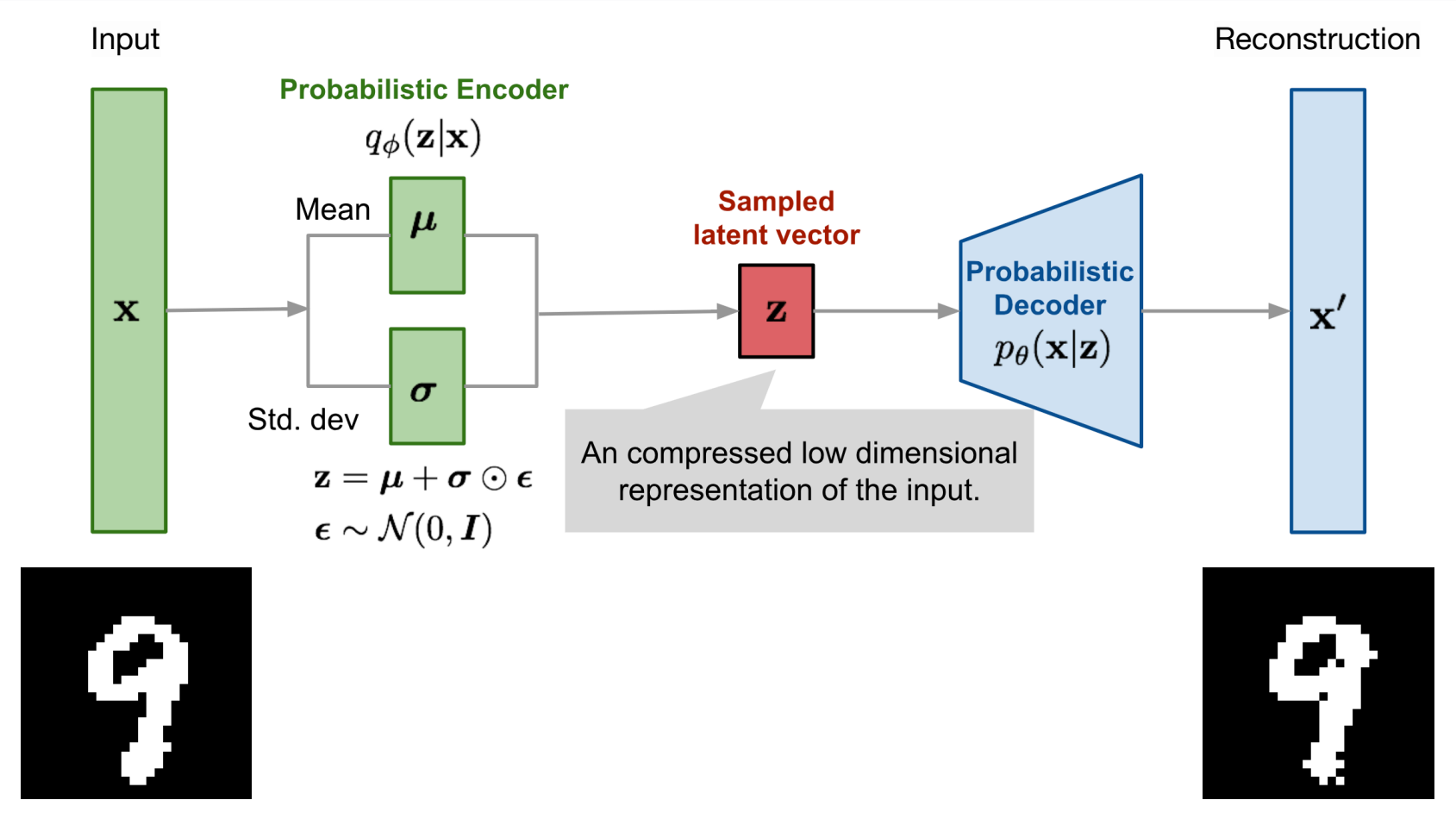
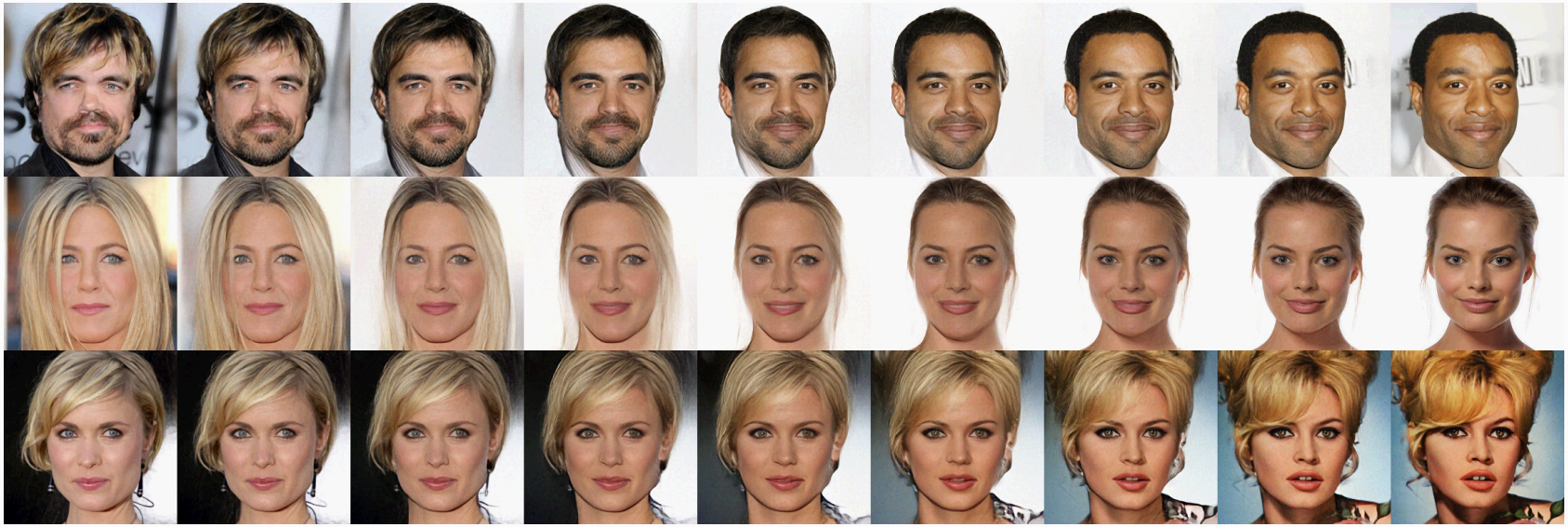



Image adapted from one by Lilian Weng

# The Auto-Encoder View of VAEs (2)

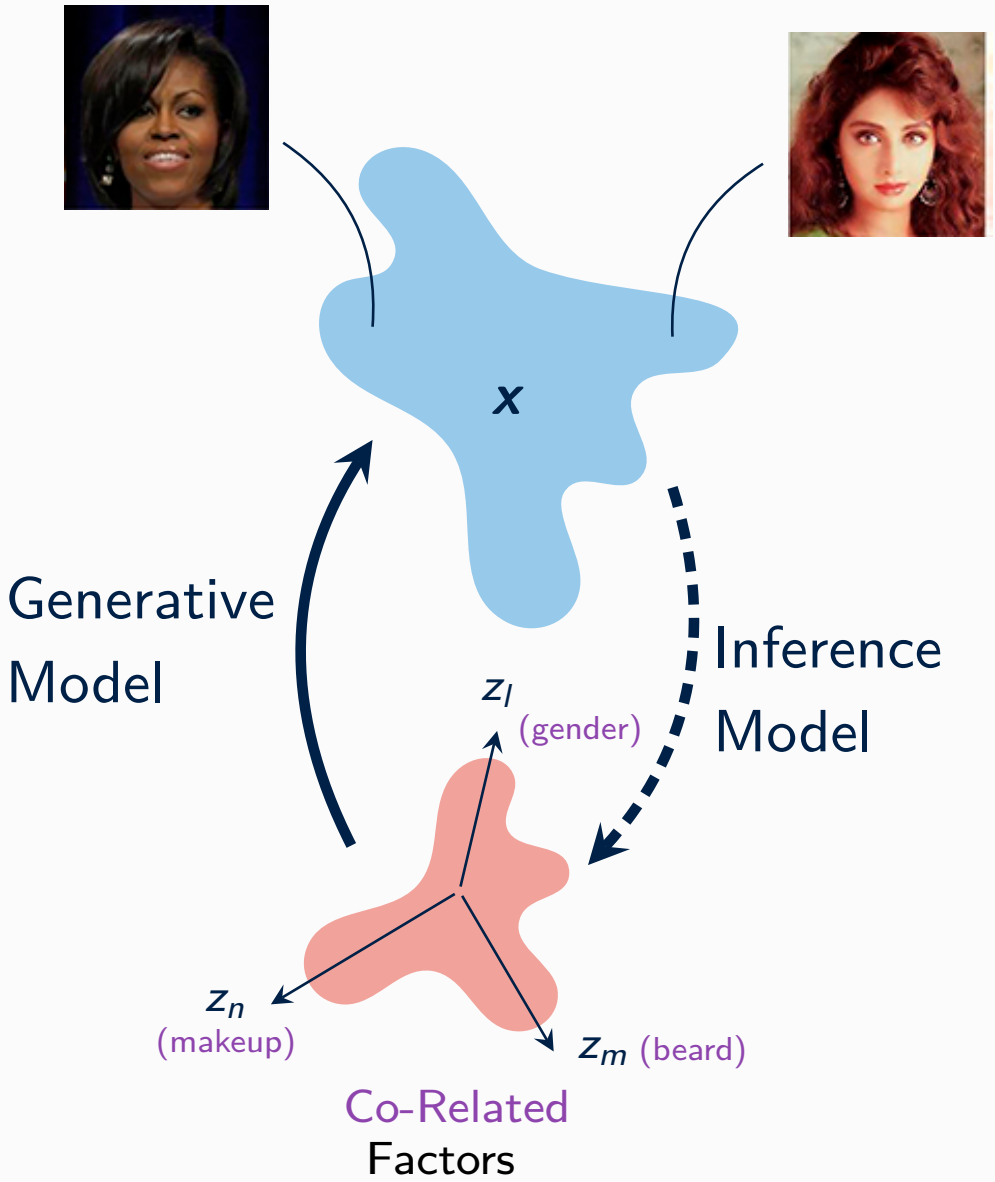
From this perspective, we can think of VAEs as a representation learning approach. We can think about manipulating images by encoding them and adjusting the latents before reconstructing, e.g. interpolating between two points[3]



[2] Kingma and Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions".

[3] Note these are not actually from a VAE, they are just demonstrative

# Disentanglement: Learning Meaningful Latents



# Disentanglement: Learning Meaningful Latents (2)


[4] Hyunjik Kim and Andriy Mnih. "Disentangling by Factorising". In: *ICML*. 2018.

# Further Reading

- The are no additional lecture notes for this lecture: you need to go investigate for yourself
- Training VAEs in Pyro: `https://pyro.ai/examples/vae.html` and `https://www.youtube.com/watch?v=vgFWeEyen6Y&t=1058s`
- Tutorial paper on VAEs: Carl Doersch. "Tutorial on variational autoencoders". In: *arXiv preprint arXiv:1606.05908* (2016)
- Video tutorial on deep generative models by Shakir Mohamed and Danilo Rezende `https://www.youtube.com/watch?v=JrO5fSskISY`
- GANs, one of the main alternatives to VAEs: Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014