

# Deep generative models

Pierre-Alexandre Mattei



# Menu for these two last lectures

---

1. Deep latent variable models and variational auto-encoders



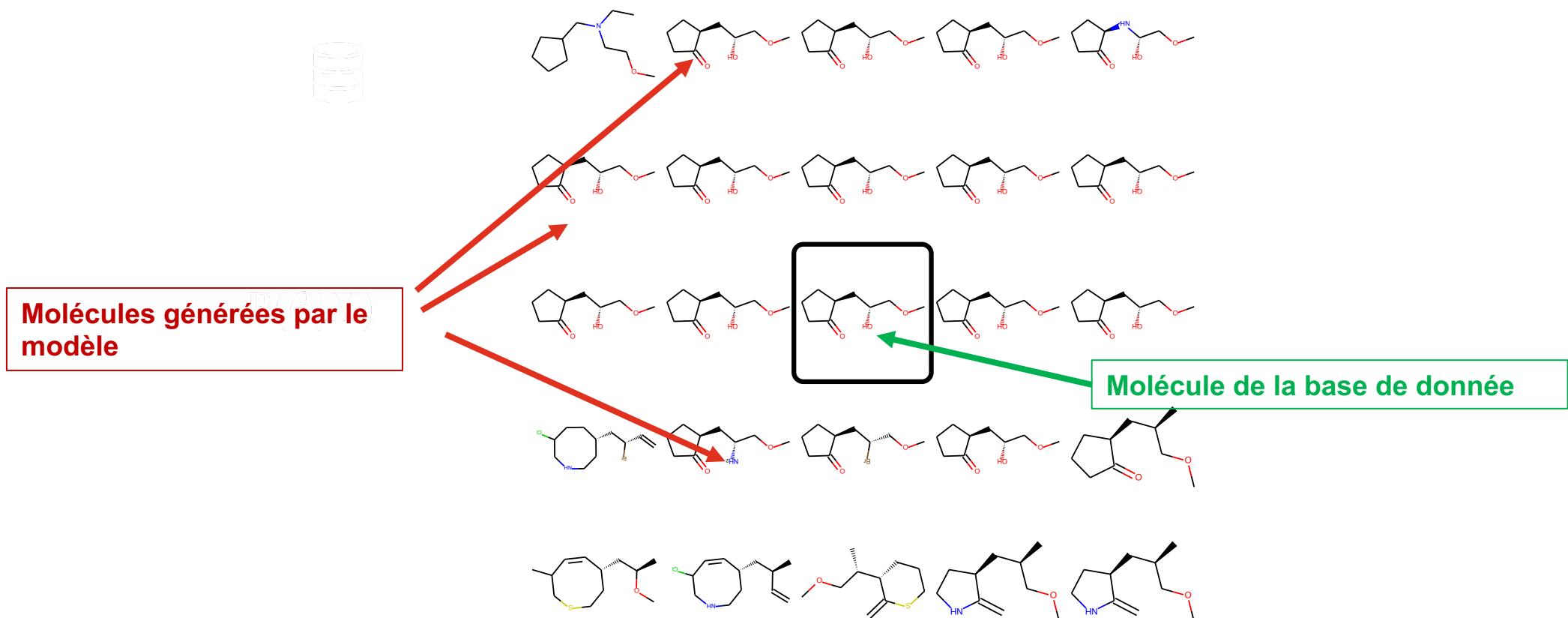
2. Diffusion models

3. Other deep generative models

# Why generative models?

---

- We want to approximate the distribution  $p_{\text{data}}(x)$  of molecules.



# Generating faces...

---



« fake » images sampled by a VAE  
Vahdat & Kautz (NeurIPS 2020)

$P(\theta|X)$

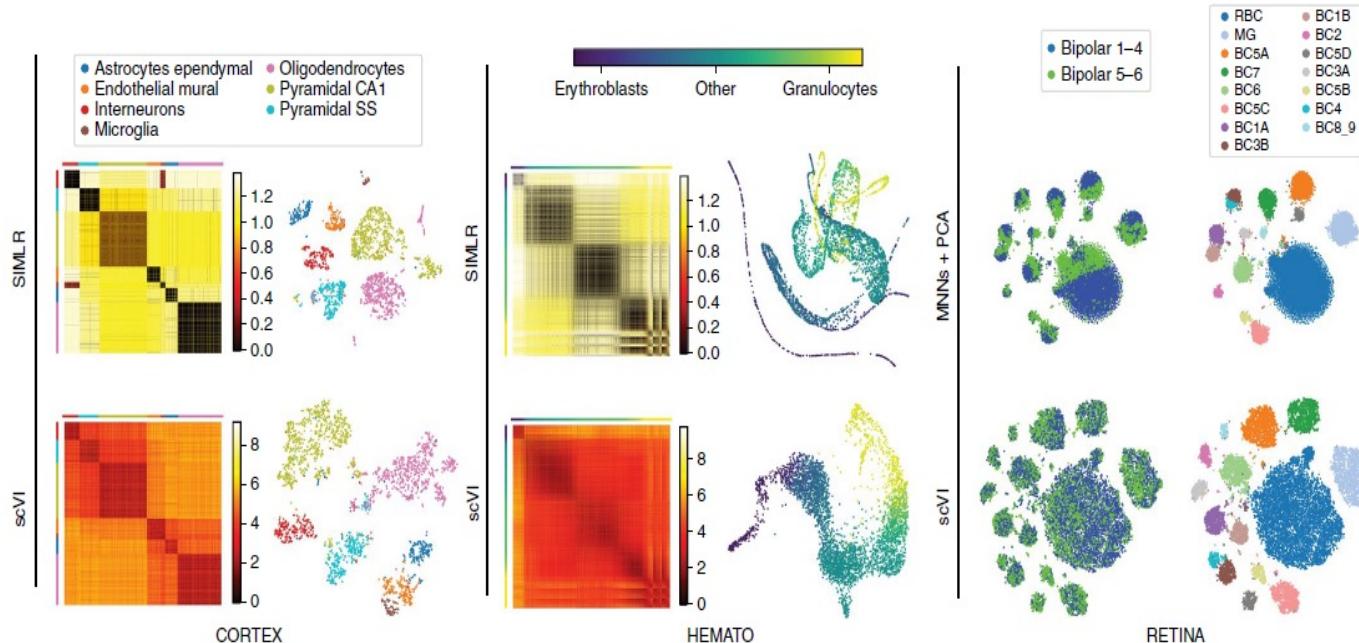
# Beyond images and molecules

---

nature**methods**

## Deep generative modeling for single-cell transcriptomics

Romain Lopez<sup>1</sup>, Jeffrey Regier<sup>ID 1</sup>, Michael B. Cole<sup>2</sup>, Michael I. Jordan<sup>1,3</sup> and Nir Yosef<sup>ID 1,4,5\*</sup>

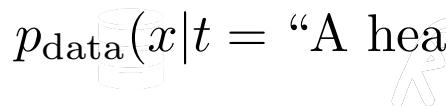


# Text to image generation

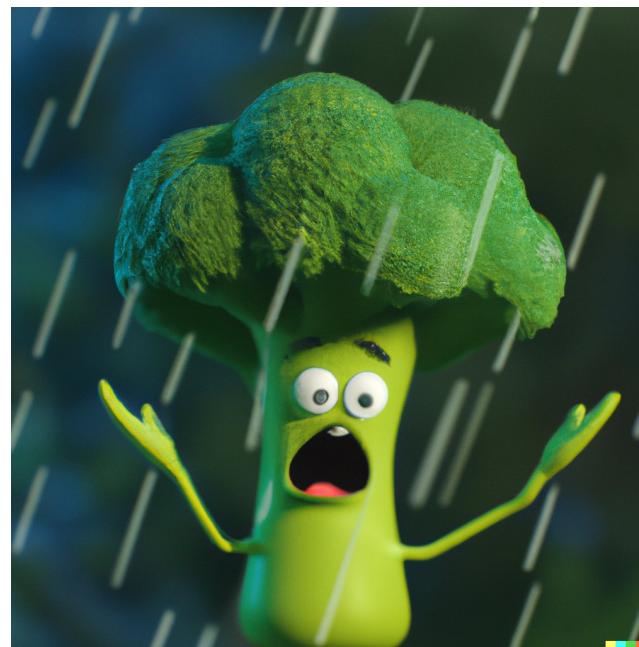
---

- The goal is here to approximate

$$p_{\text{data}}(x|t = \text{"A head of broccoli complaining about the weather"})$$

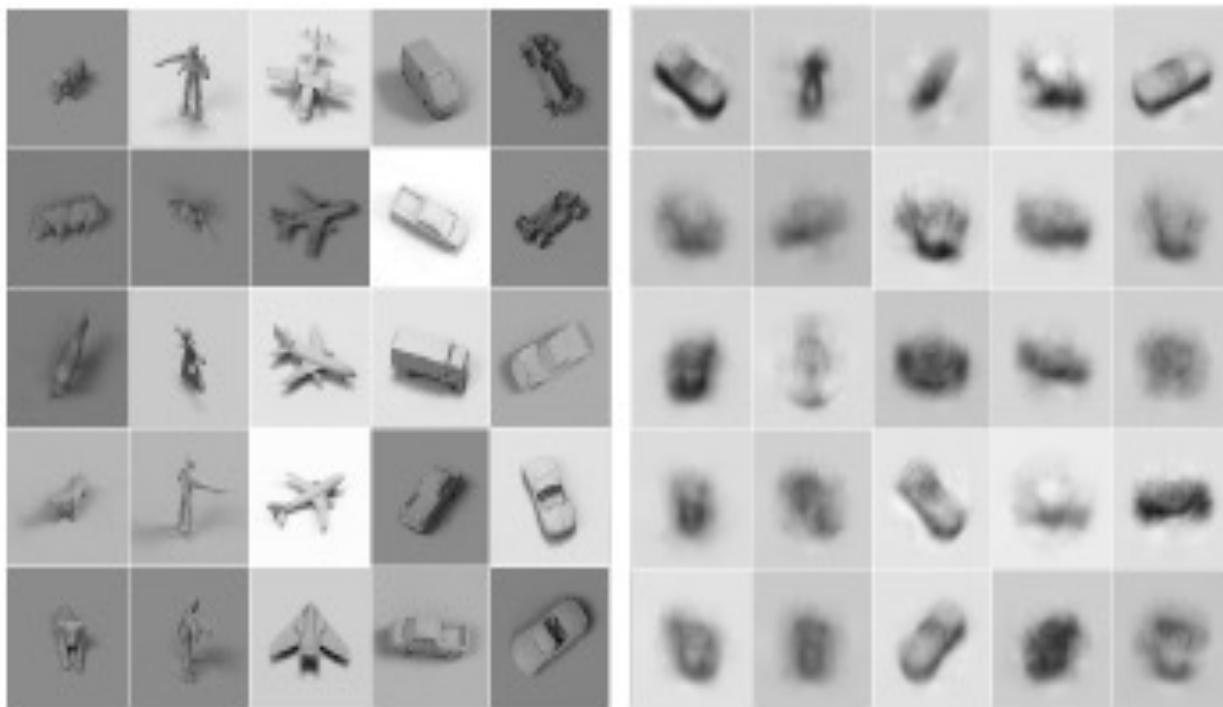


$$\mathbb{P}(\theta|X)$$



# Impressive progress

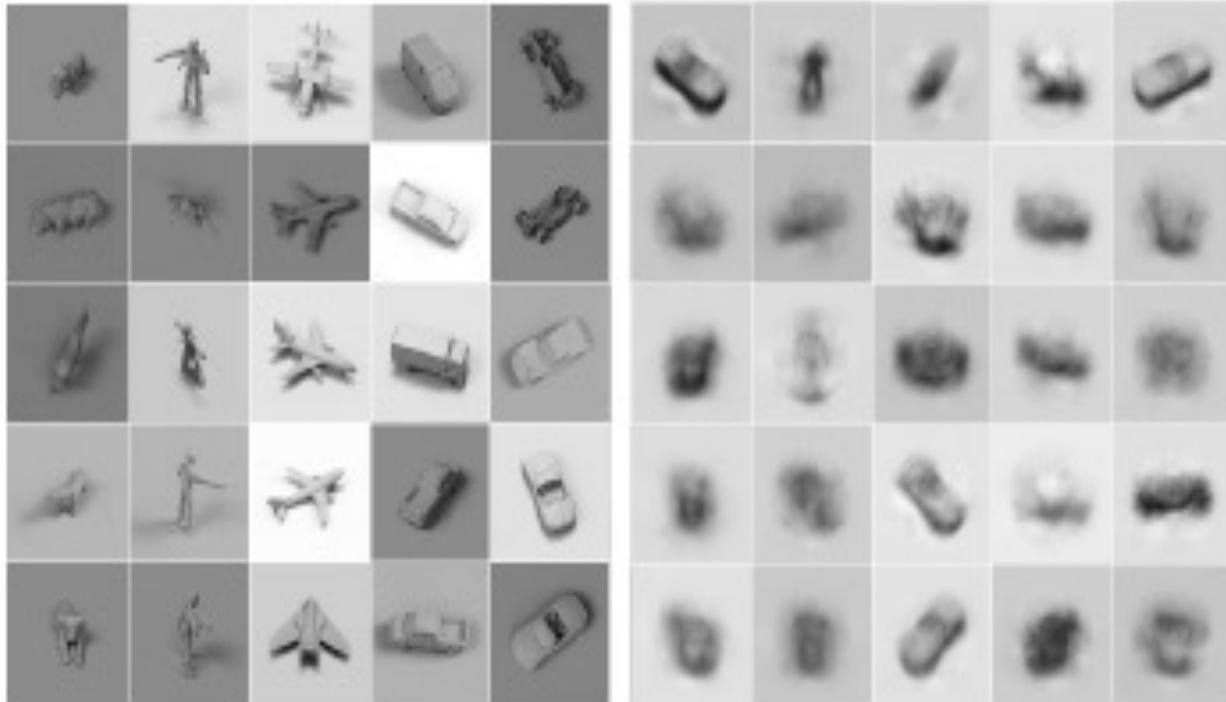
---



Rezende, Mohamed, and Wiestra (ICML 2014)

# Impressive progress

---



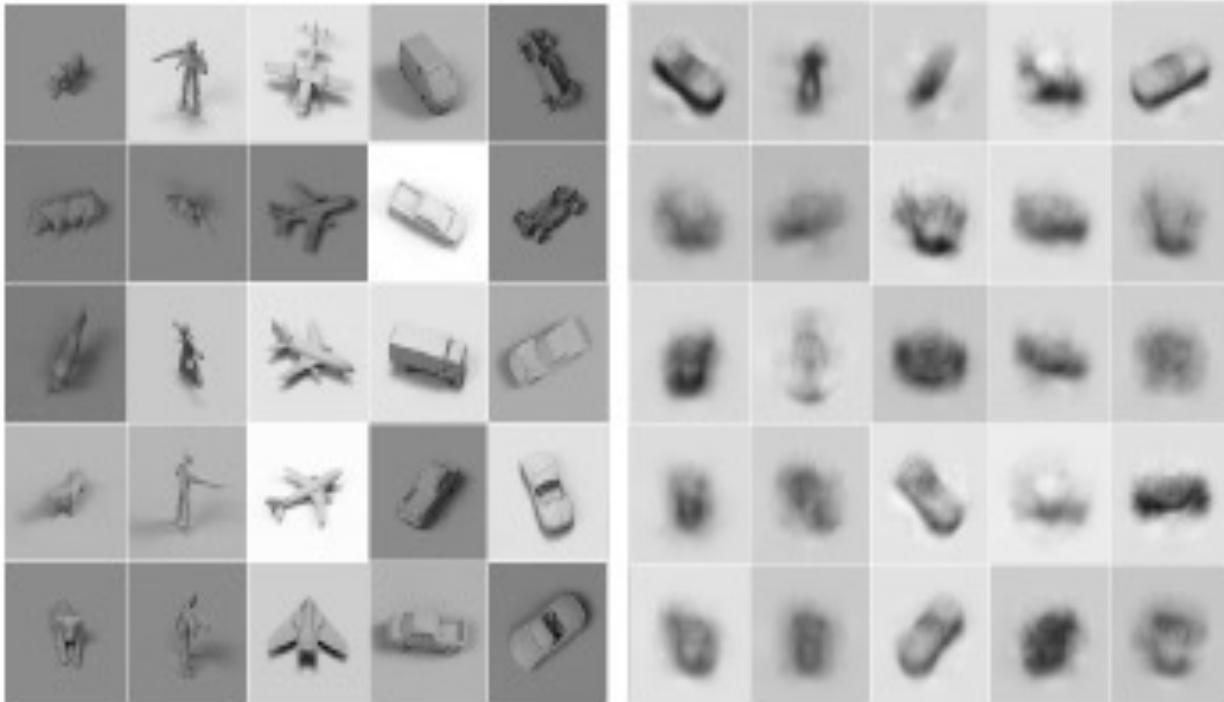
Rezende, Mohamed, and Wiestra (ICML 2014)



<https://openai.com/dall-e-2/> (2022)

# Impressive progress

---



Rezende, Mohamed, and Wiestra (ICML 2014)



<https://openai.com/dall-e-2/> (2022)



Brock, Donahue, and Simonyan (ICLR 2019)

# 1

Deep latent variable models  
VAEs and extensions

## Why latent-variable models?

---

Let's assume that we have i.i.d. data  $\mathbf{x}_1, \dots, \mathbf{x}_n$  that live in a **high-dimensional space**  $\mathcal{X}$  (for example images of newspaper articles).

Often, it is reasonable to assume that **these data essentially depend on a few important factors of variation:**

- if we have images of faces: size of nose, color of hair, glasses or not...
- if we have newspaper articles: topics of the paper, style...
- if we have molecules: physical properties, geometrical shape...

This assumption is the cornerstone of **latent variable models**, that assume that the data are governed by **unobserved random variables**  $\mathbf{z}_1, \dots, \mathbf{z}_n$  that live in a **low dimensional space**  $\mathcal{Z}$  (for example  $\mathbf{R}^2$ ). We can think of  $\mathbf{z}_i$  as a **code** summarizing the essential factors of the data point  $\mathbf{x}_i$ .

## Linear latent-variable models

---

In this slide, we assume that the data are continuous (i.e.  $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^p$ ).

**Factor analysis** is probably one of the oldest latent variable models (studied since at least the 1940s). The generative process is:

- $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_d)$ ,
- $\mathbf{x}_i | \mathbf{z}_i \sim \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \boldsymbol{\Psi})$ .

**Probabilistic PCA** (PPCA, Tipping and Bishop, JRSSB, 1999) is a slightly less general model

- $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_d)$ ,
- $\mathbf{x}_i | \mathbf{z}_i \sim \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2 \mathbf{I}_p)$ .

## Linear latent-variable models

---

In this slide, we assume that the data are continuous (i.e.  $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^p$ ).

**Factor analysis** is probably one of the oldest latent variable models (studied since at least the 1940s). The generative process is:

- $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_d)$ ,
- $\mathbf{x}_i | \mathbf{z}_i \sim \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \boldsymbol{\Psi})$ .

**Probabilistic PCA** (PPCA, Tipping and Bishop, JRSSB, 1999) is a slightly less general model

- $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_d)$ ,
- $\mathbf{x}_i | \mathbf{z}_i \sim \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2 \mathbf{I}_p)$ .

- For such models, performing maximum likelihood is not too hard:
  - EM algorithm for FA
  - Closed-form MLE for PPCA (based on the SVD of the data matrix)

# From linear (« shallow ») to deep variable models

---

How do we transform this kind of model

- $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_d)$ ,
- $\mathbf{x}_i | \mathbf{z}_i \sim \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2 \mathbf{I}_p)$ .

...into "something deep"?

?

$$\mathbb{P}(\theta|X)$$

## From linear (« shallow ») to deep variable models

---

How do we transform this kind of model

- $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_d)$ ,
- $\mathbf{x}_i | \mathbf{z}_i \sim \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2 \mathbf{I}_p)$ .

...into "something deep"?

...

We can replace the affine function  $\mathbf{z} \mapsto \mathbf{W}\mathbf{z} + \boldsymbol{\mu}$  by a neural net!

$$\mathbb{P}(\theta|X)$$

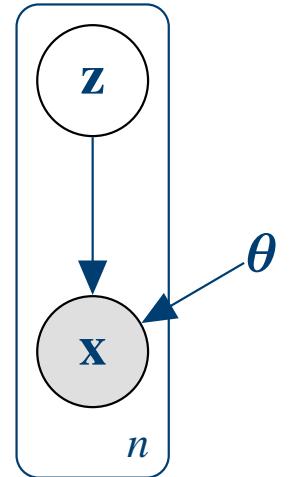
That's the key idea of **deep latent variable models (DLVMs)**, present in particular in both **variational autoencoders (VAEs)**, invented independently by Kingma and Welling (ICLR 2014) and Rezende, Mohamed & Wierstra (ICML 2014), and **generative adversarial networks (GANs)** (Goodfellow et al., NeurIPS 2014).

# Deep latent variable models

---

Assume that  $(\mathbf{x}_i, \mathbf{z}_i)_{i \leq n}$  are i.i.d. random variables driven by the model:

$$\begin{cases} \mathbf{z} \sim p(\mathbf{z}) & \text{(prior)} \\ \mathbf{x} \sim p_{\theta}(\mathbf{x} \mid \mathbf{z}) & \text{(observation model)} \end{cases}$$



where

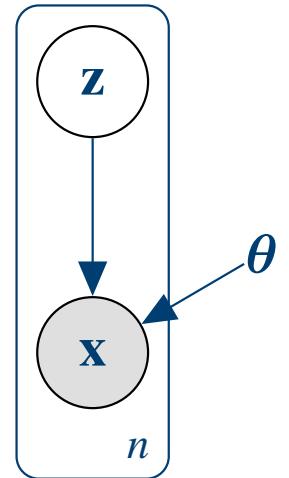
- $\mathbf{z} \in \mathbb{R}^d$  is the **latent** variable,
- $\mathbf{x} \in \mathcal{X}$  is the **observed** variable.
  - the function  $f_{\theta} : \mathbb{R}^d \rightarrow H$  is a **(deep) neural network** called the **decoder**
  - $(\Phi(\cdot \mid \eta))_{\eta \in H}$  is a parametric family called the **observation model**, usually **very simple**: unimodal and fully factorised (e.g. multivariate Gaussians or products of multinomials)

# Deep latent variable models

---

Assume that  $(\mathbf{x}_i, \mathbf{z}_i)_{i \leq n}$  are i.i.d. random variables driven by the model:

$$\begin{cases} \mathbf{z} \sim p(\mathbf{z}) & \text{(prior)} \\ \mathbf{x} \sim p_{\theta}(\mathbf{x} \mid \mathbf{z}) = \Phi(\mathbf{x} \mid f_{\theta}(\mathbf{z})) & \text{(observation model)} \end{cases}$$



where

- $\mathbf{z} \in \mathbb{R}^d$  is the **latent** variable,
- $\mathbf{x} \in \mathcal{X}$  is the **observed** variable.
  - the function  $f_{\theta} : \mathbb{R}^d \rightarrow H$  is a **(deep) neural network** called the **decoder**
  - $(\Phi(\cdot \mid \eta))_{\eta \in H}$  is a parametric family called the **observation model**, usually **very simple**: unimodal and fully factorised (e.g. multivariate Gaussians or products of multinomials)

## Deep latent variable models: the role of the prior

---

As in regular factor analysis, the prior distribution of the latent variable is often an **isotropic Gaussian**  $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}_d, \mathbf{I}_d)$ .

Note that **this prior is not a prior in the Bayesian sense** (i.e., about parameter uncertainty).

$$P(\theta|X)$$

# Deep latent variable models: the role of the observation model

---

The observation model  $(\Phi(\cdot \mid \eta))_{\eta \in H}$  usually **very simple**: unimodal and fully factorised (e.g. multivariate Gaussians or products of multinomials)

**Its parameters are the output of the decoder.**

$$\begin{cases} \mathbf{z} \sim p(\mathbf{z}) & \text{(prior)} \\ \mathbf{x} \sim p_{\theta}(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_{\theta}(\mathbf{z}), \boldsymbol{\Sigma}_{\theta}(\mathbf{z})) & \text{(Gaussian observation model)} \end{cases}$$

$$\begin{cases} \mathbf{z} \sim p(\mathbf{z}) & \text{(prior)} \\ \mathbf{x} \sim p_{\theta}(\mathbf{x} \mid \mathbf{z}) = \mathcal{B}(\mathbf{x} \mid \boldsymbol{\pi}_{\theta}(\mathbf{z})) & \text{(Bernoulli observation model)} \end{cases}$$

$$\begin{cases} \mathbf{z} \sim p(\mathbf{z}) & \text{(prior)} \\ \mathbf{x} \sim p_{\theta}(\mathbf{x} \mid \mathbf{z}) = \text{St}(\mathbf{x} \mid \boldsymbol{\mu}_{\theta}(\mathbf{z}), \boldsymbol{\Sigma}_{\theta}(\mathbf{z}), \boldsymbol{\nu}_{\theta}(\mathbf{z})) & \text{(Student's t observation model)} \end{cases}$$

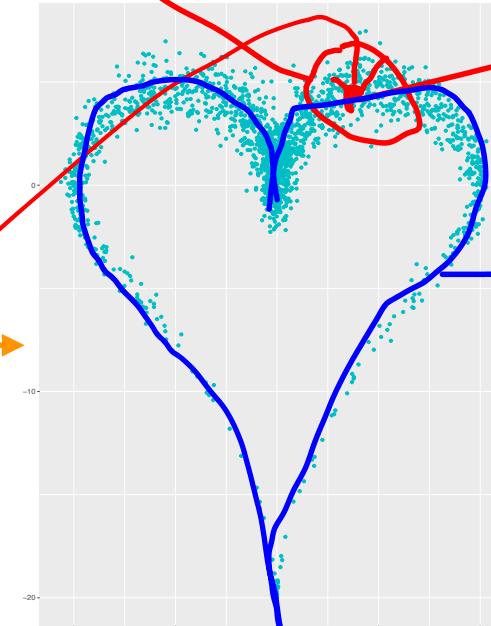
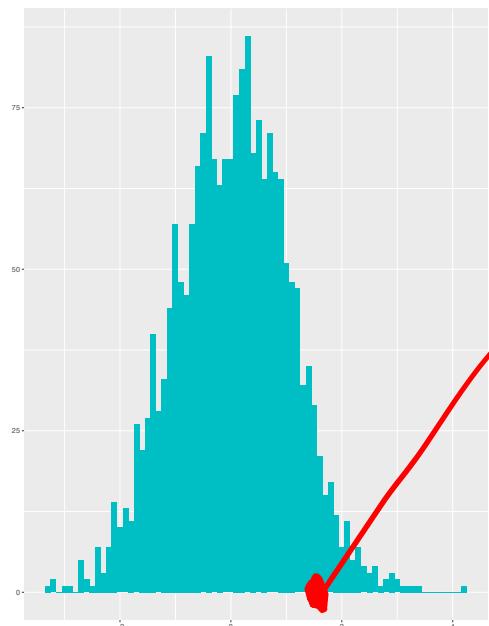
# Deep latent variable models: the role of the decoder

The role of the **decoder**  $f_{\theta} : \mathbb{R}^d \rightarrow H$  is:

- to transform  $\mathbf{z}$  (**the code**) into parameters  $\eta = f_{\theta}(\mathbf{z})$  of the observation model  $\Phi(\cdot | \eta)$ .
- The weights  $\theta$  of the **decoder** are learned.

Simple non-linear decoder ( $d = 1, p = 2$ ):  $f_{\theta}(z) = \mu_{\theta}(\mathbf{z}), \Sigma_{\theta}(\mathbf{z})$  with, for all  $z \in \mathbb{R}$ ,

$$\mu_{\theta}(z) = (10 \sin(z)^3, 10 \cos(z) - 10 \cos(z)^4), \Sigma_{\theta}(\mathbf{z}) = \text{Diag} \left( \left( \frac{\sin(z)}{3z} \right)^2, \left( \frac{\sin(z)}{z} \right)^2 \right).$$



$\{p(z)\}_{z \in \mathbb{R}}$

## Illustrative example of a DLVM for binary images

---

**Training data**  $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$  binary MNIST



# Illustrative example of a DLVM for binary images

---

**Generative model** for  $\mathbf{z} \in \mathbb{R}^2$  and  $\mathbf{x} \in \{0, 1\}^{28 \times 28}$

$$\begin{cases} \mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\mathbf{z})) \end{cases}$$

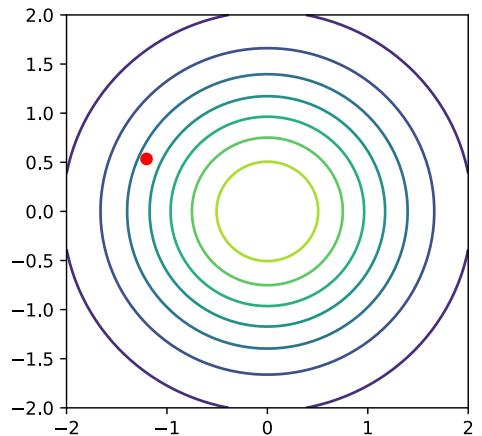
**Decoder network**

$$f(\mathbf{z}) = \text{Sigmoid}(\mathbf{V} \tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \boldsymbol{\beta})$$

$$\mathbb{P}(\theta | X)$$

**Generation**

$$\mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)$$
$$\mathbf{z} = (-1.2033, 0.5340)$$



# Illustrative example of a DLVM for binary images

---

**Generative model** for  $\mathbf{z} \in \mathbb{R}^2$  and  $\mathbf{x} \in \{0, 1\}^{28 \times 28}$

$$\begin{cases} \mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\mathbf{z})) \end{cases}$$

**Decoder network**

$$f(\mathbf{z}) = \text{Sigmoid}(\mathbf{V} \tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \boldsymbol{\beta})$$

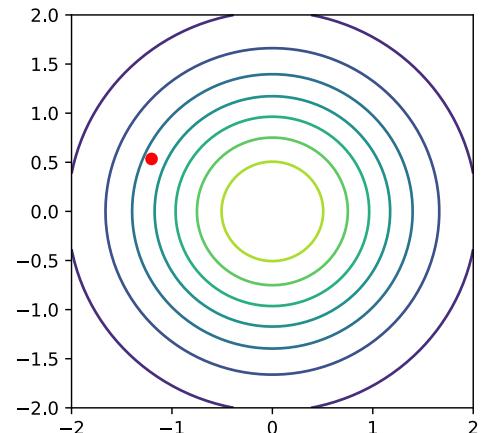
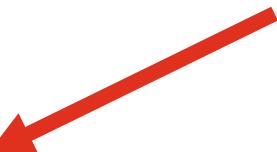
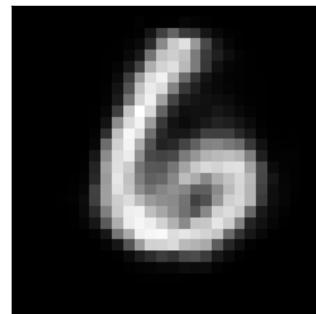
$$\mathbb{P}(\theta | X)$$

## Generation

$$\mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)$$

$$\mathbf{z} = (-1.2033, 0.5340)$$

$$f(\mathbf{z})$$



# Illustrative example of a DLVM for binary images

---

**Generative model** for  $\mathbf{z} \in \mathbb{R}^2$  and  $\mathbf{x} \in \{0, 1\}^{28 \times 28}$

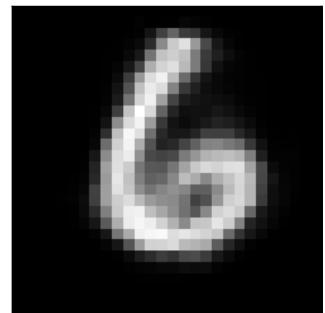
$$\begin{cases} \mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\mathbf{z})) \end{cases}$$

**Decoder network**

$$f(\mathbf{z}) = \text{Sigmoid}(\mathbf{V} \tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \boldsymbol{\beta})$$

$$\mathbb{P}(\theta|X)$$

$$f(\mathbf{z})$$

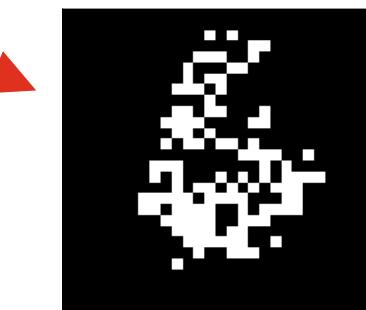
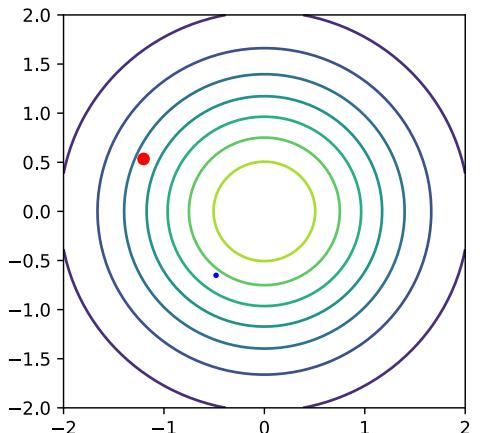


$$x^{j,k} \sim \text{Bern}(f^{j,k}(\mathbf{z}))$$

## Generation

$$\mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)$$

$$\mathbf{z} = (-1.2033, 0.5340)$$



# Illustrative example of a DLVM for binary images

---

## Generation

Generative model for  $\mathbf{z} \in \mathbb{R}^2$  and  $\mathbf{x} \in \{0, 1\}^{28 \times 28}$

$$\begin{cases} \mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\mathbf{z})) \end{cases}$$

$$\mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)$$

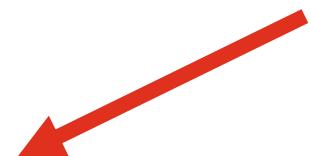
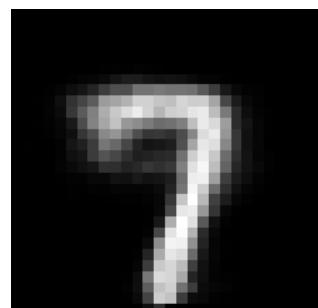
$$\mathbf{z} = (0.0791, -1.8165)$$

## Decoder network

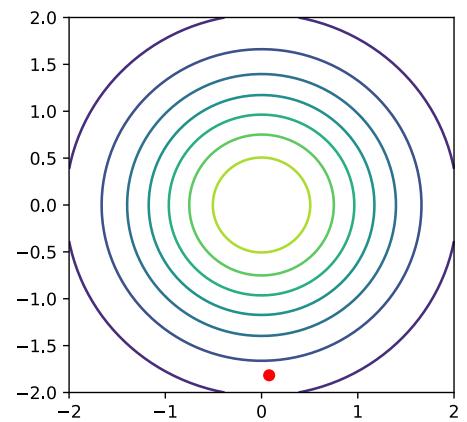
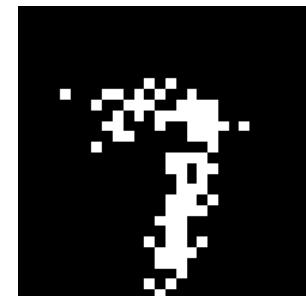
$$f(\mathbf{z}) = \text{Sigmoid}(\mathbf{V} \tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \boldsymbol{\beta})$$

$$\mathbb{P}(\theta|X)$$

$$f(z)$$



$$x^{j,k} \sim \text{Bern}(f^{j,k}(z))$$



## DLVM from one of the seminal VAE papers (Rezende et al., ICML'14)

---

0	2	0	3	8	6	7	3	8	8
9	0	5	5	0	9	7	6	4	8
4	6	3	2	4	1	7	1	7	7
5	1	8	4	8	6	6	5	4	9
3	3	0	6	1	3	2	6	2	3
6	4	5	0	1	1	4	5	8	1
7	8	3	7	9	7	1	6	7	9
0	0	4	7	3	3	1	3	2	1
3	3	9	3	6	9	8	7	8	6
2	4	8	4	9	5	1	6	8	8

Training data

6	6	9	0	5	7	0	8	0	9
3	5	7	6	6	9	1	1	3	0
4	4	4	6	5	4	0	6	4	9
2	9	7	7	0	9	0	9	4	8
6	5	4	0	0	9	9	3	2	3
3	9	5	6	1	5	0	7	7	6
5	6	2	9	7	6	9	4	0	9
2	3	1	3	4	1	5	0	4	0
1	2	5	7	6	9	9	5	3	7
6	4	3	3	7	9	0	9	4	3

Model samples

## DLVM for missing data imputation (Mattei & Frellsen, ICML'19)

---



2 2 0 7 0 8 9 8 8 7 8 1

P	3	3	9	5	5	5	5	3	3	3	3
E	6	6	6	6	8	8	8	8	8	8	8
I	1	1	1	1	1	1	1	1	1	1	1

## Maximum likelihood for DLVM

---

Given a data matrix  $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)^\top \in \mathcal{X}^n$ , the **log-likelihood function** for a DLVM is

$$\ell(\boldsymbol{\theta}) = \log p_{\boldsymbol{\theta}}(\mathbf{X}) = \sum_{i=1}^n \log p_{\boldsymbol{\theta}}(\mathbf{x}_i),$$

where

$$p_{\boldsymbol{\theta}}(\mathbf{x}_i) = \int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}) \mathbf{z}.$$

We would like to find a **MLE**  $\hat{\boldsymbol{\theta}} \in \operatorname{argmax}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})$ .

$$\mathbb{E}(\hat{\boldsymbol{\theta}} \mid \mathcal{D})$$

## Maximum likelihood for DLVM

---

Given a data matrix  $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)^\top \in \mathcal{X}^n$ , the **log-likelihood function** for a DLVM is

$$\ell(\boldsymbol{\theta}) = \log p_{\boldsymbol{\theta}}(\mathbf{X}) = \sum_{i=1}^n \log p_{\boldsymbol{\theta}}(\mathbf{x}_i),$$

where

$$p_{\boldsymbol{\theta}}(\mathbf{x}_i) = \int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}) \mathbf{z}.$$

We would like to find a **MLE**  $\hat{\boldsymbol{\theta}} \in \operatorname{argmax}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})$ .

However, even with a simple output density  $p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})$ :

- $p_{\boldsymbol{\theta}}(\mathbf{x})$  is **intractable** rendering **MLE intractable**
- $p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})$  is **intractable** rendering **EM intractable**
- **stochastic EM is not scalable** to large  $n$  and moderate  $d$ .

## Maximum likelihood for DLVMs via importance sampling

---

A general and scalable framework to tackle these issues was proposed by Kingma & Welling (2014), Rezende et al. (2014), leading to the **variational autoencoder (VAE)**.

Here, **I am going to derive this approach in a slightly different manner**, largely inspired by the following paper:

-  Burda, Grosse & Salakhutdinov (2016), *Importance weighted autoencoders*, ICLR 2016

The main idea is to use **Monte Carlo techniques** to approximate the intractable integrals

$$p_{\boldsymbol{\theta}}(\mathbf{x}_i) = \int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}) d\mathbf{z}.$$

## Beyond simple MC: importance sampling

---

- Short Monte Carlo course! Our goal is to approximate an integral



$$I = \int_{\Omega} f(x)p(x)dx$$

- We already saw the **simple MC estimate**

$$\mathbb{P}(\theta|X)$$

$$I \approx \frac{1}{K} \sum_{k=1}^K f(x_k) = \hat{I}_K.$$

## Beyond simple MC: importance sampling

---

- Short Monte Carlo course! Our goal is to approximate an integral



$$\hat{I} = \int_{\Omega} f(x)p(x)dx$$

- We already saw the **simple MC estimate**

$P(\theta|X)$

$$I \approx \frac{1}{K} \sum_{k=1}^K f(x_k) = \hat{I}_K.$$

- This estimate has nice properties:

- Unbiasedness:
- Consistency:  $\mathbb{E}[\hat{I}_K] = I$
- Asymptotic normality:  $\hat{I}_K \xrightarrow{a.s.} I$

## Beyond simple MC: importance sampling

---

- One way of assessing the accuracy of any unbiased estimate is by looking at its variance.  
**The lower the variance of an unbiased estimate, the better.**
- If the samples are iid, then the variance of simple MC will be  $\mathbb{V}[\hat{I}_K] = \frac{1}{K} \mathbb{V}[f(x_1)]$

$$\mathbb{P}(\theta|X)$$

## Beyond simple MC: importance sampling

---

- One way of assessing the accuracy of any unbiased estimate is by looking at its variance.  
**The lower the variance of an unbiased estimate, the better.**
- If the samples are iid, then the variance of simple MC will be  $\mathbb{V}[\hat{I}_K] = \frac{1}{K} \mathbb{V}[f(x_1)]$
- So the variance gets smaller and smaller at speed  $1/K$ , that's good news!
- But it can still be pretty big, depending on the value of  $\mathbb{V}[f(x_1)]$   
*( $f(x_1)$ )*
- **Can we reduce the variance of the MC estimate?**

## Beyond simple MC: importance sampling

---

- Key idea: rather than sampling from  $p(x)$ , we're going to sample from another density  $q(x)$  that we'll call a **proposal**



$$x_1, \dots, x_K \sim q$$

- But the integral is an expected value with respect to  $p$ . **Can we turn an expected value with respect to  $p$  into an expected value with respect to  $q$ ?**

$$\mathbb{P}(\theta|X)$$

## Beyond simple MC: importance sampling

---

- Key idea: rather than sampling from  $p(x)$ , we're going to sample from another density  $q(x)$  that we'll call a **proposal**



$$x_1, \dots, x_K \sim q$$

- But the integral is an expected value with respect to  $p$ . **Can we turn an expected value with respect to  $p$  into an expected value with respect to  $q$ ? Yes!**

POLYU

$$\begin{aligned} I &= \int_{\Omega} f(x)p(x)dx \\ &= \int_{\Omega} \frac{f(x)p(x)}{q(x)} q(x)dx \approx \frac{1}{K} \sum_{k=1}^K \frac{f(x_k)p(x_k)}{q(x_k)} = \hat{I}_K^q. \end{aligned}$$

## Beyond simple MC: importance sampling

---

- This new estimate  $\hat{I}_K^q$  is called an **importance sampling** estimate.
- Now, is  $\hat{I}_K^q$  any better than the simple MC estimate?



$$\mathbb{P}(\theta|X)$$

## Beyond simple MC: importance sampling

---

- This new estimate  $\hat{I}_K^q$  is called an **importance sampling** estimate.
- Now, is  $\hat{I}_K^q$  any better than the simple MC estimate? Of course, this depends of the choice of the proposal  $q$ ...



- It's clear that  $\hat{I}_K^q$  will also be unbiased, consistent, and asymptotically normal.
- The variance will be  $\mathbb{V}_{x \sim q}[f(x)p(x)/q(x)]/K$

$$\mathbb{P}(\theta|X)$$

## Beyond simple MC: importance sampling

---

- This new estimate  $\hat{I}_K^q$  is called an **importance sampling** estimate.
- Now, is  $\hat{I}_K^q$  any better than the simple MC estimate? Of course, this depends of the choice of the proposal  $q$ ...



- It's clear that  $\hat{I}_K^q$  will also be unbiased, consistent, and asymptotically normal.
- The variance will be  $\mathbb{V}_{x \sim q}[f(x)p(x)/q(x)]/K$
- For  $q^*(x) \propto f(x)p(x)$ , **the variance will be exactly zero!**
- **That seems a bit too good to be true... What's the catch?**

## The optimal importance sampling proposal

---

- For  $q^*(x) \propto f(x)p(x)$ , **the variance will be exactly zero!**
- **That seems a bit too good to be true... What's the catch?**

$$q^*(x) = \frac{f(x)p(x)}{\int f(x)p(x)dx} = \frac{f(x)p(x)}{I}$$

... and  $I$  is precisely the thing we want to compute!

# The optimal importance sampling proposal

---

- For  $q^*(x) \propto f(x)p(x)$ , **the variance will be exactly zero!**
- **That seems a bit too good to be true... What's the catch?**



$$q^*(x) = \frac{f(x)p(x)}{\int f(x)p(x)dx} = \frac{f(x)p(x)}{I}$$

... and  $I$  is precisely the thing we want to compute!

- In practice, we won't be able to find this optimal proposal, but this shows that **the improvements of importance sampling can be potentially huge!**
- This simple result is therefore a motivation for looking for good proposals.

## Maximum likelihood for DLVM

---

We want to approximate

$$p_{\theta}(\mathbf{x}_i) = \int_{\mathbb{R}^d} p_{\theta}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}) d\mathbf{z}.$$

**Idea: use importance sampling!** Let  $\mathbf{z}_{i1}, \dots, \mathbf{z}_{iK}$  follow some proposal  $q_i$ :

$$\int_{\mathbb{R}^d} p_{\theta}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}_{ik}) d\mathbf{z} \approx \frac{1}{K} \sum_{k=1}^K \frac{p_{\theta}(\mathbf{x}_i \mid \mathbf{z}_{ik}) p(\mathbf{z}_{ik})}{q_i(\mathbf{z}_{ik})}$$

Let's say that we want to choose our **proposal in a parametric family**  $(\Psi(\cdot | \kappa))_{\kappa \in \mathcal{K}}$  over  $\mathbb{R}^d$  (e.g. Gaussians).

# Maximum likelihood for DLVM

---

We want to approximate

$$p_{\theta}(\mathbf{x}_i) = \int_{\mathbb{R}^d} p_{\theta}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}) d\mathbf{z}.$$

**Idea: use importance sampling!** Let  $\mathbf{z}_{i1}, \dots, \mathbf{z}_{iK}$  follow some proposal  $q_i$ :

$$\int_{\mathbb{R}^d} p_{\theta}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}_{ik}) d\mathbf{z} \approx \frac{1}{K} \sum_{k=1}^K \frac{p_{\theta}(\mathbf{x}_i \mid \mathbf{z}_{ik}) p(\mathbf{z}_{ik})}{q_i(\mathbf{z}_{ik})}$$

Let's say that we want to choose our **proposal in a parametric family**  $(\Psi(\cdot | \kappa))_{\kappa \in \mathcal{K}}$  over  $\mathbb{R}^d$  (e.g. Gaussians).

**Problem:** we need to choose  **$n$  proposals**  $q_1, \dots, q_n$  (and  $n$  is usually large in deep learning...).

## Maximum likelihood for DLVMs: choosing proposals

---

$$\int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}_{ik}) d\mathbf{z} \approx \frac{1}{K} \sum_{k=1}^K \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}_{ik}) p(\mathbf{z})}{q_i(\mathbf{z}_{ik})}$$



What would be the optimal, zero-variance choices for  $q_1, \dots, q_n$ ?

$$\mathbb{P}(\boldsymbol{\theta} | \mathcal{X})$$

## Maximum likelihood for DLVMs: choosing proposals

---

$$\int_{\mathbb{R}^d} p_{\theta}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}_{ik}) d\mathbf{z} \approx \frac{1}{K} \sum_{k=1}^K \frac{p_{\theta}(\mathbf{x}_i \mid \mathbf{z}_{ik}) p(\mathbf{z})}{q_i(\mathbf{z}_{ik})}$$



What would be the optimal, zero-variance choices for  $q_1, \dots, q_n$ ?

$$P(\theta|X)$$

$$q_i(z) = p(z|x_i)$$

## Maximum likelihood for DLVM

---

**A solution:** Amortised variational inference, **all the  $q_i$  will be defined together via a neural net!**

**Rationale:**  $q_i$  needs to depends on  $\mathbf{x}_i$ , so we'll define it as a **conditional distribution parametrised by  $\gamma$** :

$$q_i(\mathbf{z}) = q_{\gamma}(\mathbf{z}|\mathbf{x}_i).$$

How to parametrise this conditional distribution? The key idea is that **its parameters are the output of a neural net  $g_{\gamma}$** :

$$q_{\gamma}(\mathbf{z}|\mathbf{x}_i) = \Psi(\mathbf{z}|g_{\gamma}(\mathbf{x}_i)).$$

This neural net is called the **inference network** or **encoder**.

## Maximum likelihood for DLVM

---

All of this leads to the following approximation of the likelihood

$$\ell(\boldsymbol{\theta}) \approx \sum_{i=1}^n \mathbb{E}_{\mathbf{z}_{i1}, \dots, \mathbf{z}_{iK} \sim q_{\boldsymbol{\gamma}}(\mathbf{z}|\mathbf{x}_i)} \left[ \log \frac{1}{K} \sum_{k=1}^K \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_i | \mathbf{z}_{ik}) p(\mathbf{z}_{ik})}{q_{\boldsymbol{\gamma}}(\mathbf{z}_{ik} | \mathbf{x}_i)} \right] = \mathcal{L}_K(\boldsymbol{\theta}, \boldsymbol{\gamma}).$$

Rather than maximising  $\ell(\boldsymbol{\theta})$ , we'll maximise  $\mathcal{L}_K(\boldsymbol{\theta}, \boldsymbol{\gamma})$  using SGD and the reparametrisation trick. But does it make sense to do that?

$$\mathbb{P}(\boldsymbol{\theta}|X)$$

# Maximum likelihood for DLVM

---

All of this leads to the following approximation of the likelihood

$$\ell(\boldsymbol{\theta}) \approx \sum_{i=1}^n \mathbb{E}_{\mathbf{z}_{i1}, \dots, \mathbf{z}_{iK} \sim q_{\boldsymbol{\gamma}}(\mathbf{z}|\mathbf{x}_i)} \left[ \log \frac{1}{K} \sum_{k=1}^K \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_i | \mathbf{z}_{ik}) p(\mathbf{z}_{ik})}{q_{\boldsymbol{\gamma}}(\mathbf{z}_{ik} | \mathbf{x}_i)} \right] = \mathcal{L}_K(\boldsymbol{\theta}, \boldsymbol{\gamma}).$$

Rather than maximising  $\ell(\boldsymbol{\theta})$ , we'll maximise  $\mathcal{L}_K(\boldsymbol{\theta}, \boldsymbol{\gamma})$  using SGD and the reparametrisation trick. But does it make sense to do that?

**It does make sense!** For several reasons:

- $\mathcal{L}_K(\boldsymbol{\theta}, \boldsymbol{\gamma})$  is a **lower bound of  $\ell(\boldsymbol{\theta})$**  (exercise !)
- The bounds get **tighter and tighter!**

$$\mathcal{L}_1(\boldsymbol{\theta}, \boldsymbol{\gamma}) \leq \mathcal{L}_2(\boldsymbol{\theta}, \boldsymbol{\gamma}) \leq \dots \leq \mathcal{L}_K(\boldsymbol{\theta}, \boldsymbol{\gamma}) \xrightarrow{K \rightarrow \infty} \ell(\boldsymbol{\theta}).$$

$\mathcal{L}_K(\boldsymbol{\theta}, \boldsymbol{\gamma})$  is called the **importance weighted autoencoder (IWAE) bound**, and was introduced by Burda et al. (2016).

## What about VAEs?

---

The VAE bound of Kingma & Welling (2014) and Rezende et al. (2014) is actually  $\mathcal{L}_1(\theta, \gamma)$ , which is the loosest bound!

The VAE bound can be interestingly rewritten

$$\mathcal{L}_1(\theta, \gamma) = \ell(\theta) - \text{KL} \left( \prod_{i=1}^n q_\gamma(\mathbf{z}_i | \mathbf{x}_i) \middle\| \prod_{i=1}^n p_\theta(\mathbf{z}_i | \mathbf{x}_i) \right).$$

which means that, for a given  $\theta$ , **the optimal  $q_\gamma(\mathbf{z}_i | \mathbf{x}_i)$  will be as close as possible (in a KL sense) to the true posterior  $p_\theta(\mathbf{z}_i | \mathbf{x}_i)$ .**

Concrete consequence: after training, **we may interpret the  $q_\gamma(\mathbf{z}_i | \mathbf{x}_i)$  as an (approachable) approximation of the (intractable)  $p_\theta(\mathbf{z}_i | \mathbf{x}_i)$ .**

## What about VAEs?

---

Is it still true when  $K > 1$ ? Kind of, but it gets more complicated.  
Domke & Sheldon (2019) showed that, when  $K \rightarrow \infty$ , the "closeness" is no longer in KL sense but in the sense of the  $\chi$  divergence.

$$\mathbb{P}(\theta|X)$$

## What about VAEs?

---

The VAE bound of Kingma & Welling (2014) and Rezende et al. (2014) is actually  $\mathcal{L}_1(\theta, \gamma)$ , which is the loosest bound!

The VAE bound can be interestingly rewritten

$$\mathcal{L}_1(\theta, \gamma) = \ell(\theta) - \text{KL} \left( \prod_{i=1}^n q_\gamma(\mathbf{z}_i | \mathbf{x}_i) \middle\| \prod_{i=1}^n p_\theta(\mathbf{z}_i | \mathbf{x}_i) \right).$$

which means that, for a given  $\theta$ , **the optimal  $q_\gamma(\mathbf{z}_i | \mathbf{x}_i)$  will be as close as possible (in a KL sense) to the true posterior  $p_\theta(\mathbf{z}_i | \mathbf{x}_i)$ .**

Concrete consequence: after training, **we may interpret the  $q_\gamma(\mathbf{z}_i | \mathbf{x}_i)$  as an (approachable) approximation of the (intractable)  $p_\theta(\mathbf{z}_i | \mathbf{x}_i)$ .**

## What about VAEs?

---

The VAE bound can be also rewritten

$$\mathcal{L}_1(\theta, \gamma) = \mathbb{E}_{\mathbf{z}_i \sim q_{\gamma}(\mathbf{z}_i | \mathbf{x}_i)} [p_{\theta}(\mathbf{x}_i | \mathbf{z}_i)] - \text{KL} \left( \prod_{i=1}^n q_{\gamma}(\mathbf{z}_i | \mathbf{x}_i) \middle\| \prod_{i=1}^n p_{\theta}(\mathbf{z}_i) \right).$$

- $\mathbb{E}_{\mathbf{z}_i \sim q_{\gamma}(\mathbf{z}_i | \mathbf{x}_i)} [p_{\theta}(\mathbf{x}_i | \mathbf{z}_i)]$  can be interpreted as (the opposite of) a **reconstruction error**.
- the **KL between the approximate posterior and the prior** can be interpreted as a regulariser. If  $q_{\gamma}(\mathbf{z} | \mathbf{x})$  is Gaussian, then this can be computed in **closed-form**.

This version of the bound resembles the opposite of the loss of a **KL-regularised autoencoder**, hence the name **variational autoencoder (VAE)**, coined by Kingma and Welling (ICLR 2014).

## What about VAEs?

---

- The VAE bound can also be rewritten

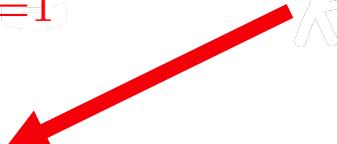
$$\mathcal{L}_1(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \sum_{i=1}^n \mathbb{E}_{\mathbf{z}_i \sim q_{\boldsymbol{\gamma}}(\mathbf{z}|\mathbf{x}_i)} [\log p_{\boldsymbol{\theta}}(\mathbf{x}_i|\mathbf{z}_i)] - \text{KL} \left( \prod_{i=1}^n q_{\boldsymbol{\gamma}}(\mathbf{z}_i|\mathbf{x}_i) \middle\| \prod_{i=1}^n p_{\boldsymbol{\theta}}(\mathbf{z}_i) \right).$$

$\mathbb{P}(\theta|X)$

## What about VAEs?

---

- The VAE bound can also be rewritten

$$\mathcal{L}_1(\theta, \gamma) = \sum_{i=1}^n \mathbb{E}_{\mathbf{z}_i \sim q_\gamma(\mathbf{z}|\mathbf{x}_i)} [\log p_\theta(\mathbf{x}_i|\mathbf{z}_i)] - \text{KL} \left( \prod_{i=1}^n q_\gamma(\mathbf{z}_i|\mathbf{x}_i) \middle\| \prod_{i=1}^n p_\theta(\mathbf{z}_i) \right).$$


**Reconstruction error:**

mismatch between  $\mathbf{x}_i$  and its reconstruction (obtained by auto-encoding it)

## What about VAEs?

---

- The VAE bound can also be rewritten

$$\mathcal{L}_1(\theta, \gamma) = \sum_{i=1}^n \mathbb{E}_{\mathbf{z}_i \sim q_\gamma(\mathbf{z}|\mathbf{x}_i)} [\log p_\theta(\mathbf{x}_i|\mathbf{z}_i)] - \text{KL} \left( \prod_{i=1}^n q_\gamma(\mathbf{z}_i|\mathbf{x}_i) \middle\| \prod_{i=1}^n p_\theta(\mathbf{z}_i) \right).$$

**Reconstruction error:**

mismatch between  $\mathbf{x}_i$  and its reconstruction (obtained by auto-encoding it)

**KL regulariser:**

makes sure the encodings are not too far away from the prior.  
At the end of the day, a scatter plot of the encodings will kinda look like the prior.

## What about VAEs?

---

- The VAE bound can also be rewritten

$$\mathcal{L}_1(\theta, \gamma) = \sum_{i=1}^n \mathbb{E}_{\mathbf{z}_i \sim q_\gamma(\mathbf{z}|\mathbf{x}_i)} [\log p_\theta(\mathbf{x}_i|\mathbf{z}_i)] - \text{KL} \left( \prod_{i=1}^n q_\gamma(\mathbf{z}_i|\mathbf{x}_i) \middle\| \prod_{i=1}^n p_\theta(\mathbf{z}_i) \right).$$

**Reconstruction error:**

mismatch between  $\mathbf{x}_i$  and its reconstruction (obtained by auto-encoding it)

**KL regulariser:**

makes sure the encodings are not too far away from the prior.  
At the end of the day, a scatter plot of the encodings will kinda look like the prior.

## Why is this a reconstruction error?

- In the particular case of a Gaussian observation model  $p_{\theta}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{\theta}(\mathbf{z}), \sigma^2 \mathbf{I}_D)$  with a constant and isotropic covariance, we can use the formula of the Gaussian density to get



$$\mathbb{P}(\theta|X)$$

## Why is this a reconstruction error?

- In the particular case of a Gaussian observation model  $p_{\theta}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{\theta}(\mathbf{z}), \sigma^2 \mathbf{I}_D)$  with a constant and isotropic covariance, we can use the formula of the Gaussian density to get

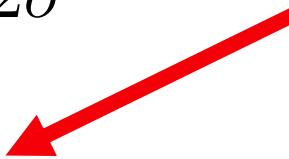
$$\log p_{\theta}(\mathbf{x}|\mathbf{z}) = -\frac{D}{2} \log(2\pi) - D \log \sigma - \frac{1}{2\sigma^2} \|\mathbf{x} - \boldsymbol{\mu}_{\theta}(\mathbf{z})\|_2^2$$

$$\mathbb{P}(\theta|X)$$

## Why is this a reconstruction error?

- In the particular case of a Gaussian observation model  $p_{\theta}(x|z) = \mathcal{N}(x|\mu_{\theta}(z), \sigma^2 I_D)$  with a constant and isotropic covariance, we can use the formula of the Gaussian density to get

$$\log p_{\theta}(x|z) = -\frac{D}{2} \log(2\pi) - D \log \sigma - \frac{1}{2\sigma^2} \|x - \mu_{\theta}(z)\|_2^2$$



$P(\theta|X)$

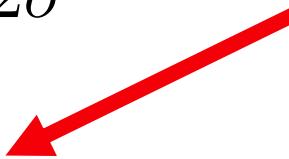
**Mean squared error**  
between  $x_i$  and its  
reconstruction (obtained by  
auto-encoding it)

- The other terms do not depend on  $\theta$ , so it makes sense to see  $\log p_{\theta}(x|z)$  as a reconstruction error.

## Why is this a reconstruction error?

- In the particular case of a Gaussian observation model  $p_{\theta}(x|z) = \mathcal{N}(x|\mu_{\theta}(z), \sigma^2 I_D)$  with a constant and isotropic covariance, we can use the formula of the Gaussian density to get

$$\log p_{\theta}(x|z) = -\frac{D}{2} \log(2\pi) - D \log \sigma - \frac{1}{2\sigma^2} \|x - \mu_{\theta}(z)\|_2^2$$



$P(\theta|X)$

**Mean squared error**  
between  $x_i$  and its  
reconstruction (obtained by  
auto-encoding it)

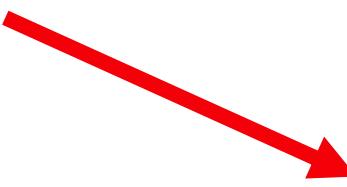
- The other terms do not depend on  $\theta$ , so it makes sense to see  $\log p_{\theta}(x|z)$  as a reconstruction error.
- Note that the autoencoding process is stochastic, as it involves sampling  $z \sim q_{\gamma}(z|x)$

## Why is this a reconstruction error?

- If we have a Bernoulli observation model  $p_{\theta}(\mathbf{x}|\mathbf{z}) = \prod_{j=1}^d \mathcal{B}(x_j|\pi_{\theta}(\mathbf{z})_j)$ , we get

$$\log p_{\theta}(\mathbf{x}|\mathbf{z}) = - \sum_{j=1}^d \text{XEnt}(\mathbf{x}, \pi_{\theta}(\mathbf{z}))$$

$\mathbb{P}(\theta|X)$



**Cross-entropy loss**  
between  $\mathbf{x}_i$  and its  
reconstruction (obtained by  
auto-encoding it)

## Why is the KL term a regulariser?

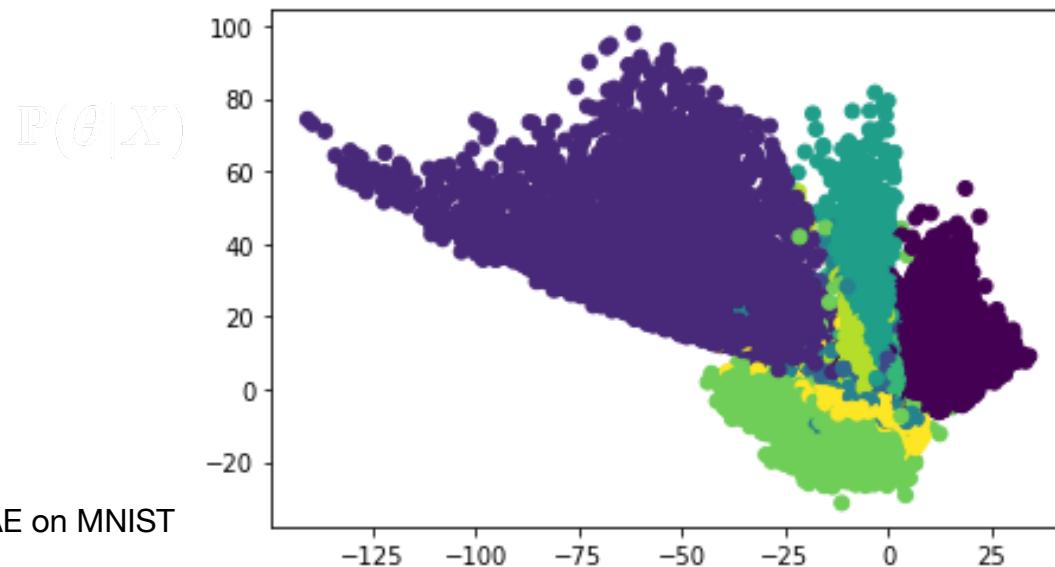
- The KL part of the VAE bound is equal to  $\text{KL} \left( \prod_{i=1}^n q_{\gamma}(\mathbf{z}_i | \mathbf{x}_i) \middle\| \prod_{i=1}^n p_{\theta}(\mathbf{z}_i) \right)$ , which is the **divergence between the approximate posterior (aka encoding) and the prior**.
- Minimising the VAE bound will therefore lead to solutions such that **encodings are somewhat collectively similar to the prior**.



$$P(\theta|X)$$

## Why is the KL term a regulariser?

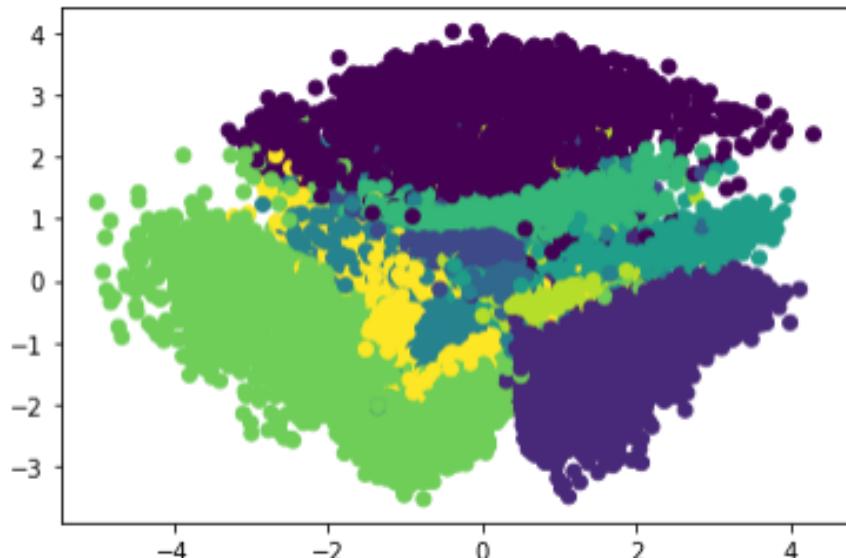
- The KL part of the VAE bound is equal to  $\text{KL} \left( \prod_{i=1}^n q_{\gamma}(\mathbf{z}_i | \mathbf{x}_i) \parallel \prod_{i=1}^n p_{\theta}(\mathbf{z}_i) \right)$ , which is the **divergence between the approximate posterior (aka encoding) and the prior**.
- Minimising the VAE bound will therefore lead to solutions such that **encodings are somewhat collectively similar to the prior**.
- Typically, remember that the prior is often simply standard Gaussian! In practice, the encodings of traditional VAEs have no reason to be standard Gaussian...



2D code space of a standard AE on MNIST

## Why is the KL term a regulariser?

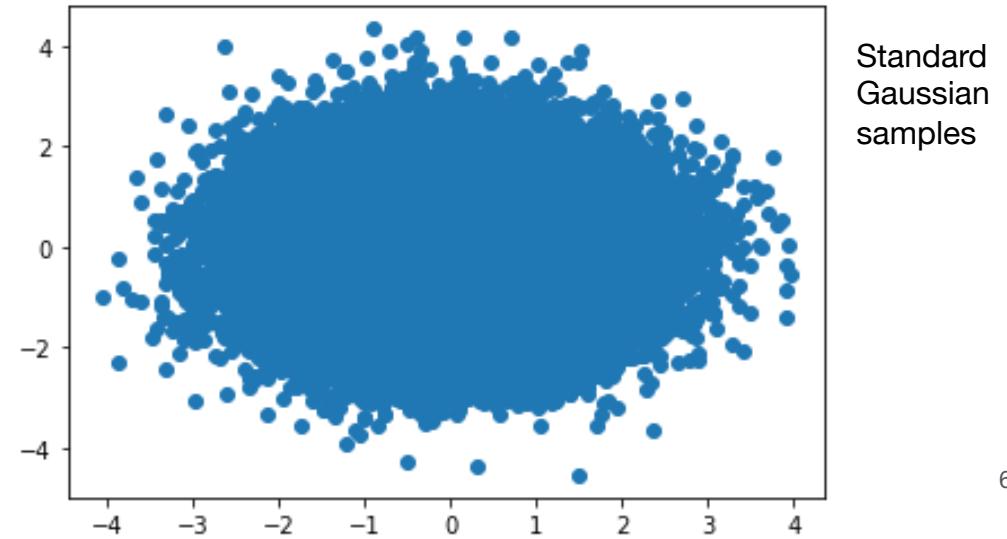
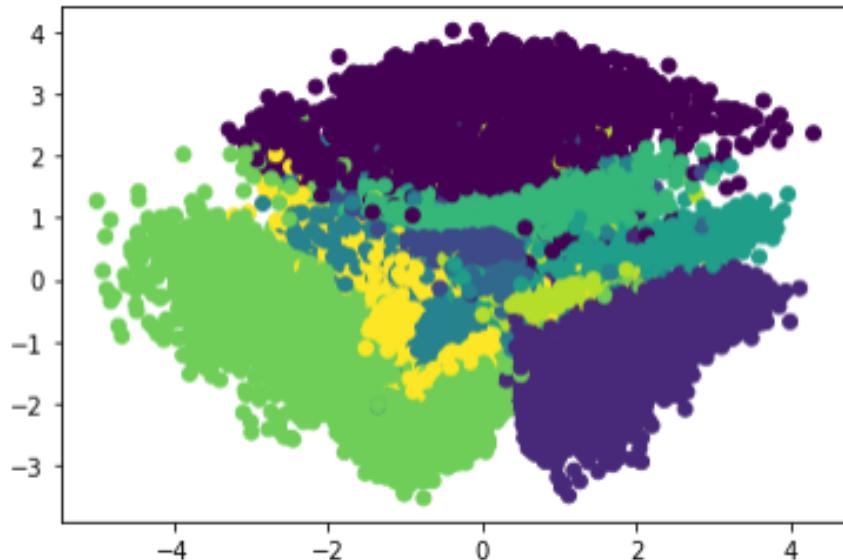
- The KL part of the VAE bound is equal to  $\text{KL} \left( \prod_{i=1}^n q_\gamma(\mathbf{z}_i | \mathbf{x}_i) \middle\| \prod_{i=1}^n p_\theta(\mathbf{z}_i) \right)$ , which is the **divergence between the approximate posterior (aka encoding) and the prior**.
- Minimising the VAE bound will therefore lead to solutions such that **encodings are somewhat collectively similar to the prior**.
- Typically, remember that the prior is often simply standard Gaussian! In practice, **the encodings of a VAE are actually quite Gaussian!**



2D code space of  
a VAE on MNIST

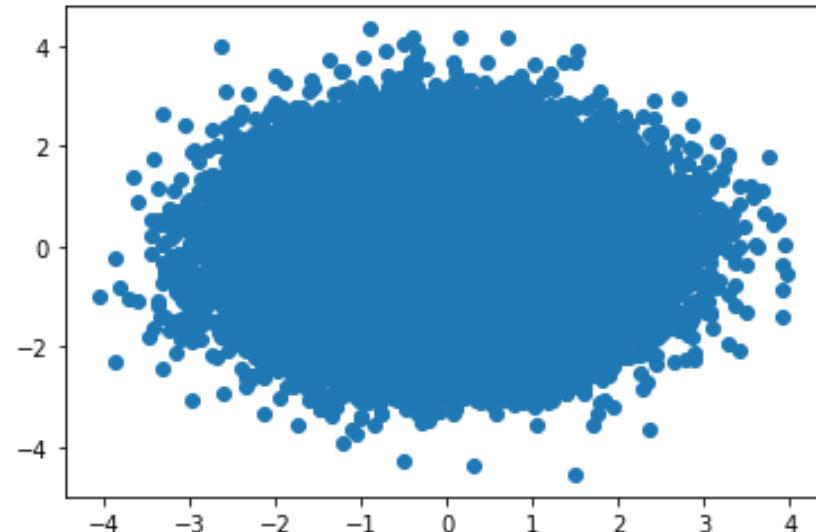
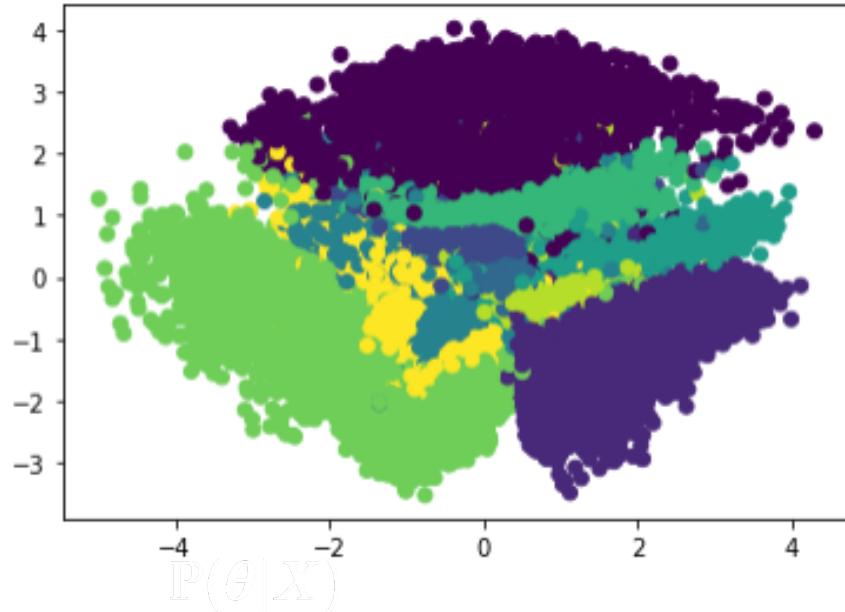
## Why is the KL term a regulariser?

- The KL part of the VAE bound is equal to  $\text{KL} \left( \prod_{i=1}^n q_\gamma(\mathbf{z}_i | \mathbf{x}_i) \middle\| \prod_{i=1}^n p_\theta(\mathbf{z}_i) \right)$ , which is the **divergence between the approximate posterior (aka encoding) and the prior**.
- Minimising the VAE bound will therefore lead to solutions such that **encodings are somewhat collectively similar to the prior**.
- Typically, remember that the prior is often simply standard Gaussian! In practice, **the encodings of a VAE are actually quite Gaussian!**



## More on this KL regulariser

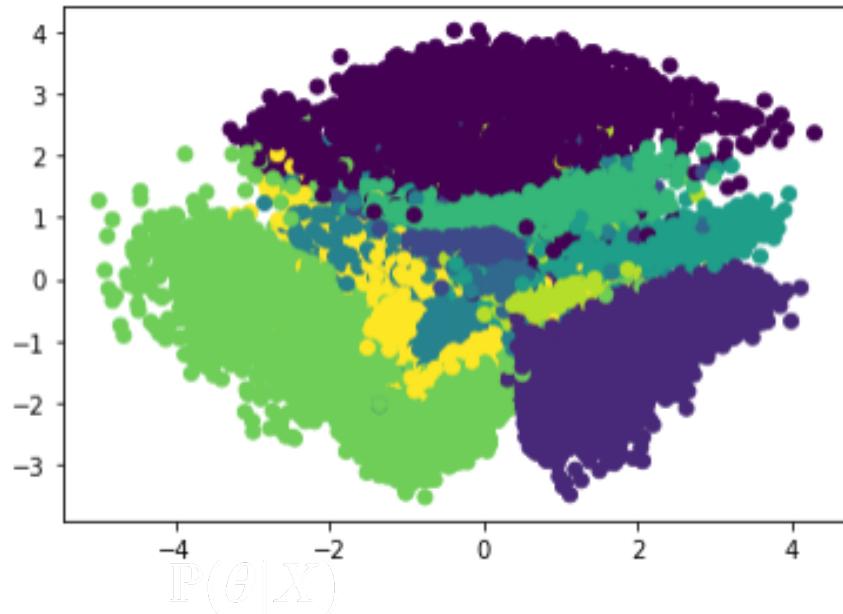
- So, the encodings of a VAE are actually quite Gaussian!



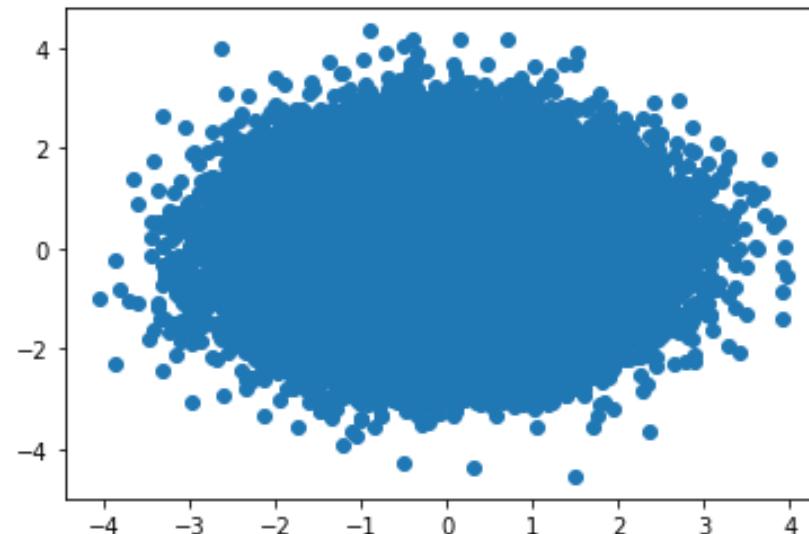
- Is it a good or a bad thing?

## More on this KL regulariser

- So, the encodings of a VAE are actually quite Gaussian!



2D code space of  
a VAE on MNIST

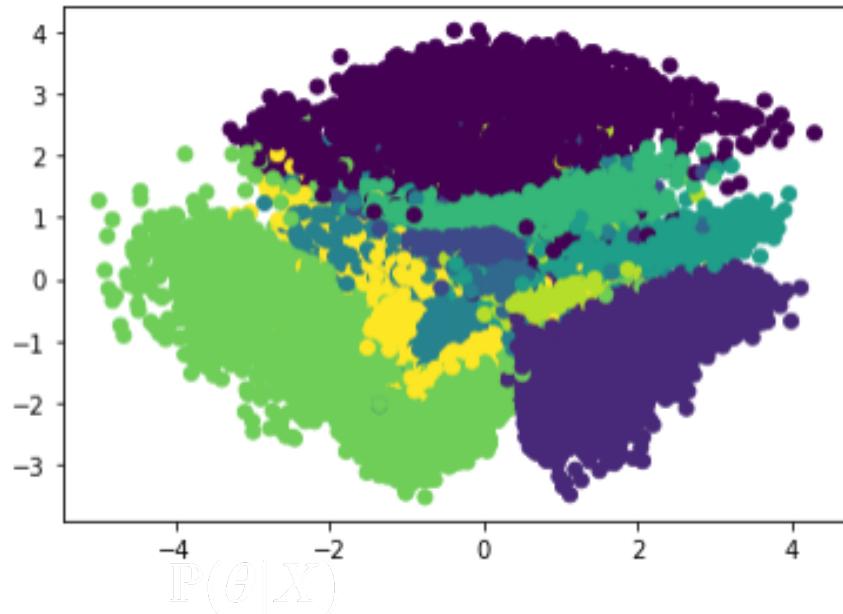


Standard  
Gaussian  
samples

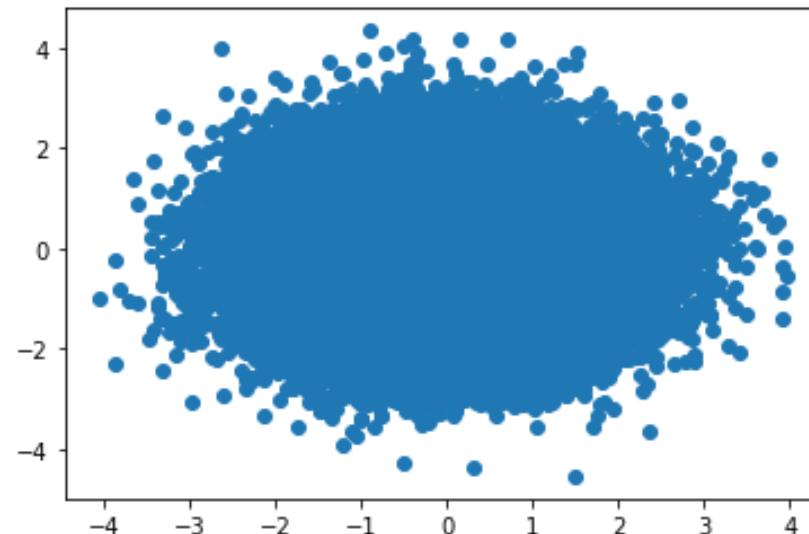
- Is it a good or a bad thing?
  - Good: Gaussian data are well-behaved (in particular, Euclidean geometry makes sense), the latent space has no nonlinear structure

## More on this KL regulariser

- So, the encodings of a VAE are actually quite Gaussian!



2D code space of  
a VAE on MNIST

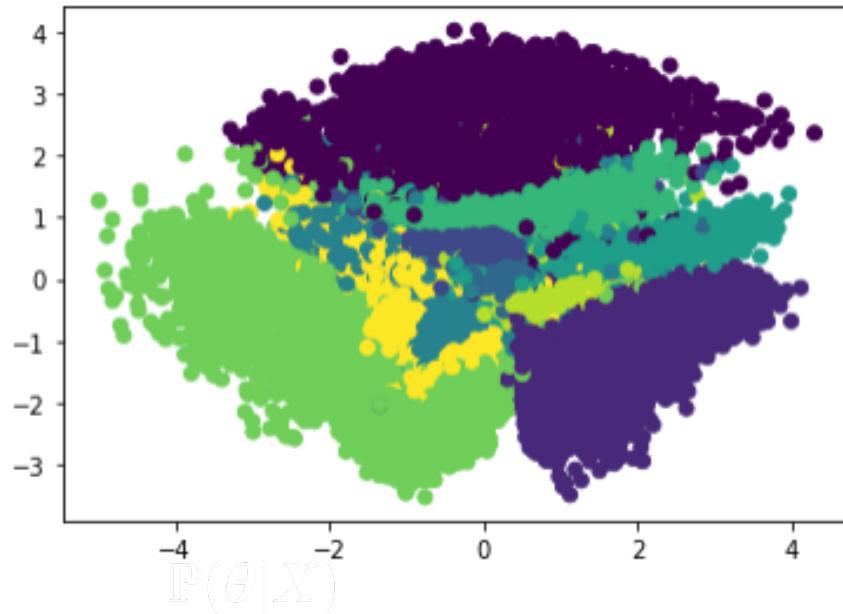


Standard  
Gaussian  
samples

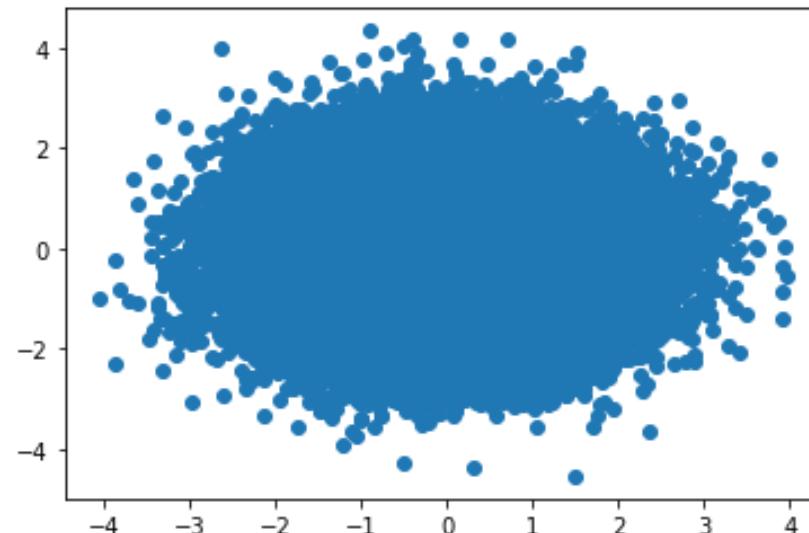
- Is it a good or a bad thing?
  - Good: Gaussian data are well-behaved (in particular, Euclidean geometry makes sense), the latent space has no nonlinear structure
  - Bad: The latent space is not discriminative! Any idea how to simply solve this?

## More on this KL regulariser

- So, the encodings of a VAE are actually quite Gaussian!



2D code space of  
a VAE on MNIST



Standard  
Gaussian  
samples

- Is it a good or a bad thing?
  - Good: Gaussian data are well-behaved (in particular, Euclidean geometry makes sense), the latent space has no nonlinear structure
  - Bad: The latent space is not discriminative! Any idea how to simply solve this? One solution is to use a GMM prior.

## More on this KL regulariser

---

- In the common case where both prior and approximate posterior are Gaussian, this KL regulariser is just a KL between Gaussians, which has a closed-form expression. Here is the general formula for  $p$ -variate Gaussians with full covariance

$$\begin{aligned}\mathbf{KL}(\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) || \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)) &= \\ \frac{1}{2} \left( \mathbf{tr}(\boldsymbol{\Sigma}_1^{-1} \boldsymbol{\Sigma}_0) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^T \boldsymbol{\Sigma}_1^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) + \log \frac{\det \boldsymbol{\Sigma}_1}{\det \boldsymbol{\Sigma}_0} - p \right)\end{aligned}$$

$$\mathbb{P}(\theta | X)$$

## More on this KL regulariser

---

- In the common case where both prior and approximate posterior are Gaussian, this KL regulariser is just a KL between Gaussians, which has a closed-form expression. Here is the general formula for  $p$ -variate Gaussians with full covariance

$$\begin{aligned}\mathbf{KL}(\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) || \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)) &= \\ \frac{1}{2} \left( \mathbf{tr}(\boldsymbol{\Sigma}_1^{-1} \boldsymbol{\Sigma}_0) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^T \boldsymbol{\Sigma}_1^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) + \log \frac{\det \boldsymbol{\Sigma}_1}{\det \boldsymbol{\Sigma}_0} - p \right)\end{aligned}$$

- It's easy to backprop through this, and this was used in the seminal VAE papers!  
 $P(\theta|X)$
- Not easy to generalise beyond Gaussians...

## How do we actually train VAEs?

---

- What have we done so far? We have created a family of **lower bounds of the log-likelihood**, but how do optimise them?
- Remember that the IWAE bounds are defined as


$$\mathcal{L}_K(\boldsymbol{\theta}, \gamma) = \sum_{i=1}^n \mathbb{E}_{\mathbf{z}_{i1}, \dots, \mathbf{z}_{iK} \sim q_\gamma(\mathbf{z}|\mathbf{x}_i)} \left[ \log \frac{1}{K} \sum_{k=1}^K \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_i | \mathbf{z}_{ik}) p(\mathbf{z}_{ik})}{q_\gamma(\mathbf{z}_{ik} | \mathbf{x}_i)} \right]$$

$\mathbb{P}(\boldsymbol{\theta}|X)$

- **How do we maximise this? We can't even compute this exactly?**

## How do we actually train VAEs?

---

- What have we done so far? We have created a family of **lower bounds of the log-likelihood**, but how do optimise them?
- Remember that the IWAE bounds are defined as


$$\mathcal{L}_K(\boldsymbol{\theta}, \gamma) = \sum_{i=1}^n \mathbb{E}_{\mathbf{z}_{i1}, \dots, \mathbf{z}_{iK} \sim q_\gamma(\mathbf{z}|\mathbf{x}_i)} \left[ \log \frac{1}{K} \sum_{k=1}^K \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_i | \mathbf{z}_{ik}) p(\mathbf{z}_{ik})}{q_\gamma(\mathbf{z}_{ik} | \mathbf{x}_i)} \right]$$

$\mathbb{P}(\theta|X)$

- **How do we maximise this? We can't even compute this exactly?**
- The idea is to use stochastic gradient descent (SGD, or one of its variants). What do we need to compute to perform SGD on an objective?

## How do we actually train VAEs?

---

- What have we done so far? We have created a family of **lower bounds of the log-likelihood**, but how do optimise them?
- Remember that the IWAE bounds are defined as

$$\mathcal{L}_K(\theta, \gamma) = \sum_{i=1}^n \mathbb{E}_{\mathbf{z}_{i1}, \dots, \mathbf{z}_{iK} \sim q_\gamma(\mathbf{z}|\mathbf{x}_i)} \left[ \log \frac{1}{K} \sum_{k=1}^K \frac{p_\theta(\mathbf{x}_i | \mathbf{z}_{ik}) p(\mathbf{z}_{ik})}{q_\gamma(\mathbf{z}_{ik} | \mathbf{x}_i)} \right]$$

$\mathbb{P}(\theta|X)$

- **How do we maximise this? We can't even compute this exactly?**
- The idea is to use stochastic gradient descent (SGD, or one of its variants). What do we need to compute to perform SGD on an objective? **Unbiased estimates of the gradients!**
- As we'll see, it is doable to compute unbiased estimates of  $\nabla_{\theta, \gamma} \mathcal{L}_K(\theta, \gamma)$

# Unbiased IWAE gradients: looking at a more general problem

---

- Let's look at the problem in a more general form, we want unbiased estimates of the gradients of a function of the form

$$f(\theta, \gamma) = \mathbb{E}_{\mathbf{w} \sim \pi_\gamma(\mathbf{w})} [g(\theta, \gamma, \mathbf{w})]$$

- In the next few slides, we will describe a few recipes to compute such estimates. **This is a task that is useful in a lot of ML contexts**, e.g. reinforcement learning, explainability... For more details, and more recipe, you may look at the following nice review

Journal of Machine Learning Research 21 (2020) 1-62

## Monte Carlo Gradient Estimation in Machine Learning

Shakir Mohamed<sup>\*1</sup>

Mihaela Rosca<sup>\*1 2</sup>

Michael Figurnov<sup>\*1</sup>

Andriy Mnih<sup>\*1</sup>

SHAKIR@GOOGLE.COM

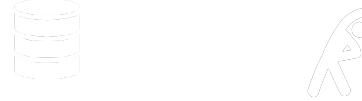
MIHAELACR@GOOGLE.COM

MFIGURNOV@GOOGLE.COM

AMNIH@GOOGLE.COM

Let's start with the easy part:  $\nabla_{\theta}$

- Note that the first parameter only appears **inside** the expectation

$$f(\boldsymbol{\theta}, \gamma) = \mathbb{E}_{\mathbf{w} \sim \pi_{\gamma}(\mathbf{w})} [g(\boldsymbol{\theta}, \gamma, \mathbf{w})]$$


$$\mathbb{P}(\theta|X)$$

## Let's start with the easy part: $\nabla_{\theta}$

- Note that the first parameter only appears **inside** the expectation

$$f(\boldsymbol{\theta}, \gamma) = \mathbb{E}_{\mathbf{w} \sim \pi_\gamma(\mathbf{w})} [g(\boldsymbol{\theta}, \gamma, \mathbf{w})]$$


- Therefore, if  $g$  and  $\pi$  are nice enough (e.g. when  $\nabla_{\theta}g$  can be dominated), we can use Leibniz's integral rule to get

$$\mathbb{P}(\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}, \gamma) = \mathbb{E}_{\mathbf{w} \sim \pi_\gamma(\mathbf{w})} [\nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}, \gamma, \mathbf{w})]$$

## Let's start with the easy part: $\nabla_{\theta}$

- Note that the first parameter only appears **inside** the expectation

$$f(\boldsymbol{\theta}, \gamma) = \mathbb{E}_{\mathbf{w} \sim \pi_\gamma(\mathbf{w})} [g(\boldsymbol{\theta}, \gamma, \mathbf{w})]$$


- Therefore, if  $g$  and  $\pi$  are nice enough (e.g. when  $\nabla_{\theta} g$  can be dominated), we can use Leibniz's integral rule to get

$$\mathbb{P}(\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}, \gamma) = \mathbb{E}_{\mathbf{w} \sim \pi_\gamma(\mathbf{w})} [\nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}, \gamma, \mathbf{w})]$$

which means that we can get an unbiased estimate of  $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}, \gamma)$  by simply sampling  $\mathbf{w}_1, \dots, \mathbf{w}_K \sim \pi_\gamma(\mathbf{w})$  and then computing

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}, \gamma) \approx \frac{1}{K} \sum_{k=1}^K \nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}, \gamma, \mathbf{w}_k)$$

## Let's start with the easy part: $\nabla_{\theta}$

- Note that the first parameter only appears **inside** the expectation

$$f(\boldsymbol{\theta}, \gamma) = \mathbb{E}_{\mathbf{w} \sim \pi_{\gamma}(\mathbf{w})} [g(\boldsymbol{\theta}, \gamma, \mathbf{w})]$$


- Therefore, if  $g$  and  $\pi$  are nice enough (e.g. when  $\nabla_{\theta}g$  can be dominated), we can use Leibniz's integral rule to get

$$\mathbb{P}(\boldsymbol{\theta}) \nabla_{\theta} f(\boldsymbol{\theta}, \gamma) = \mathbb{E}_{\mathbf{w} \sim \pi_{\gamma}(\mathbf{w})} [\nabla_{\theta} g(\boldsymbol{\theta}, \gamma, \mathbf{w})]$$

which means that we can get an unbiased estimate of  $\nabla_{\theta} f(\boldsymbol{\theta}, \gamma)$  by simply sampling  $\mathbf{w}_1, \dots, \mathbf{w}_K \sim \pi_{\gamma}(\mathbf{w})$  and then computing

$$\nabla_{\theta} f(\boldsymbol{\theta}, \gamma) \approx \frac{1}{K} \sum_{k=1}^K \nabla_{\theta} g(\boldsymbol{\theta}, \gamma, \mathbf{w})$$

**Often  $K=1$   
will be  
enough!**

## Now the tricky part: $\nabla_\gamma$

- This parameter appears both **inside** and **outside** the expectation:

$$f(\theta, \gamma) = \mathbb{E}_{\mathbf{w} \sim \pi_\gamma(\mathbf{w})} [g(\theta, \gamma, \mathbf{w})]$$


$$\mathbb{P}(\theta|X)$$

## Now the tricky part: $\nabla_{\gamma}$

- This parameter appears both **inside** and **outside** the expectation:

$$f(\theta, \gamma) = \mathbb{E}_{w \sim \pi_\gamma(w)} [g(\theta, \gamma, w)]$$


- So it's not that simple... Let us write the expectation:

$$f(\theta, \gamma) = \int g(\theta, \gamma, w) \pi_\gamma(w) d\mathbf{w}$$


## Now the tricky part: $\nabla_{\gamma}$

- This parameter appears both **inside** and **outside** the expectation:

$$f(\theta, \gamma) = \mathbb{E}_{w \sim \pi_{\gamma}(w)} [g(\theta, \gamma, w)]$$


- So it's not that simple... Let us write the expectation:

$$f(\theta, \gamma) = \int g(\theta, \gamma, w) \pi_{\gamma}(w) d\mathbf{w}$$


- **It makes sense to use Leibniz's rule again:**

$$\nabla_{\gamma} f(\theta, \gamma) = \int \nabla_{\gamma} [g(\theta, \gamma, w) \pi_{\gamma}(w)] d\mathbf{w}$$

## Now the tricky part: $\nabla_{\gamma}$

- It makes sense to use Leibniz's rule again:

$$\nabla_{\gamma} f(\theta, \gamma) = \int \nabla_{\gamma} [g(\theta, \gamma, w) \pi_{\gamma}(w)] dw$$

And now we can use the usual rule to **differentiate a product**:

$$\nabla_{\gamma} f(\theta, \gamma) = \int \nabla_{\gamma} [g(\theta, \gamma, w)] \pi_{\gamma}(w) dw + \int g(\theta, \gamma, w) \nabla_{\gamma} [\pi_{\gamma}(w)] dw$$

$$P(\theta|X)$$

## Now the tricky part: $\nabla_{\gamma}$

- It makes sense to use Leibniz's rule again:

$$\nabla_{\gamma} f(\theta, \gamma) = \int \nabla_{\gamma} [g(\theta, \gamma, w) \pi_{\gamma}(w)] dw$$

And now we can use the usual rule to **differentiate a product**:



$$\nabla_{\gamma} f(\theta, \gamma) = \int \nabla_{\gamma} [g(\theta, \gamma, w)] \pi_{\gamma}(w) dw + \int g(\theta, \gamma, w) \nabla_{\gamma} [\pi_{\gamma}(w)] dw$$

- **Which one of these two terms is easy to unbiasedly estimate?**

## Now the tricky part: $\nabla_{\gamma}$

- It makes sense to use Leibniz's rule again:

$$\nabla_{\gamma} f(\theta, \gamma) = \int \nabla_{\gamma} [g(\theta, \gamma, w) \pi_{\gamma}(w)] dw$$

And now we can use the usual rule to **differentiate a product**:



$$\nabla_{\gamma} f(\theta, \gamma) = \int \nabla_{\gamma} [g(\theta, \gamma, w)] \pi_{\gamma}(w) dw + \int g(\theta, \gamma, w) \nabla_{\gamma} [\pi_{\gamma}(w)] dw$$

- **Which one of these two terms is easy to unbiasedly estimate? The first one because it is an expectation!**

$$\int \nabla_{\gamma} [g(\theta, \gamma, w)] \pi_{\gamma}(w) dw \approx \frac{1}{K} \sum_{k=1}^K \nabla_{\gamma} [g(\theta, \gamma, w_k)]$$

## Now the tricky part: $\nabla_{\gamma}$

- The real tricky part is what's left: the second term

$$\int g(\theta, \gamma, w) \nabla_{\gamma} [\pi_{\gamma}(w)] dw$$

that is not an expected value!



$$P(\theta|X)$$

## Now the tricky part: $\nabla_{\gamma}$

- The real tricky part is what's left: the second term

$$\int g(\theta, \gamma, w) \nabla_{\gamma} [\pi_{\gamma}(w)] dw$$

that is not an expected value!

- So, it's not an expected value, but we can turn it into one by dividing/multiplying!



$$\int g(\theta, \gamma, w) \frac{\nabla_{\gamma} [\pi_{\gamma}(w)]}{\pi_{\gamma}(w)} \pi_{\gamma}(w) dw$$

$$P(\theta|X)$$

## Now the tricky part: $\nabla_{\gamma}$

- The real tricky part is what's left: the second term

$$\int g(\theta, \gamma, w) \nabla_{\gamma} [\pi_{\gamma}(w)] dw$$

that is not an expected value!

- So, it's not an expected value, but we can turn it into one by dividing/multiplying!



$$\int g(\theta, \gamma, w) \frac{\nabla_{\gamma} [\pi_{\gamma}(w)]}{\pi_{\gamma}(w)} \pi_{\gamma}(w) dw$$

Now, if we also remark that  $\frac{\nabla_{\gamma} [\pi_{\gamma}(w)]}{\pi_{\gamma}(w)} = \nabla_{\gamma} \log \pi_{\gamma}(w)$ , we finally get

$P(\theta|X)$

## Now the tricky part: $\nabla_{\gamma}$

- The real tricky part is what's left: the second term

$$\int g(\theta, \gamma, w) \nabla_{\gamma} [\pi_{\gamma}(w)] dw$$

that is not an expected value!

- So, it's not an expected value, but we can turn it into one by dividing/multiplying!



$$\int g(\theta, \gamma, w) \frac{\nabla_{\gamma} [\pi_{\gamma}(w)]}{\pi_{\gamma}(w)} \pi_{\gamma}(w) dw$$

Now, if we also remark that  $\frac{\nabla_{\gamma} [\pi_{\gamma}(w)]}{\pi_{\gamma}(w)} = \nabla_{\gamma} \log \pi_{\gamma}(w)$ , we finally get

$$\int g(\theta, \gamma, w) \nabla_{\gamma} [\pi_{\gamma}(w)] dw = \mathbb{E}_{w \sim \pi_{\gamma}} [g(\theta, \gamma, w) \nabla_{\gamma} \log \pi_{\gamma}(w)]$$

$$\approx \frac{1}{K} \sum_{k=1}^K g(\theta, \gamma, w_k) \nabla_{\gamma} \log \pi_{\gamma}(w_k)$$

## Now the tricky part: $\nabla_{\gamma}$

- The estimate  $\int g(\theta, \gamma, w) \nabla_{\gamma} [\pi_{\gamma}(w)] dw = \mathbb{E}_{w \sim \pi_{\gamma}} [g(\theta, \gamma, w) \nabla_{\gamma} \log \pi_{\gamma}(w)]$   
 $\approx \frac{1}{K} \sum_{k=1}^K g(\theta, \gamma, w_k) \nabla_{\gamma} \log \pi_{\gamma}(w_k)$

is called the **score function gradient**, or the **REINFORCE** estimate.

$$\mathbb{P}(\theta|X)$$

## Now the tricky part: $\nabla_{\gamma}$

- The estimate  $\int g(\theta, \gamma, w) \nabla_{\gamma} [\pi_{\gamma}(w)] dw = \mathbb{E}_{w \sim \pi_{\gamma}} [g(\theta, \gamma, w) \nabla_{\gamma} \log \pi_{\gamma}(w)]$   
 $\approx \frac{1}{K} \sum_{k=1}^K g(\theta, \gamma, w_k) \nabla_{\gamma} \log \pi_{\gamma}(w_k)$

is called the **score function gradient**, or the **REINFORCE** estimate.

- One big issue is that it can have **potentially very large variance**, and typically requires a lot of samples to be accurate.

$$\mathbb{P}(\theta|X)$$

## Now the tricky part: $\nabla_{\gamma}$

- The estimate  $\int g(\theta, \gamma, w) \nabla_{\gamma} [\pi_{\gamma}(w)] dw = \mathbb{E}_{w \sim \pi_{\gamma}} [g(\theta, \gamma, w) \nabla_{\gamma} \log \pi_{\gamma}(w)]$   
 $\approx \frac{1}{K} \sum_{k=1}^K g(\theta, \gamma, w_k) \nabla_{\gamma} \log \pi_{\gamma}(w_k)$

is called the **score function gradient**, or the **REINFORCE** estimate.

- One big issue is that it can have **potentially very large variance**, and typically requires a lot of samples to be accurate.

$$\mathbb{P}(\theta|X)$$

- Can we do better? In general, not really. **But in some specific cases, yes!**

## Now the tricky part: $\nabla_{\gamma}$

- The estimate  $\int g(\theta, \gamma, w) \nabla_{\gamma} [\pi_{\gamma}(w)] dw = \mathbb{E}_{w \sim \pi_{\gamma}} [g(\theta, \gamma, w) \nabla_{\gamma} \log \pi_{\gamma}(w)]$   
 $\approx \frac{1}{K} \sum_{k=1}^K g(\theta, \gamma, w_k) \nabla_{\gamma} \log \pi_{\gamma}(w_k)$

is called the **score function gradient**, or the **REINFORCE** estimate.

- One big issue is that it can have **potentially very large variance**, and typically requires a lot of samples to be accurate.

$$P(\theta|X)$$

- Can we do better? In general, not really. **But in some specific cases, yes!**
- In the next slide, we'll see one of such cases: **the Gaussian reparametrisation trick.**

## Reparametrisation trick for $\nabla_{\gamma}$

- Our goal is still to estimate our tricky term  $\nabla_{\gamma} \int g(\theta, \gamma, w) \pi_{\gamma}(w) dw$
- Again, the main issue is that the density depends on the parameter of interest. Can we destroy this dependence? Can we **push  $\gamma$  away from the density** we're integrating against?



$$P(\theta|X)$$

## Reparametrisation trick for $\nabla_{\gamma}$

- Our goal is still to estimate our tricky term  $\nabla_{\gamma} \int g(\theta, \gamma, w) \pi_{\gamma}(w) dw$
- Again, the main issue is that the density depends on the parameter of interest. Can we destroy this dependence? Can we **push  $\gamma$  away from the density** we're integrating against?  

- In some important cases, **yes!** The main example is the **Gaussian case**  $\pi_{\gamma} = \mathcal{N}(\mu_{\gamma}, \Sigma_{\gamma})$

$$P(\theta|X)$$

## Reparametrisation trick for $\nabla_{\gamma}$

- Our goal is still to estimate our tricky term  $\nabla_{\gamma} \int g(\theta, \gamma, w) \pi_{\gamma}(w) dw$
- Again, the main issue is that the density depends on the parameter of interest. Can we destroy this dependence? Can we **push  $\gamma$  away from the density** we're integrating against?
- In some important cases, **yes!** The main example is the **Gaussian case**  $\pi_{\gamma} = \mathcal{N}(\mu_{\gamma}, \text{Diag}(\sigma_{\gamma}))$
- In this setting, it is clear that sampling  $w \sim \pi_{\gamma}$  can be done by computing

$$P(\theta|X)$$

$$\mathbf{w} = \mu_{\gamma} + \sigma_{\gamma} \odot \boldsymbol{\varepsilon}$$

where  $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \mathbf{I})$

## Reparametrisation trick for $\nabla_{\gamma}$

- Sampling  $w \sim \pi_{\gamma}$  can be done by computing  $w = \mu_{\gamma} + \sigma_{\gamma} \odot \varepsilon$

where  $C_{\gamma}$  is the Cholesky decomposition of the covariance, and  $\varepsilon \sim \mathcal{N}(0, \mathbf{I})$ .



$$\mathbb{P}(\theta|X)$$

## Reparametrisation trick for $\nabla_{\gamma}$

- Sampling  $\mathbf{w} \sim \pi_{\gamma}$  can be done by computing  $\mathbf{w} = \mu_{\gamma} + \sigma_{\gamma} \odot \boldsymbol{\varepsilon}$

and  $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \mathbf{I})$ .

- But now, the only random thing is  $\boldsymbol{\varepsilon}$ . This means we can rewrite our expectation as an expectation over  $\boldsymbol{\varepsilon}$

$$\nabla_{\gamma} \int g(\theta, \gamma, \mathbf{w}) \pi_{\gamma}(\mathbf{w}) d\mathbf{w} = \nabla_{\gamma} \int g(\theta, \gamma, \mu_{\gamma} + C_{\gamma} \boldsymbol{\varepsilon}) p(\boldsymbol{\varepsilon}) d\boldsymbol{\varepsilon}$$

## Reparametrisation trick for $\nabla_{\gamma}$

- Sampling  $\mathbf{w} \sim \pi_{\gamma}$  can be done by computing  $\mathbf{w} = \mu_{\gamma} + \sigma_{\gamma} \odot \boldsymbol{\varepsilon}$

where.  $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \mathbf{I})$

- But now, the only random thing is  $\boldsymbol{\varepsilon}$ . This means we can rewrite our expectation as an expectation over  $\boldsymbol{\varepsilon}$

$$\nabla_{\gamma} \int g(\theta, \gamma, \mathbf{w}) \pi_{\gamma}(\mathbf{w}) d\mathbf{w} = \nabla_{\gamma} \int g(\mu_{\gamma} + \sigma_{\gamma} \odot \boldsymbol{\varepsilon}) p(\boldsymbol{\varepsilon}) d\boldsymbol{\varepsilon}$$

...and finally use Leibniz's rule

$$\begin{aligned} \nabla_{\gamma} \int g(\theta, \gamma, \mathbf{w}) \pi_{\gamma}(\mathbf{w}) d\mathbf{w} &= \nabla_{\gamma} \int g(\mu_{\gamma} + \sigma_{\gamma} \odot \boldsymbol{\varepsilon}) p(\boldsymbol{\varepsilon}) d\boldsymbol{\varepsilon} \\ &\approx \frac{1}{K} \sum_{k=1}^K \nabla_{\gamma} [g(\theta, \gamma, \mu_{\gamma} + \sigma_{\gamma} \odot \boldsymbol{\varepsilon}_k)] \end{aligned}$$

## Reparametrisation trick for $\nabla_{\gamma}$

- The estimate

$$\begin{aligned}\nabla_{\gamma} \int g(\theta, \gamma, w) \pi_{\gamma}(w) dw &= \nabla_{\gamma} \int g(\mu_{\gamma} + \sigma_{\gamma} \odot \varepsilon) p(\varepsilon) d\varepsilon \\ &\approx \frac{1}{K} \sum_{k=1}^K \nabla_{\gamma} [g(\theta, \gamma, \mu_{\gamma} + \sigma_{\gamma} \odot \varepsilon_k)]\end{aligned}$$


is often called the **reparametrisation trick** estimate. It has considerably less variance in practice than the score gradient, but can be less generally applied.

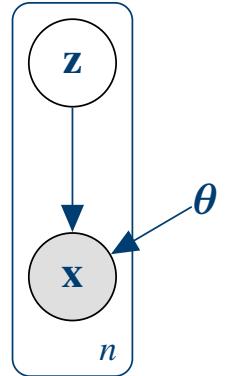
$P(\theta|X)$

- Beyond Gaussians, this can be done for more complex distributions (Dirichlet, Student's t, GMMs), but this is not easy, in particular for **discrete distributions**.
- It is automatically implemented in many libraries, for instance **Tensorflow Probability**, or **Pytorch distributions**

## A quick summary of VAEs/IWAEs so far

- We have defined a graphical model called a **deep latent variable model**  

- We have seen how to train this model by doing **approximate maximum likelihood via amortised variational inference**
- We saw that there was an **important interplay between this inference technique, and various sampling techniques** (importance sampling to define the bounds, various methods to estimate its gradients without bias).



$$P(\theta|X)$$

# 3

From VAEs to diffusion models

## From VAEs to hierarchical VAEs

---

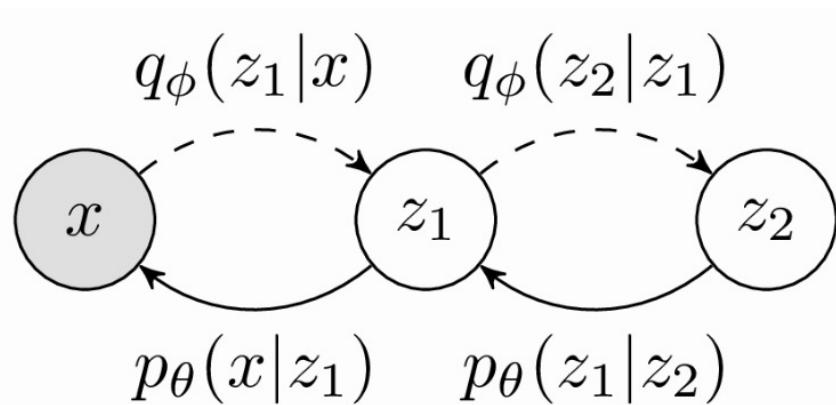
- One way to apply the good old « deeper is better » motto to VAEs is to make them **hierarchical** (aka deep, aka laddered).
- The idea is simply to make  **a VAE within a VAE** by choosing the VAE prior to be itself a deep latent variable model.

$$\mathbb{P}(\theta|X)$$

## From VAEs to hierarchical VAEs

---

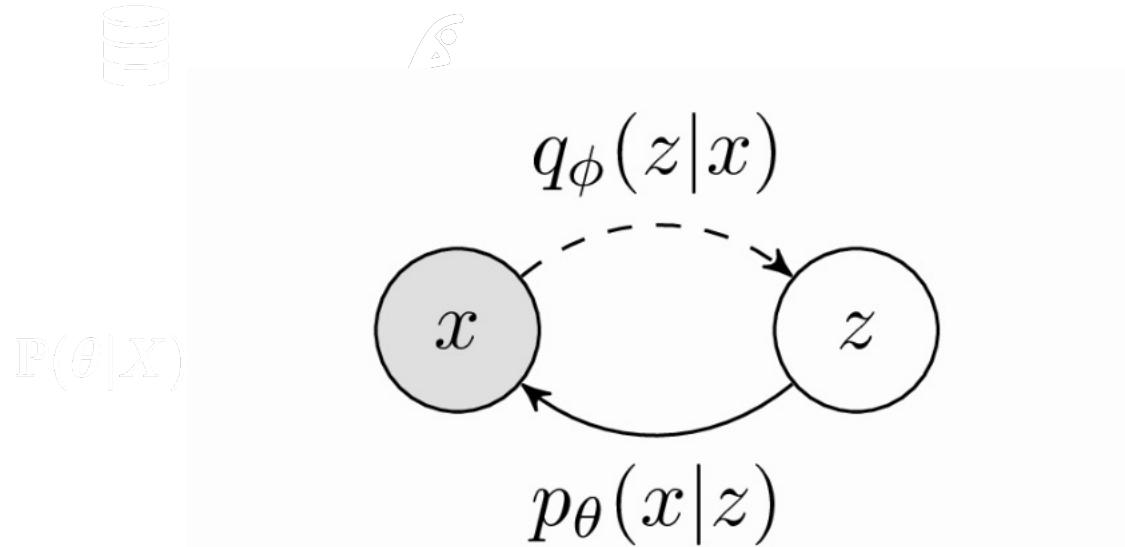
- One way to apply the good old « deeper is better » motto to VAEs is to make them **hierarchical** (aka deep, aka laddered).
- The idea is simply to make  **a VAE within a VAE** by choosing the VAE prior to be itself a deep latent variable model.



- One can then derive likelihood bounds in the exact same way.

# On DAGs and VAEs

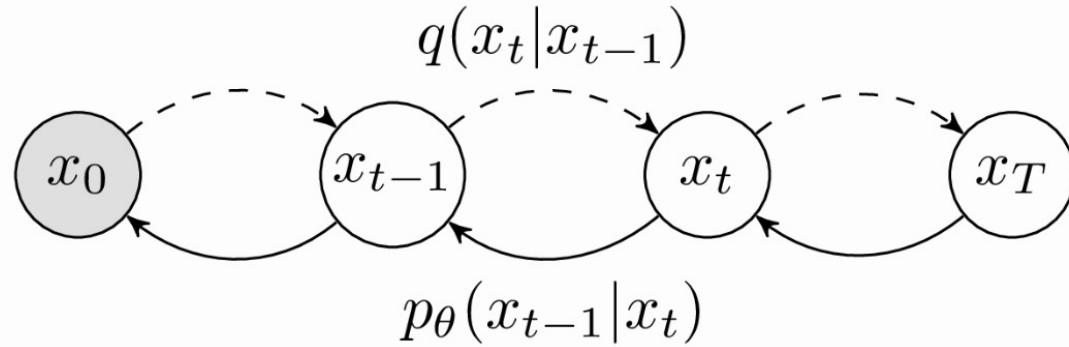
- In a standard VAE, there are **two DAGs**: the generative model itself, and the inference model (that we essentially only need for training).



- Since there are only two nodes, there are not many ways of defining DAGs...

## Diffusion models as hierarchical VAEs

- Diffusion models can be seen as a particular kind a hierarchical VAE, with a peculiar structure:



- The dimension of all latent spaces are equal to the dimension of the data
- The **inference model is fixed, and is just adding noise**

$$q(x_t|x_{t-1}) = \mathcal{N}(x_T ; x_{t-1}\sqrt{1 - \beta_t}, \beta_t I)$$

such that at the end there only remains pure noise  $q(x_T|x_0) \approx \mathcal{N}(0, I)$

- The only thing we really model is  $p(x_{t-1}|x_t)$ , whose task is to **denoise the noisy latent variables.** It is generally parametrised as a U-Net.

# Diffusion models as hierarchical VAEs

- For more details on this, see

➤ **Angus Turner's blog**

[https://angusturner.github.io/generative\\_models/2021/06/29/diffusion-probabilistic-models-I.html](https://angusturner.github.io/generative_models/2021/06/29/diffusion-probabilistic-models-I.html)

➤ **Understanding Diffusion Models: A Unified Perspective**, arXiv preprint from Calvin Luo

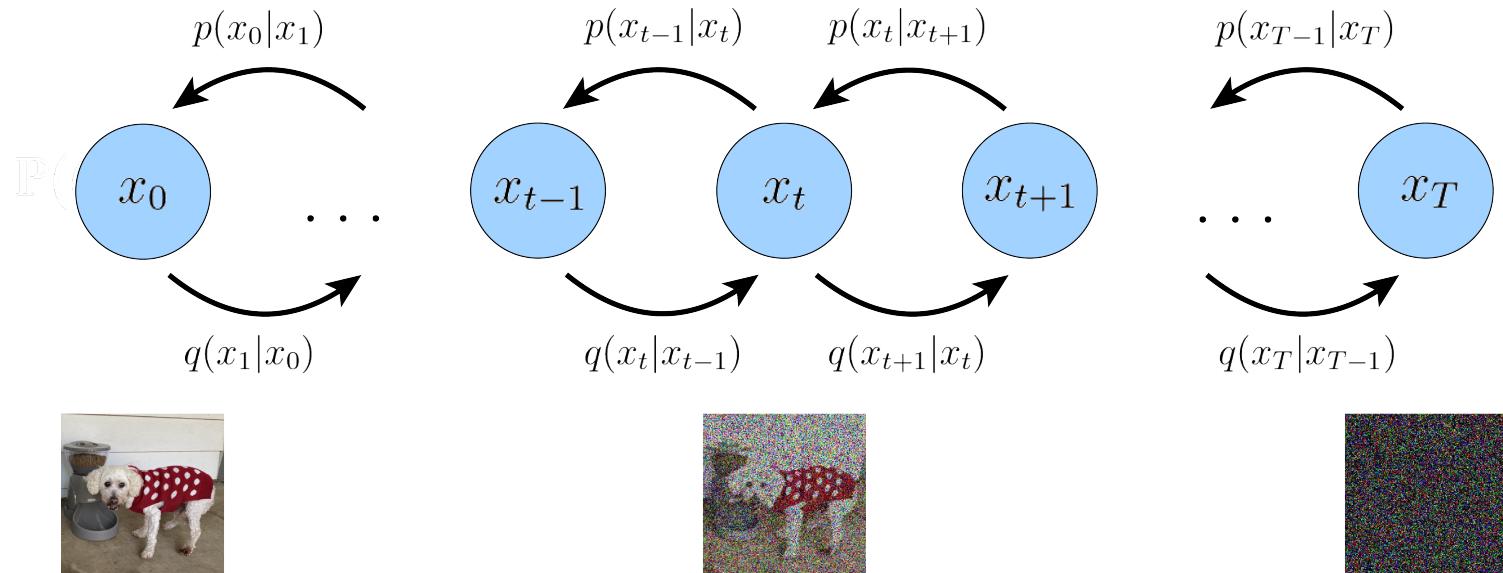


Fig. from Luo (2022)

# Diffusion models are not just VAEs

---

- There are also related to
  - **Score matching**, see e.g. Yang Song's blog <https://yang-song.net/blog/2021/score/>
  - **Another nice blog post:** <https://lilianweng.github.io/posts/2021-07-11-diffusion-models/>



$$\mathbb{P}(\theta|X)$$