# Deep Learning and Applications

DSA 5204 • Lecture 9

Dr Low Yi Rui (Aaron)
Department of Mathematics

National University of Singapore

# Test (25 Mar 2023, 10:10-10:50am)

- **Format: 20 multiple choice/selection questions**
- **Duration: 40 minutes**
- **Platform: Canvas Quiz**
- **Syllabus: First 8 slides (everything so far except generative models)**
- **Instructions**
  - *Lecture on 25 March will be **fully online***
  - *Log in to zoom session at **10am** to set up*
  - *Join the usual zoom session on your laptop/desktop. Have a second device (handphone) join the zoom session and show via its camera your hands, face and your laptop/desktop screen. Join Zoom with exactly 2 devices!*
  - ***Attendance will be taken, you need to attend the zoom session for your score to count***

# Last Time

**Previously, we introduced some unsupervised and supervised learning algorithms**

- Autoencoders
- Semi-supervised, multi-task and transfer learning

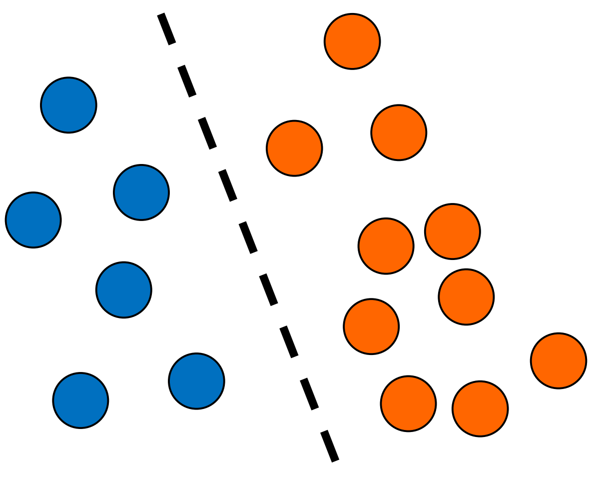
**Things in common: still focused on overcoming missing labels**

**Today and in the next lectures, we will look at very different task – generating new samples by learning distributions.**

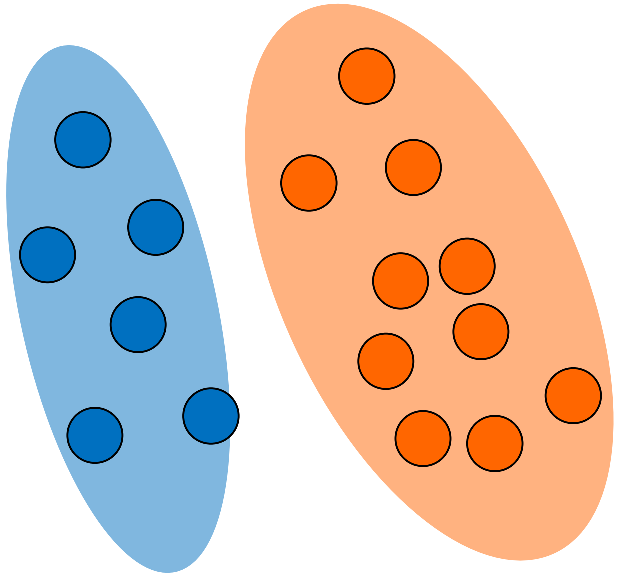
# Generative Models Overview

# Discriminative vs Generative Models

Discriminative Modelling $f^*(\boldsymbol{x})$



Generative Modelling $p^*(\boldsymbol{x})$



# Modelling Distributions

There are two main ways to model distributions, with important differences.

| Density Estimation | Generative Models |
|---|---|
| Given: $\mathcal{D} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots \boldsymbol{x}^{(N)}\}$ <br> $\boldsymbol{x}^{(i)} \sim p^*$ i.i.d. | Given: $\mathcal{D} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots \boldsymbol{x}^{(N)}\}$ <br> $\boldsymbol{x}^{(i)} \sim p^*$ i.i.d. |
| Goal: find $\hat{p} \approx p^*$ | Goal: sample new $\tilde{\boldsymbol{x}} \sim p^*$(approximately) |

Differences

- With $\hat{p}$, we may not be able to sample from it easily
- Even if we can sample approximately from $p^*$, we may not have a good representation of it

# Gaussian Mixture Models

**For Gaussian mixture models (GMM), we consider a parametric family**

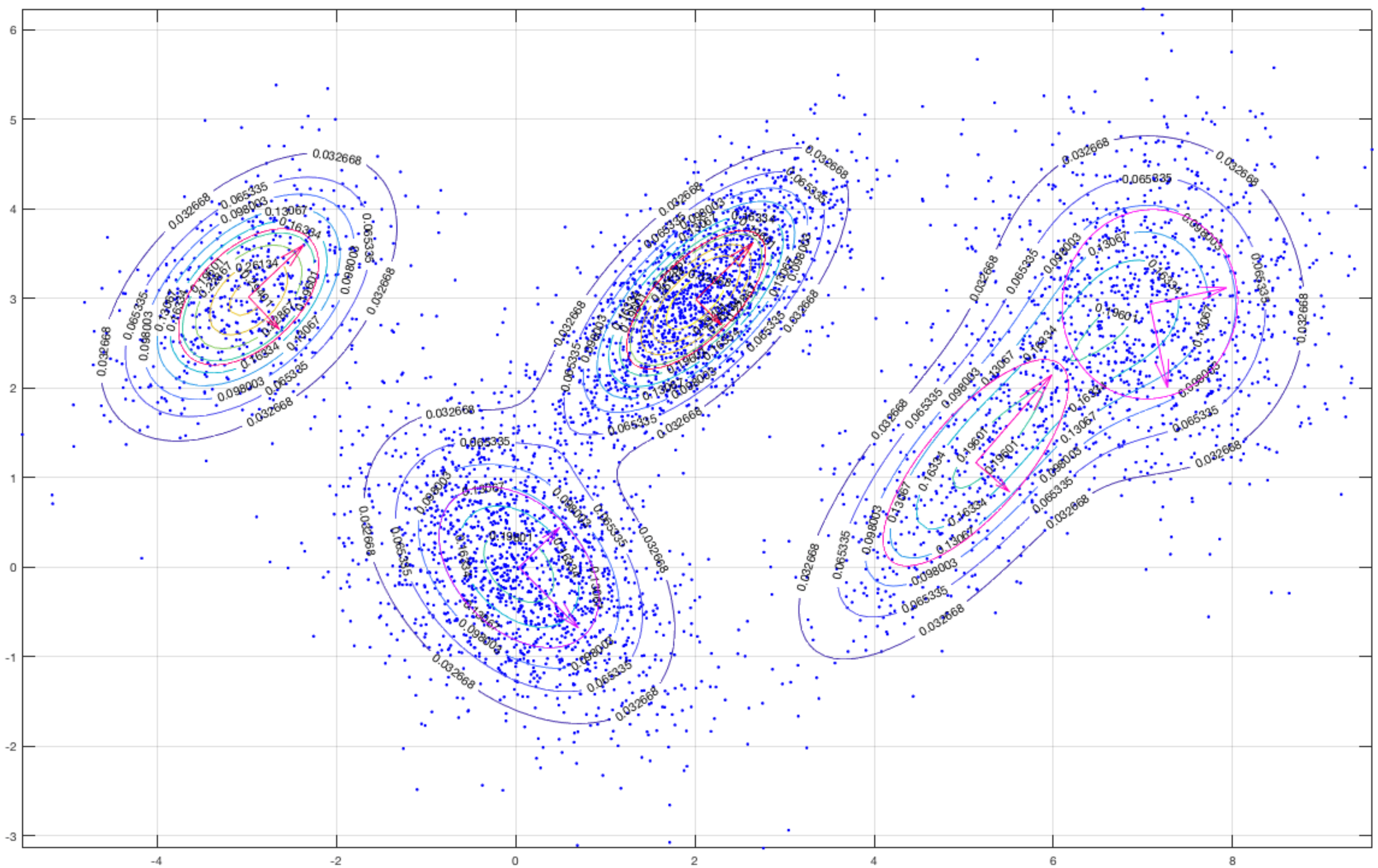
$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_j \alpha_j p_{\boldsymbol{\mu}_j,\Sigma_j}(\boldsymbol{x}) \quad \alpha_j \geq 0 \quad \sum_j \alpha_j = 1$$

where $p_{\boldsymbol{\mu},\Sigma}(x)$ **is the PDF of Gaussian RVs with mean $\mu$ and covariance matrix $\Sigma$**

**Once fitted, we can generate samples from $p_{\boldsymbol{\theta}}$, by**

- Generate index $i$ from distribution $\{\alpha_j\}$
- Generate from $p_{\boldsymbol{\mu}_i,\Sigma_i}$

# Gaussian Mixture Models

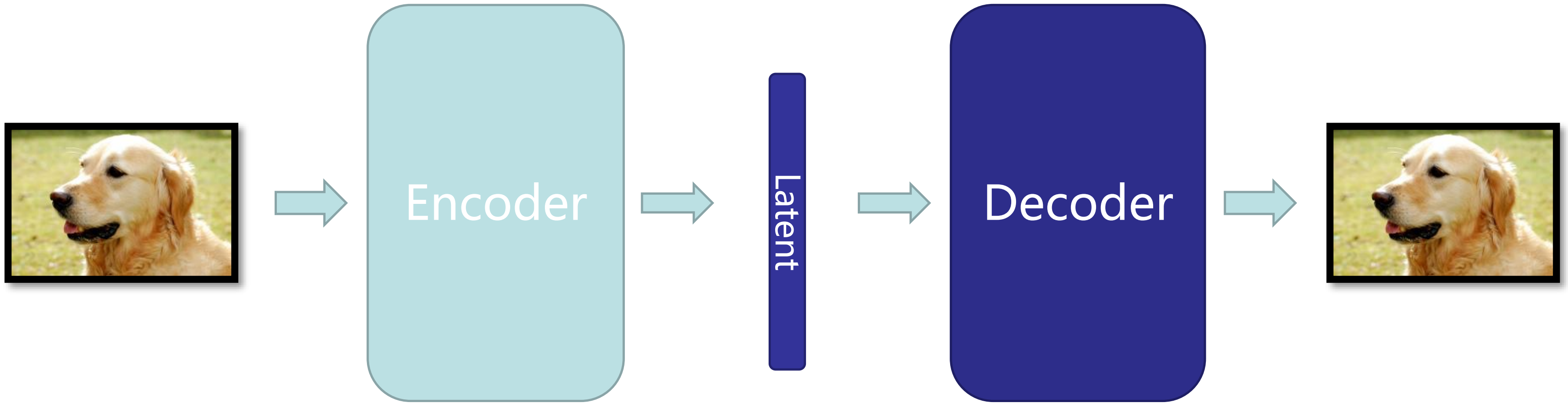


# Limitations of GMM

- A simple mixture of Gaussian is rarely enough to model high dimensional data
- Number of Gaussians is a hyper-parameter that is hard to tune in practice
- Cannot use domain knowledge (e.g. translation invariance, CNNs)

# Deep Learning Based Generative Model?

**Recall that we introduced the autoencoder architecture**

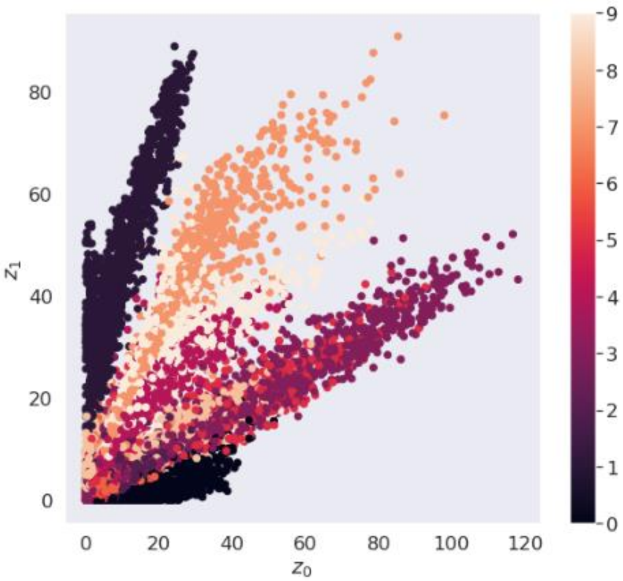


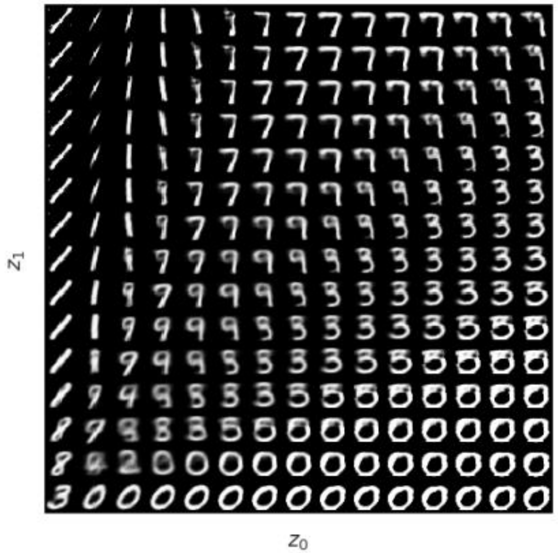

Idea: generating something in latent space, and then decode?

# Demo: Generating Samples using Autoencoder

# The issue with AEs

## The latent space representation is not “continuous”




Therefore, to improve on this, we should try to make the latent space correspond smoothly with the output!

# Variational Autoencoder

# The Maximum Likelihood Approach

**Given $p^*(\boldsymbol{x})$, consider**

- A parametric model $p_{\boldsymbol{\theta}}(\boldsymbol{x})$
- A dataset $\mathcal{D} = \left\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots \boldsymbol{x}^{(N)}\right\}$ with $\boldsymbol{x}^{(i)} \sim p^*(\boldsymbol{x})$.

**One way to estimate $p^*$ using $p_{\boldsymbol{\theta}}$ is the *maximum likelihood estimator (MLE)*:**

$$\widehat{\boldsymbol{\theta}} = \operatorname*{argmax}_{\boldsymbol{\theta}} \prod_i p_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right)$$

**This is equivalent to maximizing the log-likelihood**

$$\widehat{\boldsymbol{\theta}} = \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_i \log\left(p_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right)\right)$$

# Example: MLE for Gaussian Family
## <Lecture Notebook>

**Consider $\theta = (\mu, \sigma)$ and the parametric Gaussian family in 1D**

$$p_{\boldsymbol{\theta}}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$

**Applying MLE to a dataset $\{x^{(1)}, x^{(2)}, \dots, x^{(N)}\}$**

$$\widehat{\boldsymbol{\theta}} = (\hat{\mu}, \hat{\sigma}) = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}\left(x^{(i)}\right)$$

**We find**

$$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x^{(i)} \qquad \hat{\sigma}^2 = \frac{1}{N}\sum_{i=1}^{N}\left(x^{(i)} - \hat{\mu}\right)^2$$

# Sampling using MLEs

**Once we found $\hat{\theta}$, we can generate samples**

$$x' \sim p_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x})$$

**Limitations of this approach**

- If we use a simple parametric form $p_{\boldsymbol{\theta}}$ (e.g. Gaussian), then it may not be a good model for the data
- If we use a complex parametric form (e.g. NNs), then there is often no easy way to perform optimization and sampling…

**This motivates another way to build generative models using *latent random variables***

# Latent Generative Models

**An alternative to direct modelling is to consider *latent* generative models.**

**Here, we write**

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})\boldsymbol{dz}$$

where

- $\boldsymbol{z}$: *latent* variable
- $p_{\boldsymbol{\theta}}(\boldsymbol{z})$: latent variable *prior* distribution
- $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$: *generative/ conditional* distribution

# Example: Latent Generative Model

**Let**

- $z$ **be a uniform random variable on the interval** $[a, b]$
- $x$ **be conditional random normal with mean** $\mu$ **and variance** $z^2$

**Then**

$$p_{\boldsymbol{\theta}}(z) = \frac{1}{b-a} \mathbb{I}_{a \le z \le b}$$

$$p_{\boldsymbol{\theta}}(x|z) = \frac{1}{\sqrt{2\pi z^2}} e^{-\frac{(x-\mu)^2}{2z^2}}$$

with $\boldsymbol{\theta} = (a, b, \mu)$

# Direct vs Latent Approach

**Given a data point $x$**

**Direct Approach:**

- Learn $p_{\boldsymbol{\theta}}(\boldsymbol{x})$
- Sample $\boldsymbol{x}' \sim p_{\boldsymbol{\theta}}(\boldsymbol{x})$

**Latent Variable Approach**

- Learn posterior latent distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$
- Learn generative distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$
- Sample from $\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, and then $\boldsymbol{x}' \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$

# Posterior Model for $p_\theta(z|x)$

Computationally Intractable

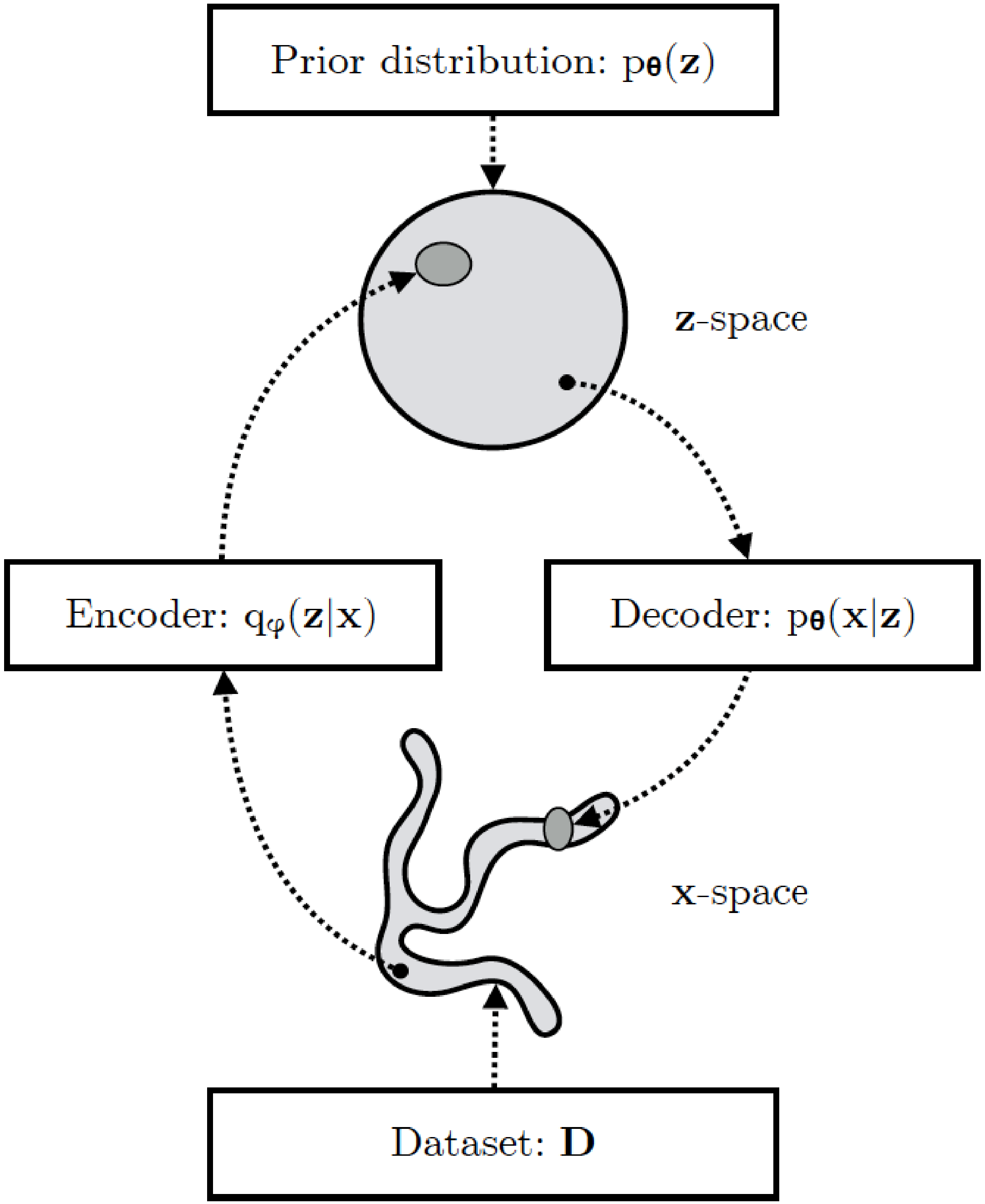
**By Bayes theorem, we have**

$$p_\theta(z|x) = \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})p_\theta(\boldsymbol{z})}{\boxed{\int p_\theta(\boldsymbol{x}, \boldsymbol{z}')d\boldsymbol{z}'}} \rightarrow P_\theta(x)$$

**Instead of computing this, we will resort to a** ***posterior model***

$$q_\phi(\boldsymbol{z}|\boldsymbol{x}) \approx p_\theta(\boldsymbol{z}|\boldsymbol{x})$$

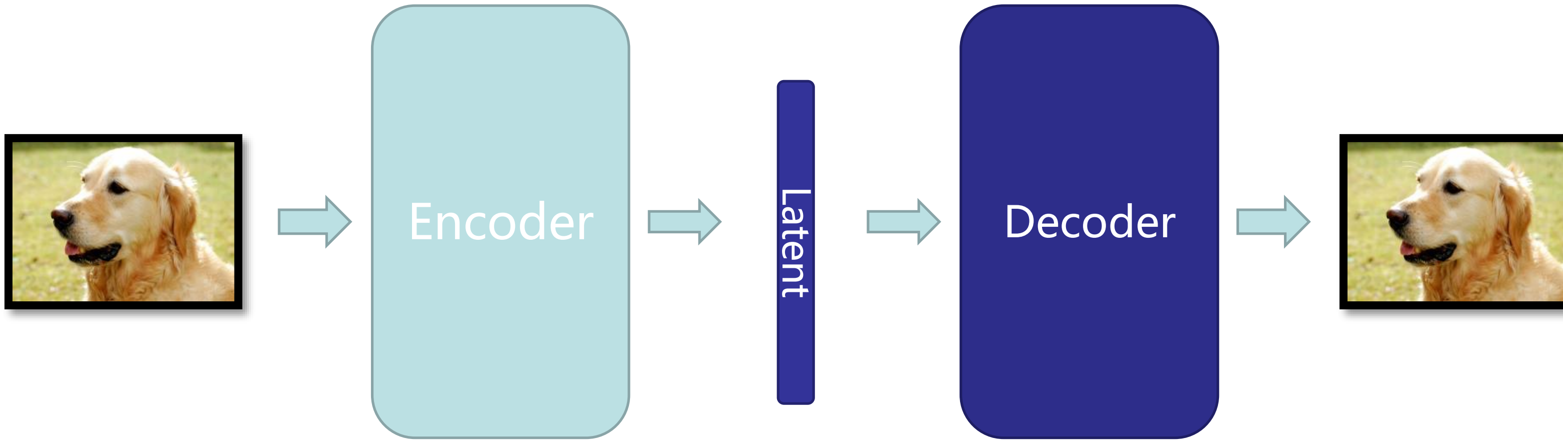
whose parameters $\phi$ **can be trained to make this approximation good.**


Prior distribution: $p_{\theta}(\mathbf{z})$
$\mathbf{z}$-space
Encoder: $q_{\varphi}(\mathbf{z}|\mathbf{x})$
Decoder: $p_{\theta}(\mathbf{x}|\mathbf{z})$
$\mathbf{x}$-space
Dataset: $\mathbf{D}$

# Comparing with Autoencoders

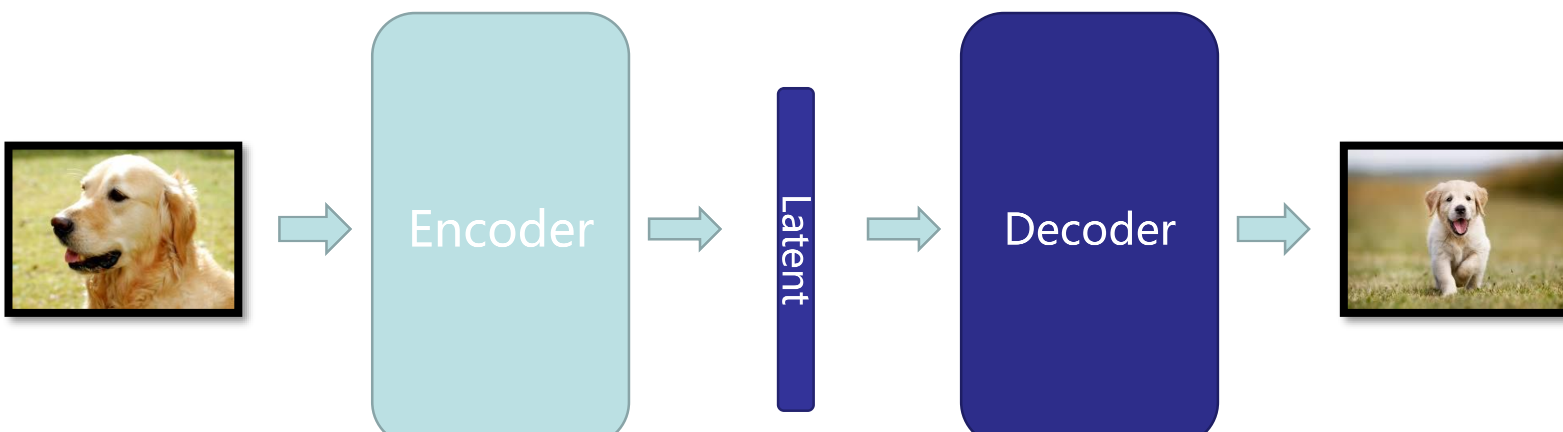
Encoder: $\boldsymbol{z} = f_{\boldsymbol{\phi}}(\boldsymbol{x})$

Decoder: $\boldsymbol{x}' = g_{\boldsymbol{\theta}}(\boldsymbol{z})$




Encoder: $\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$

Decoder: $\boldsymbol{x}' \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$



# Variational Autoencoders

**In this sense, the latent variable generative process is very much like an autoencoder. Thus, we call it a variational autoencoder (VAE).**

**The term "variational" refers to our approximation of the posterior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ model by $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$.**

**It remains to ask:**

- How do we parameterize $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ (generative distribution) and $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ (posterior/variational distribution)?
- How do we train $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$?

# The Loss for VAEs

# Training Autoencoders vs VAEs

**Recall that in AEs, we can simply train the network using inputs as labels**

$$\min_{\boldsymbol{\theta},\boldsymbol{\phi}} L(\boldsymbol{x}, \boldsymbol{x}') = L\left(\boldsymbol{x}, g_{\boldsymbol{\theta}}\left(f_{\boldsymbol{\phi}}(\boldsymbol{x})\right)\right)$$

**In the probabilistic case of VAEs, $x'$ and $x$ are *not required* (rather, *required not*) to be the same!**

**Instead, we train their *distributions* $q_{\boldsymbol{\phi}}, p_{\boldsymbol{\theta}}$. So, how can we do training?**

# Preliminaries I: KL Divergence

**Given two densities $p(x)$ and $q(x)$. The Kullback-Leibler (KL) Divergence is a way to measure their "distance" from one another.**

**For continuous RVs, we have**

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

**For discrete RVs, replace integral by sum**

**Important Properties:**

- $D_{KL}(p\|q) \geq 0$ with equality if and only if $p = q$
- $D_{KL}(p\|q)$ is convex in both $p$ and $q$
- $D_{KL}(p\|q)$ is not symmetric (in particular, not a metric)

# Example: KL Divergence of Gaussians <Lecture Notebook>

**Consider two Gaussian PDFs in 1D**

$$p_{\boldsymbol{\theta}_1}(x) = \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{1}{2\sigma_1^2}(x-\mu_1)^2\right)$$

$$p_{\boldsymbol{\theta}_2}(x) = \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{1}{2\sigma_2^2}(x-\mu_2)^2\right)$$

**Then,**

$$D_{KL}\left(p_{\boldsymbol{\theta}_1} \| p_{\boldsymbol{\theta}_2}\right) = \log\frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2}{2\sigma_2^2} - \frac{1}{2} + \frac{(\mu_1-\mu_2)^2}{2\sigma_2^2}$$

**Note the properties of $D_{KL}$**

- $D_{KL}\left(p_{\boldsymbol{\theta}_1} \| p_{\boldsymbol{\theta}_2}\right) \geq 0$. Note $-\log u + \frac{1}{2}u^2 - \frac{1}{2} \geq 0$ for $u > 0$
- $D_{KL}\left(p_{\boldsymbol{\theta}_1} \| p_{\boldsymbol{\theta}_2}\right) = 0$ iff $\mu_1 = \mu_2$ and $\sigma_1 = \sigma_2$
- $D_{KL}\left(p_{\boldsymbol{\theta}_1} \| p_{\boldsymbol{\theta}_2}\right) \neq D_{KL}\left(p_{\boldsymbol{\theta}_2} \| p_{\boldsymbol{\theta}_1}\right)$

# Preliminaries II: Monte-Carlo Estimation of Expectations/Integrals

**Given a random variable $x \in \mathbb{R}^d$ with density $p$ and a function $f: \mathbb{R}^d \to \mathbb{R}$, we wish to compute the expectation**

$$\mathbb{E}[f(\boldsymbol{x})] = \int f(\boldsymbol{x})p(\boldsymbol{x})\boldsymbol{dx}$$

**An efficient way (especially for large $d$) is *Monte-Carlo approximation*, where**

$$\mathbb{E}[f(\boldsymbol{x})] \approx \frac{1}{N}\sum_{i=1}^{N} f\left(\boldsymbol{x}^{(i)}\right) \quad \boldsymbol{x}^{(\boldsymbol{i})} \sim p \ \ (\mathrm{i.i.d.})$$

which becomes exact in the limit of $N \to \infty$.

# Evidence Lower Bound (ELBO)
## <Lecture Notebook>

**Let us now apply MLE estimation, but now on the latent model**

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \log \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})\boldsymbol{dz}$$

**The problem is that we can't easily compute the integral, so we will use some tricks.**

**We can show that**

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right)\right]}_{\substack{L(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi}) \\ \text{Evidence Lower Bound (ELBO)}}} + \underbrace{D_{KL}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\|p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\right)}_{\substack{K(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi}) \\ \text{KL Divergence}}}$$

# Optimizing the ELBO

**Note that the KL divergence is non-negative, hence**

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = L(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi}) + K(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi}) \geq L(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi})$$

**In other words,** $L$ ***is a lower bound for the log-likelihood.***

**Therefore, we can replace**

$$\max_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \quad \rightarrow \quad \max_{\boldsymbol{\theta},\boldsymbol{\phi}} L(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi})$$

**Key point: the latter can be computed without computing the exact posterior** $p_{\boldsymbol{\theta}}(z|x)$**!**

# Multiple Datapoints and GD

**The case for multiple i.i.d. datapoints is completely analogous**

$$L(\mathcal{D};\boldsymbol{\theta},\boldsymbol{\phi}) = \frac{1}{N}\sum_{i=1}^{N} L\left(\boldsymbol{x}^{(i)};\boldsymbol{\theta},\boldsymbol{\phi}\right)$$

$$= \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{q_{\boldsymbol{\phi}}\left(\boldsymbol{z}|\boldsymbol{x}^{(i)}\right)}\left[\log\left(\frac{p_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)},\boldsymbol{z}\right)}{q_{\boldsymbol{\phi}}\left(\boldsymbol{z}|\boldsymbol{x}^{(i)}\right)}\right)\right]$$

**As always, we optimize $L$ via (stochastic) gradient descent on $\theta,\phi$, which requires the computation of the gradients**

$$\nabla_{\boldsymbol{\theta}}L \quad \text{and} \quad \nabla_{\boldsymbol{\phi}}L$$

# Gradient with respect to $\theta$

**The gradient with respect to $\theta$ is easy**

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

**This can be computed by Monte-Carlo approximation provided we can sample from $q_{\boldsymbol{\phi}}(\cdot\,|\boldsymbol{x})$.**

# Gradient with respect to $\phi$

**The gradient with respect to $\phi$ is more involved, since**

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

**Hence, we need to rewrite this in order to**

- Apply Monte-Carlo approximation
- Apply back-propagation

# The Reparameterization Trick

**Suppose we have a random variable $z$ with a parameterized conditional distribution**

$$\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$$

**We can reparametrize $z$ as a *deterministic transformation of a fixed random variable***

$$\boldsymbol{z} = g(\boldsymbol{u}, \boldsymbol{\phi}, \boldsymbol{x})$$

where

- $g$ is a deterministic function depending on $\boldsymbol{\phi}$
- $\boldsymbol{u} \sim p_0(\boldsymbol{u})$ is fixed and independent of $\boldsymbol{\phi}$

# Example: Reparametrizing Gaussians

**A $d$-dimensional Gaussian random variable with mean $\mu +$ $x$ and covariance matrix $C$ can be parameterized by**

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = (2\pi)^{-\frac{d}{2}} \det(C)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\boldsymbol{z} - \boldsymbol{\mu} - \boldsymbol{x})^T C^{-1} (\boldsymbol{z} - \boldsymbol{\mu} - \boldsymbol{x})\right)$$

**where $\boldsymbol{\phi} = (\boldsymbol{\mu}, C)$**

**Using the reparameterization trick, we can write**

$$\boldsymbol{z} = g(\boldsymbol{u}, \boldsymbol{\phi}, \boldsymbol{x})$$

**where $\boldsymbol{u} \sim \mathcal{N}(0, I)$ and**

$$g(\boldsymbol{u}, \boldsymbol{\phi}, \boldsymbol{x}) = \boldsymbol{\mu} + \boldsymbol{x} + C^{\frac{1}{2}}\boldsymbol{u} \ \sim \ \mathcal{N}(\boldsymbol{\mu} + \boldsymbol{x}, C)$$

# Why Reparametrize?

Consider any function $f$. We have

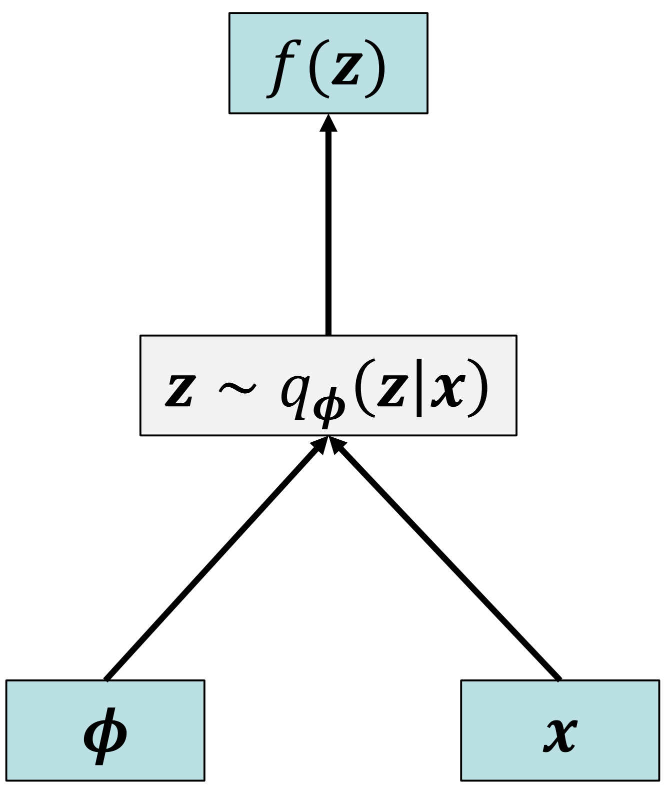
$$\mathbb{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[f(\boldsymbol{z})] = \mathbb{E}_{\boldsymbol{u}\sim p_0(\boldsymbol{u})}\left[f\left(g(\boldsymbol{u},\boldsymbol{\phi},\boldsymbol{x})\right)\right]$$

Then, we have

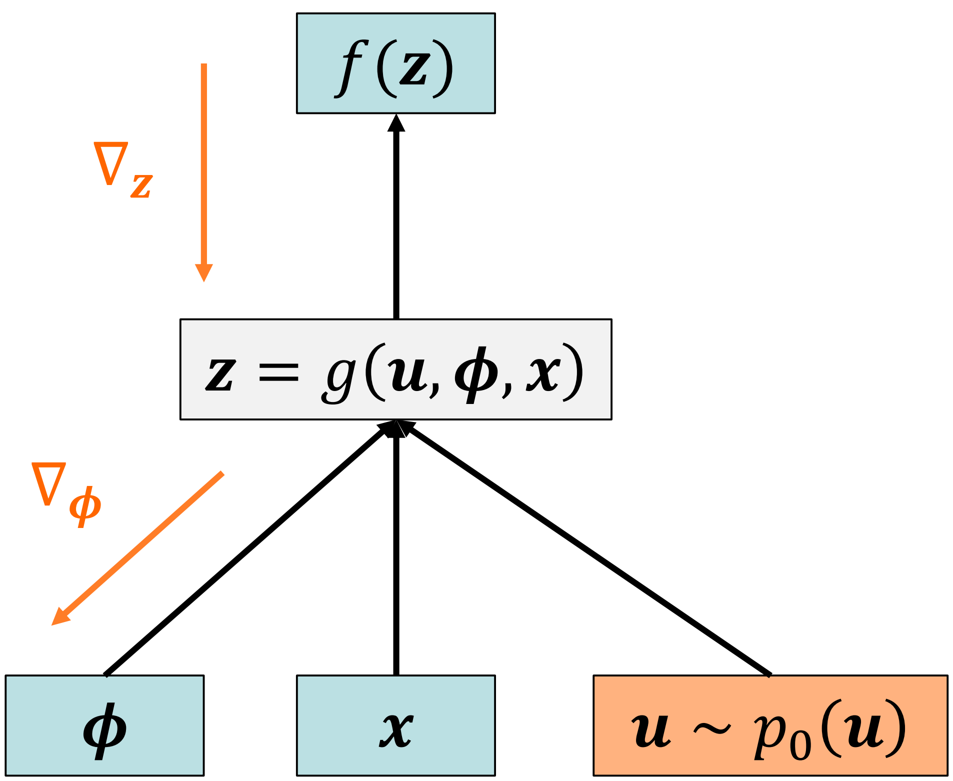
$$\nabla_{\boldsymbol{\phi}}\mathbb{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[f(\boldsymbol{z})] = \mathbb{E}_{\boldsymbol{u}\sim p_0(\boldsymbol{u})}\left[\nabla_{\boldsymbol{\phi}} f\left(g(\boldsymbol{u},\boldsymbol{\phi},\boldsymbol{x})\right)\right]$$

The latter can be approximated by Monte-Carlo averages!

## Original Form




## Reparametrized Form



# Gradient with respect to $\phi$ revisited

**Under reparameterization, we have**

$$L(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - \log\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right)\right]$$
$$= \mathbb{E}_{p_0(\boldsymbol{u})}\left[\log p_{\boldsymbol{\theta}}\left(\boldsymbol{x}, g(\boldsymbol{u},\boldsymbol{\phi},\boldsymbol{x})\right) - \log\left(q_{\boldsymbol{\phi}}(g(\boldsymbol{u},\boldsymbol{\phi},\boldsymbol{x})|x)\right)\right]$$

**And hence**

$$\nabla_{\boldsymbol{\phi}} L(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi}) = \mathbb{E}_{p_0(\boldsymbol{u})}\left[\nabla_{\boldsymbol{\phi}} \log p_{\boldsymbol{\theta}}\left(\boldsymbol{x}, g(\boldsymbol{u},\boldsymbol{\phi},\boldsymbol{x})\right) - \nabla_{\boldsymbol{\phi}}\log\left(q_{\boldsymbol{\phi}}(g(\boldsymbol{u},\boldsymbol{\phi},\boldsymbol{x})|x)\right)\right]$$

**This can be computed by Monte-Carlo approximation.**

# Neural Network Representations

**Since the prior $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ is not conditioned on any data, we may pick it to be some simple form, e.g. factorized/iid Gaussians**

**Next, we need to discuss how to represent the densities**

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) \text{ and } q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$$

**In practice, we should do it in such a way that**

- They can be easily evaluated
- They can be easily differentiated (e.g. by back-prop)
- We can easily sample from them

# Representing $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$

**Suppose $x \in \{0,1\}^d$ (binary data). Then a common choice is the *factorized Bernoulli model.***

**Recall: Let $x$ be a scalar Bernoulli random variable, then**

$$x = \begin{cases} 1 & \text{with prob } s \\ 0 & \text{with prob } 1-s \end{cases}$$

**Here $s \in [0,1]$ is the probability of successful outcome**

**The distribution function then depends on $s$, and is given by**

$$p_s(x) = s^x(1-s)^{1-x}$$

# Representing $p_{\boldsymbol{\theta}}(x|z)$

**In general, we may represent $p_{\boldsymbol{\theta}}(x|z)$ as *factorized Bernoulli model* with the success probability vector $s$ being a deterministic function of the latent variable $z$**

$$\boldsymbol{s} = \mathrm{DecodingNN}(\boldsymbol{z}; \boldsymbol{\theta}) \quad \boldsymbol{s} \in [0,1]^d$$

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = \prod_j s_j^{x_j} (1 - s_j)^{1-x_j}$$

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = \sum_j x_j \log s_j + (1 - x_j) \log(1 - s_j)$$

**The last term is just the negative of the binary cross-entropy loss!**

# Representing $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$

**Similarly, we can pick the simplest case of *factorized Gaussian model* for the encoding distribution**

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) = \mathrm{EncodingNN}(\boldsymbol{x}; \boldsymbol{\phi})$$

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \prod_j \left(2\pi\sigma_j^2\right)^{-\frac{1}{2}} \exp\left(-\frac{1}{2\sigma_j^2}\left(z_j - \mu_j\right)^2\right)$$

**Under the reparameterization trick, we have**

$$\boldsymbol{z} = g(\boldsymbol{u}, \boldsymbol{\phi}, \boldsymbol{x}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \circ \boldsymbol{u} \qquad \boldsymbol{u} \sim p_0 = \mathcal{N}(\boldsymbol{0}, I)$$

**Giving**

$$\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \sum_j -\frac{1}{2}\log\left(2\pi\sigma_j^2\right) - \frac{1}{2}u_j^2 \rightarrow u_j = \frac{z_j - \mu_j}{\sigma_j}$$

# Simplifying the ELBO Loss
## <Lecture Notebook>

**Under the current choices, we can dramatically simplify the ELBO loss**

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right)\right] = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right)\right]$$
$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})]$$
$$= \mathbb{E}_{p_0(\boldsymbol{u})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{g}(\boldsymbol{u},\boldsymbol{\phi},\boldsymbol{x})) + \log p_{\boldsymbol{\theta}}(\boldsymbol{g}(\boldsymbol{u},\boldsymbol{\phi},\boldsymbol{x})) - \log q_{\boldsymbol{\phi}}(\boldsymbol{g}(\boldsymbol{u},\boldsymbol{\phi},\boldsymbol{x})|\boldsymbol{x})]$$

**We have**

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = \sum_j x_j \log s_j + (1 - x_j)\log(1 - s_j)$$

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{z}) = -\frac{1}{2}\sum_j [z_j^2 + \log(2\pi)] = -\frac{1}{2}\|u\|^2 - \frac{1}{2}\|\sigma\|^2$$

$$\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = -\frac{1}{2}\sum_j [u_j^2 + \log(2\pi) + 2\log(\sigma_j)]$$

The only term that requires further attention is

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{z}) = -\frac{1}{2}\sum_j \left[z_j^2 + \log(2\pi)\right]$$

$z_j = \mu_j + u_j \sigma_j$

Since we have to get rid of $z$ in order to let the gradient flow through the reparameterization trick.

We can show that (dropping constants)

$$\mathbb{E}_{\boldsymbol{p_0}(\boldsymbol{u})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{z})] = \mathbb{E}_{\boldsymbol{u}\sim\mathcal{N}(\boldsymbol{0},I)}\left[-\frac{1}{2}\sum_j \left(\mu_j + \sigma_j u_j\right)^2\right]$$

$$= -\frac{1}{2}\sum_j \left(\mu_j^2 + \sigma_j^2\right)$$

# Summary of VAE (Basic Form)

**Input dimension** $= d$**. Latent dimension** $= m$**.**

**Encoder:**

$$(\underbrace{\boldsymbol{y}_1}_{\boldsymbol{\mu}}, \underbrace{\boldsymbol{y}_2}_{\log \boldsymbol{\sigma}}) = \mathrm{EncodingNN}(\boldsymbol{x}; \boldsymbol{\phi})$$

$$\boldsymbol{z} = \boldsymbol{y}_1 + e^{\boldsymbol{y}_2} \circ \boldsymbol{u} \quad \boldsymbol{u} \sim \mathcal{N}(\mathbf{0}, I) \qquad (\boldsymbol{u} \in \mathbb{R}^m)$$

**Decoder:**

$$\boldsymbol{s} = \mathrm{DecodingNN}(\boldsymbol{z}; \boldsymbol{\theta}) \qquad \left(\boldsymbol{s} \in \mathbb{R}^d\right)$$

$$\text{Inference: } \boldsymbol{x}' \sim \mathrm{Bernoulli}(\boldsymbol{s})$$

**Loss:**

$$-L(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\phi}) = \mathrm{BCE}(\boldsymbol{x}, \boldsymbol{s}) + \frac{1}{2}\|\boldsymbol{y}_1\|^2 + \frac{1}{2}\|e^{\boldsymbol{y}_2}\|^2 - \sum_j y_{2,j}$$

# Some Remarks

- **The loss is usually split into two parts**

$$-L(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi}) = \underbrace{\mathrm{BCE}(\boldsymbol{x},\boldsymbol{s})}_{\text{Reconstruction Loss}} + \underbrace{\left[\frac{1}{2}\|\boldsymbol{y}_1\|^2 + \frac{1}{2}\|e^{\boldsymbol{y}_2}\|^2 - \sum_j y_{2,j}\right]}_{\text{KL−divergence loss}}$$

**Why this name? Observe**

$$-L = -\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\log\left(\frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z})}\right)$$

$$= \mathrm{BCE}(\boldsymbol{x},\boldsymbol{s}) + D_{KL}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\|p_{\boldsymbol{\theta}}(\boldsymbol{z})\right)$$

- **The multiple-sample loss sums over all single sample losses**
- **Further reading on VAEs (https://arxiv.org/abs/1312.6114)**

# Demo: Variational Autoencoder

# Summary

**In this lecture, we introduced a useful class of generative models known as variational autoencoders**

**Some key take-aways**

- Unlike AEs, VAEs can learn “continuous” representations in latent space, giving rise to useful generative models
- VAEs also belongs to a powerful class of generative models making use of *latent variables*