Linear and deep latent variable models

from PPCA to VAEs

Pierre-Alexandre Mattei

Inria

UNIVERSITÉ CÔTE D'AZUR

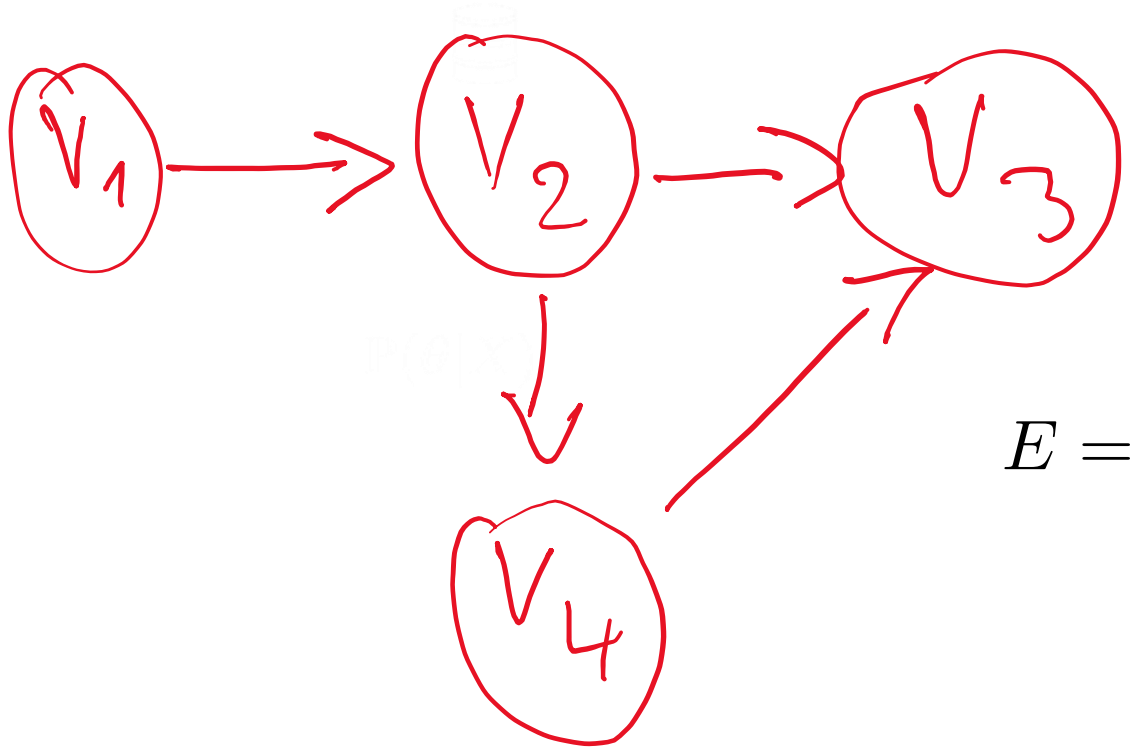
3iA Côte d'Azur
Institut interdisciplinaire d'intelligence artificielle

1

Recap on directed graphical models

What's a graph, mathematically?

Definition (directed graph)**.** *A* graph *is a pair* $G = (V, E)$ *comprising a set* V *of* ***vertices or nodes*** *together with a set* $E \subset V \times V$ *of* ***edges or arcs****.*



$$V = \{v_1, v_2, v_3, v_4\}$$

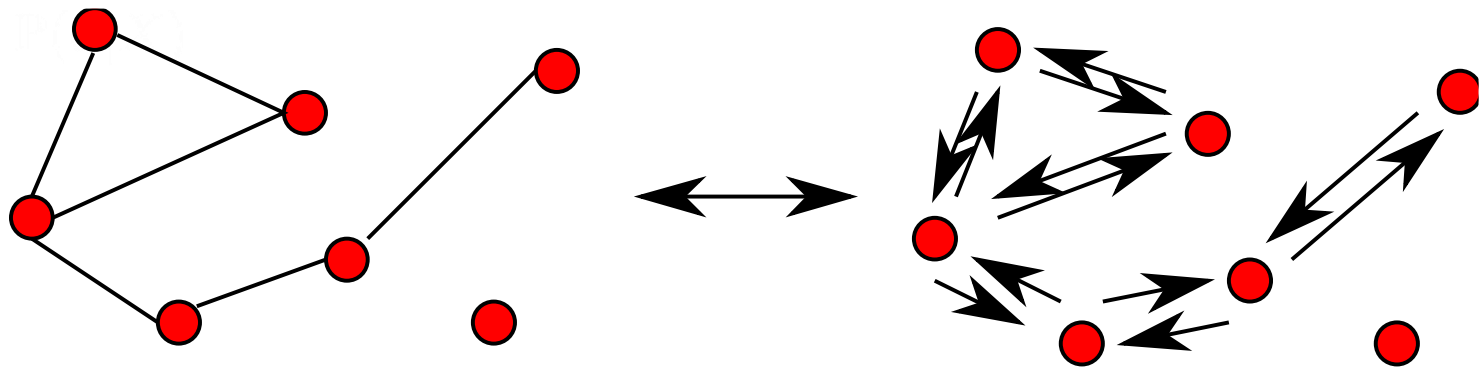
$$E = \{(v_1, v_2), (v_2, v_3), (v_2, v_4), (v_3, v_4)\}$$

What's a graph, mathematically?

Definition (directed graph)**.** *A* graph *is a pair* $G = (V, E)$ *comprising a set* V *of* ***vertices or nodes*** *together with a set* $E \subset V \times V$ *of* ***edges or arcs****.*

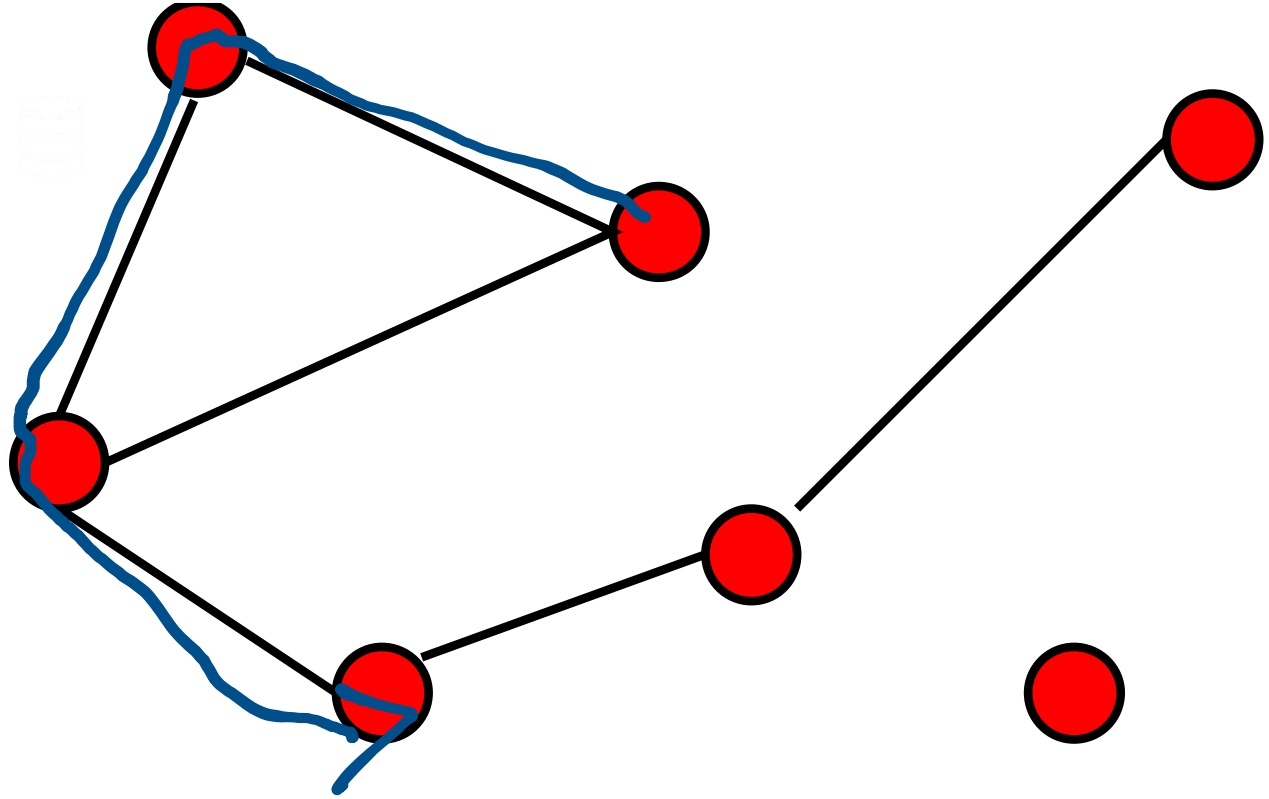
Definition (undirected graph)**.** $G = (V, E)$ *is* ***undirected*** *if all edges go in both directions: for all* $(u, v) \in V \times V$ *such that* $u \neq v$*, we have:*

$$(u, v) \in E \iff (v, u) \in E.$$



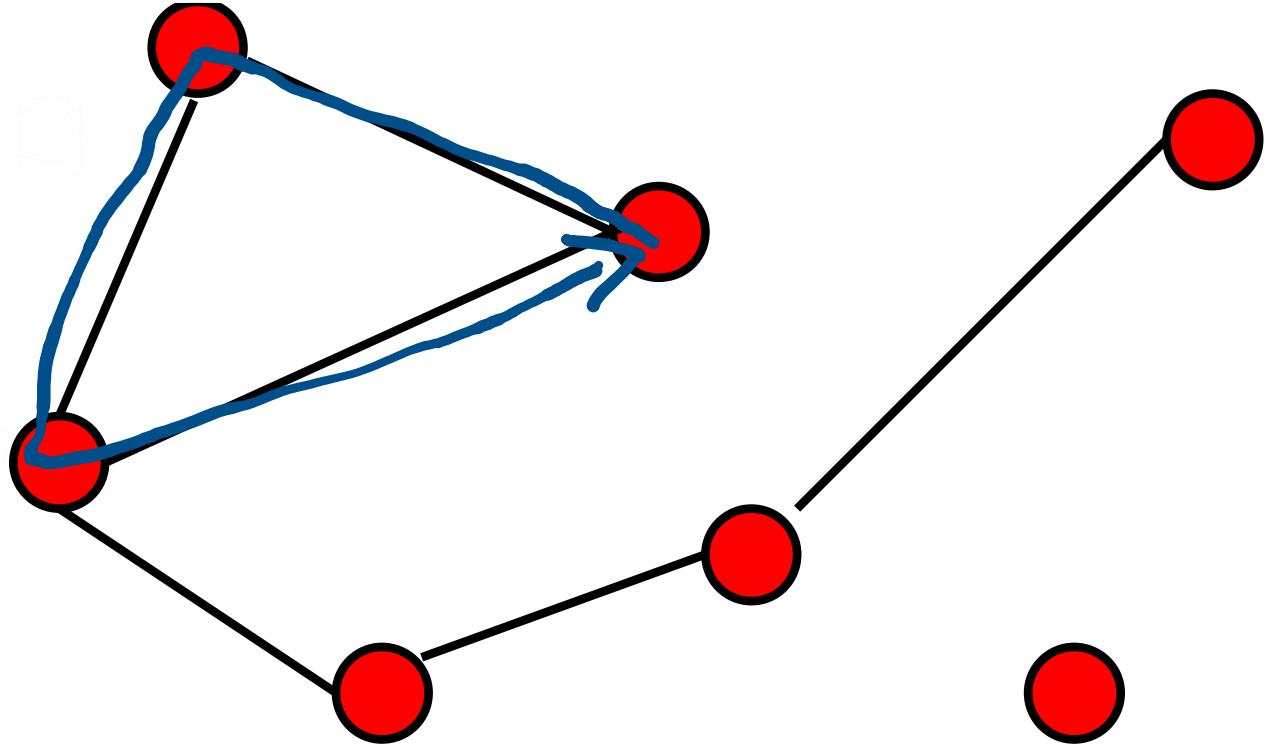
What's a path?

Definition. *A* ***path*** *is a sequence of connected vertices that are globally distinct.*



What's a cycle?

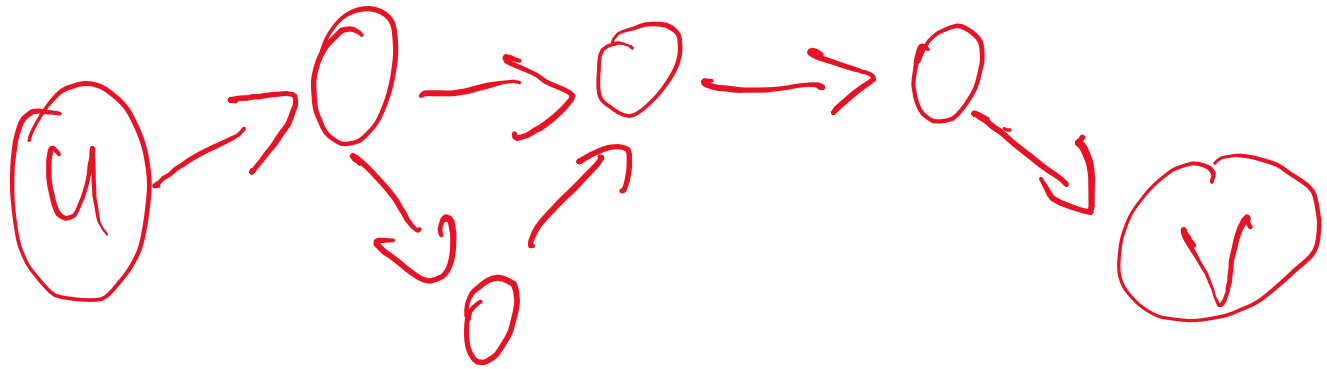
Definition. *A path that begins and end at the same point is called a* ***cycle****.*



Some definitions specific to directed graphs

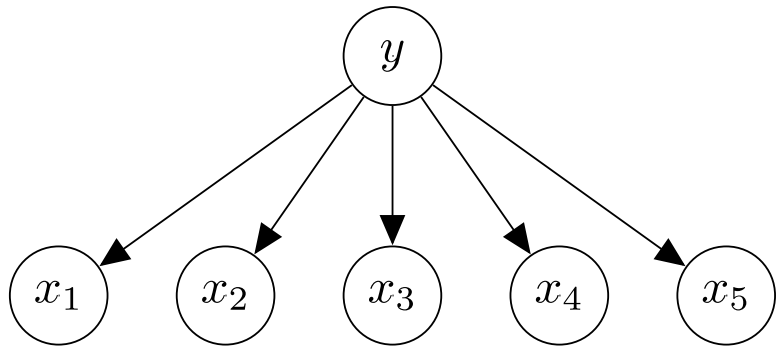
Definition. *u is a **parent** of v if $(u, v) \in E$. We also say that v is a **children** of u.*

Definition. *u is an **ancestor** of v if there exists a path from u to v. We also say that v is a **descendant** of u.*

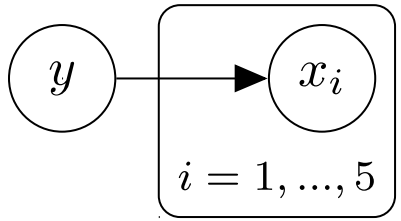


The useful plate notation

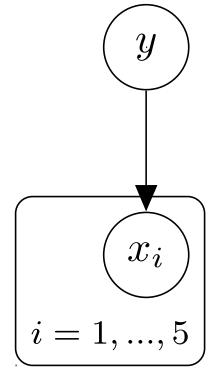
- Often we deal with graphs with recurring stucture, for instance



- The **plate notation** is a more compact way of drawing the same graph:



or



A key concept: DAGs

Definition (DAG). *A* ***directed acyclic graph*** *(DAG) is a directed graph without any cycle.*

What are directed graphical models?

- Let $G = (V, E)$ be a DAG whose nodes are denoted $V = \{1, ..., d\}$
- Let $X = (X_1, ..., X_d)$ be a random variable with density $p(x) = p(x_1, ..., x_d)$.
- Every node of the graph corresponds to a random variable (one vertex for each feature)

Definition (Directed graphical model)**.** *We say that p* ***factorises in*** *G (denoted $p \in \mathcal{L}(G)$) when, for all x,*

$$p(x) = \prod_{i=1}^{d} p(x_i | x_{\mathrm{pa}_i}),$$

where, for all node i, pa_i denotes the set of parents of node i.

2

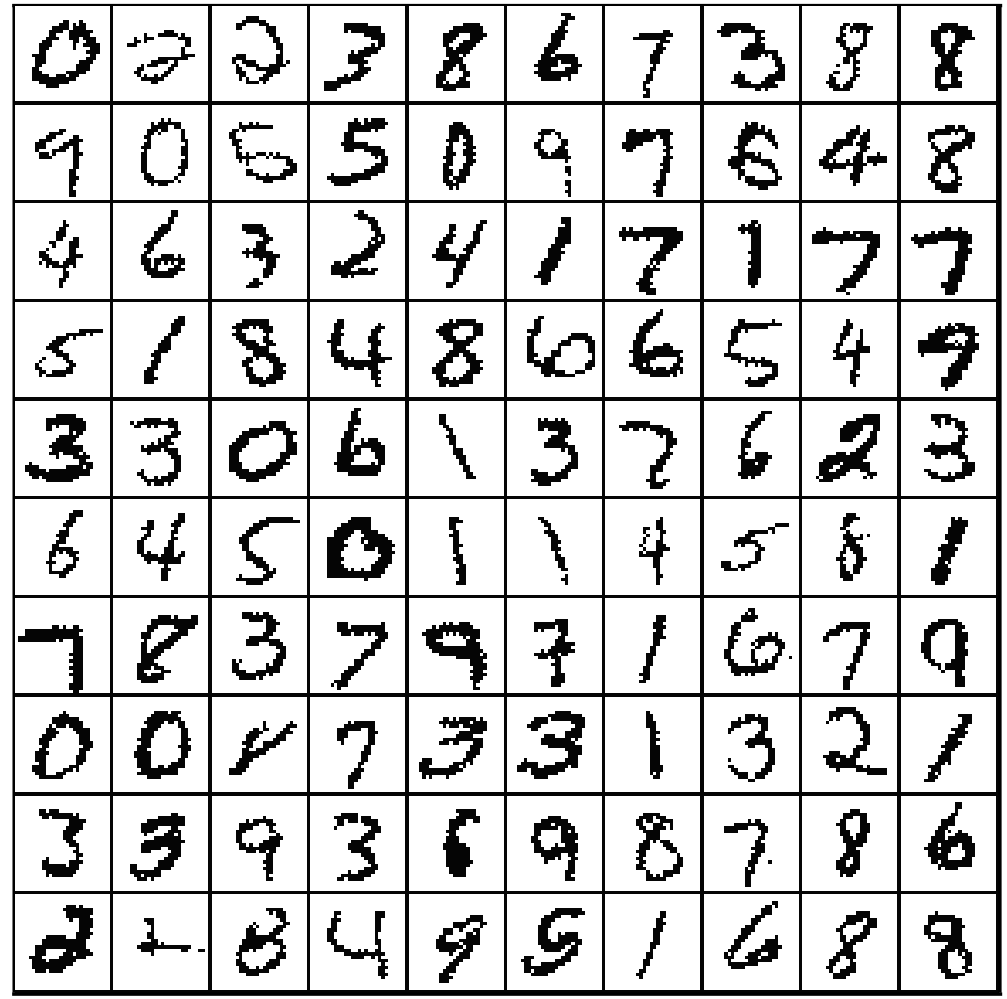
More examples of directed models and notations

Recurring « fil rouge » example

- We will often consider as a working example a **binary version of MNIST**

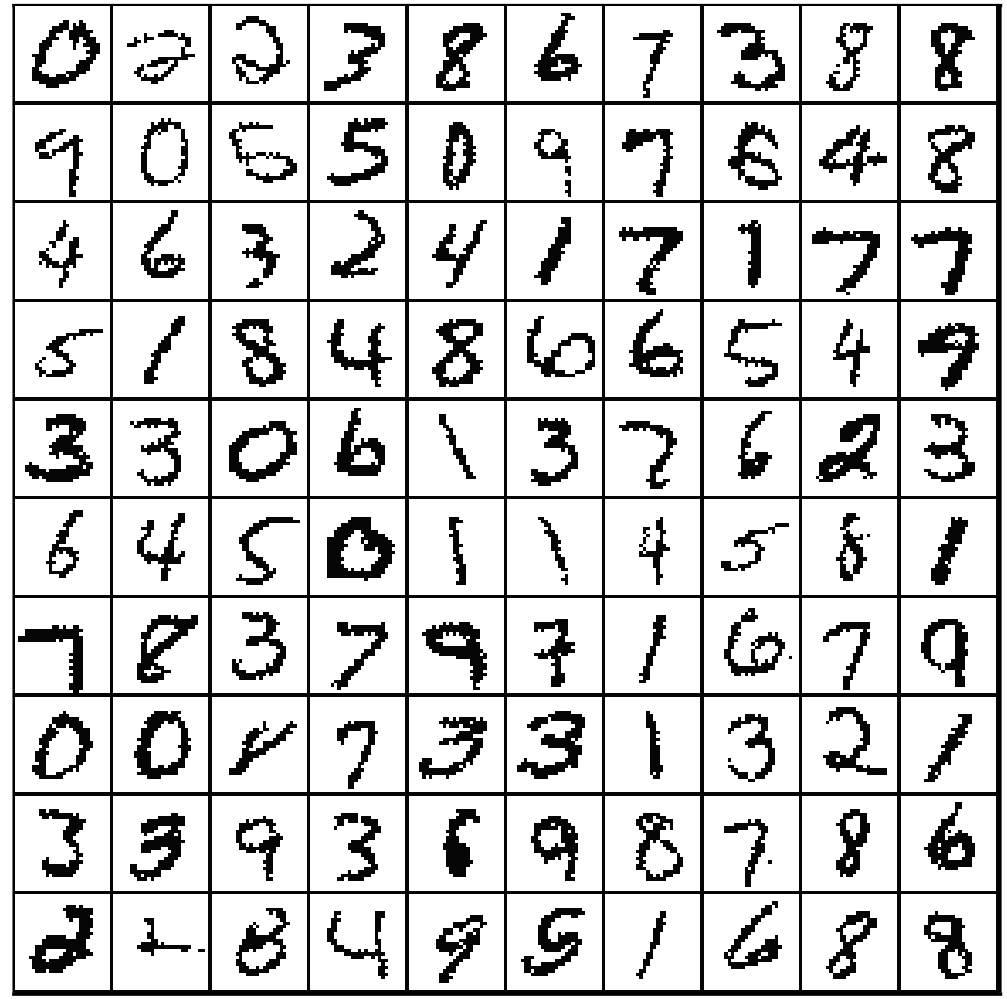
- $x_1, ..., x_n \in \mathcal{X} \quad \mathcal{X} = \{0,1\}^{28\times 28}$

- $y_1, ..., y_n \in \{0, \ldots, 9\}$



Recurring « fil rouge » example

- We will often consider as a working example a **binary version of MNIST**
- $x_1, ..., x_n \in \mathcal{X} \quad \mathcal{X} = \{0,1\}^{28\times 28}$
- $y_1, ..., y_n \in \{0, \ldots, 9\}$
- Binarised by Hugo Larochelle
- Sometimes called « statical binary MNIST » in VAE papers



A basic example of parametrised distribution for binary MNIST

- Let's go back to our binary images, that live in $\mathcal{X} = \{0, 1\}^{28 \times 28}$
- A basic model would be to assume that each pixel corresponds to a different coin flip!

A basic example of parametrised distribution for binary MNIST

- Let's go back to our binary images, that live in $\mathcal{X} = \{0,1\}^{28\times 28}$
- A basic model would be to assume that each pixel corresponds to a different coin flip!

$$x_{ij} \sim_{\text{iid}} \mathcal{B}(\theta_j) \text{ with } \theta_j \in (0,1) \text{ for } i \leq n, j \leq 28 \times 28$$

A basic example of parametrised distribution for binary MNIST

- Let's go back to our binary images, that live in $\mathcal{X} = \{0,1\}^{28\times 28}$
- A basic model would be to assume that each pixel corresponds to a different coin flip!

$$x_{ij} \sim_{\text{iid}} \mathcal{B}(\theta_j) \text{ with } \theta_j \in (0,1) \text{ for } i \leq n, j \leq 28 \times 28$$

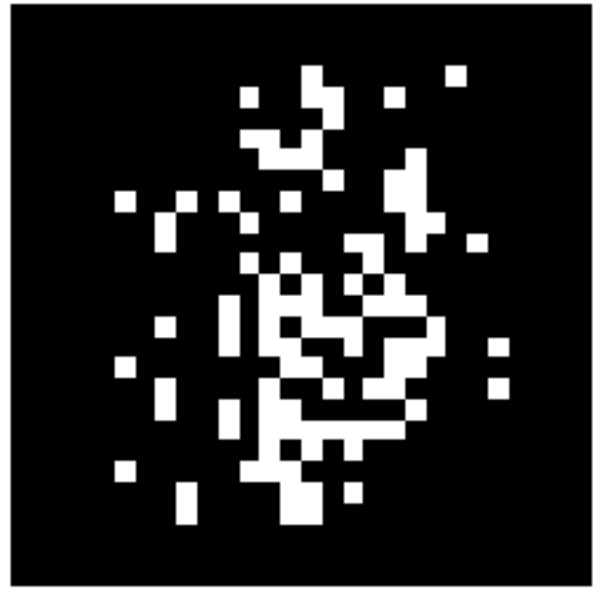
- This may be rewritten

$$p_\theta(x_1, \ldots, x_n) = \prod_{i=1}^{n} p_\theta(x_i) = \prod_{i=1}^{n} \prod_{j=1}^{28\times 28} \mathcal{B}(x_{ij}|\theta_j)$$

Why is this model not great?

The MLE is just $\hat{\theta}_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij}$

Here is a sample from the model…



Real pixels are not independent!

The graphical model for this weak baseline

The factorisation is

$$p_\theta(x_1, \ldots, x_n) = \prod_{i=1}^{n} p_\theta(x_i) = \prod_{i=1}^{n} \prod_{j=1}^{28\times 28} \mathcal{B}(x_{ij}|\theta_j)$$

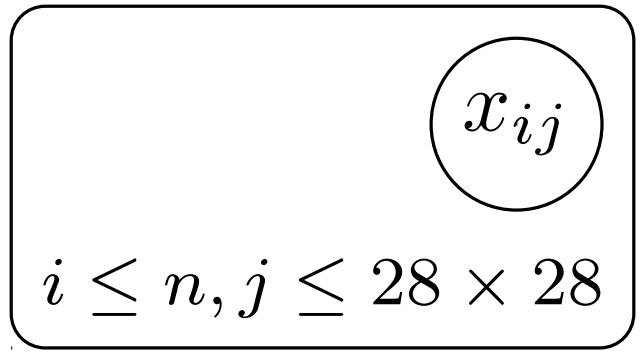
What is the graphical model?

The graphical model for this weak baseline

The factorisation is

$$p_\theta(x_1, \ldots, x_n) = \prod_{i=1}^{n} p_\theta(x_i) = \prod_{i=1}^{n} \prod_{j=1}^{28\times 28} \mathcal{B}(x_{ij}|\theta_j)$$

What is the graphical model?

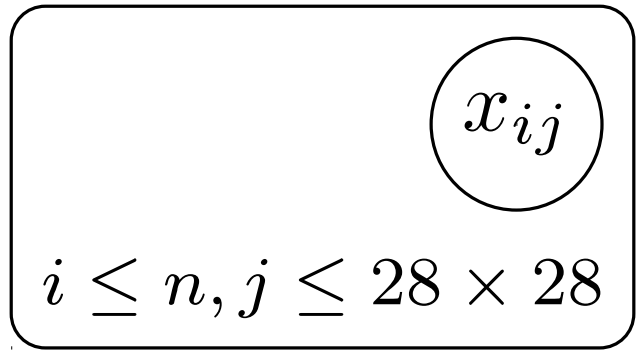


The graphical model for this weak baseline

The factorisation is

$$p_\theta(x_1, \ldots, x_n) = \prod_{i=1}^{n} p_\theta(x_i) = \prod_{i=1}^{n} \prod_{j=1}^{28\times 28} \mathcal{B}(x_{ij}|\theta_j)$$

What is the graphical model?



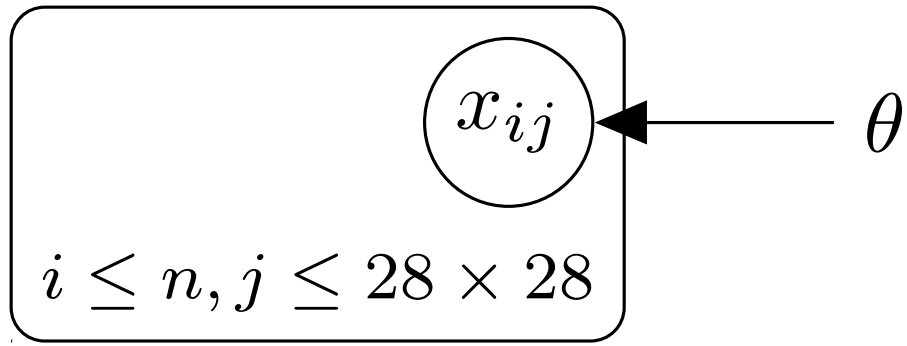
…aka “trivial graph” (because it has no edges)

The graphical model for this weak baseline

The factorisation is

$$p_\theta(x_1, \ldots, x_n) = \prod_{i=1}^{n} p_\theta(x_i) = \prod_{i=1}^{n} \prod_{j=1}^{28\times 28} \mathcal{B}(x_{ij}|\theta_j)$$

What is the graphical model?



…aka "trivial graph" (because it has no edges)

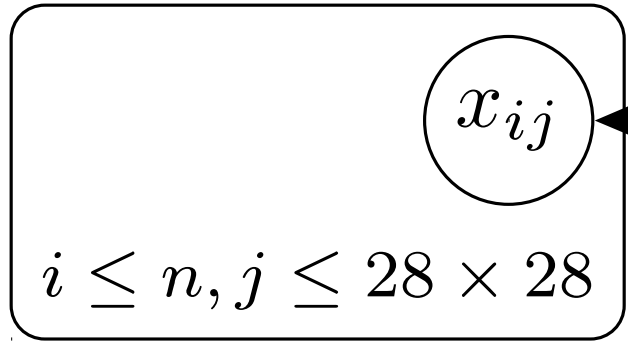
The graphical model for this weak baseline

The factorisation is

$$p_\theta(x_1, \ldots, x_n) = \prod_{i=1}^{n} p_\theta(x_i) = \prod_{i=1}^{n} \prod_{j=} \mathcal{B}($$

How do we improve this weak model?

What is the graphical model?



θ

…aka “trivial graph” (because it has no edges)

The graphical model for this weak baseline

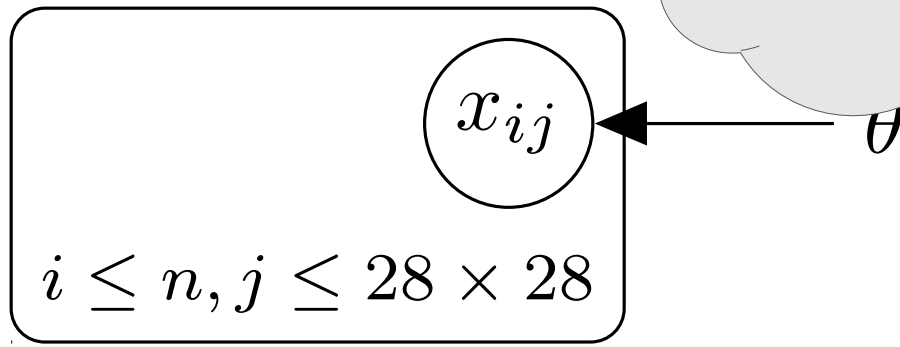
The factorisation is

$$p_\theta(x_1, \ldots, x_n) = \prod_{i=1}^{n} p_\theta(x_i) = \prod_{i=1}^{n} \prod_{j=} \mathcal{B}($$

How do we improve this weak model?

Hint: think of GMMs!

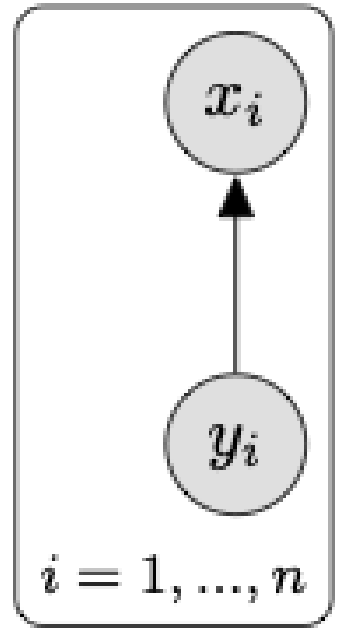
What is the graphical model?



…aka "trivial graph" (because it has no edges)

Adding a node that represents the class improves the model!

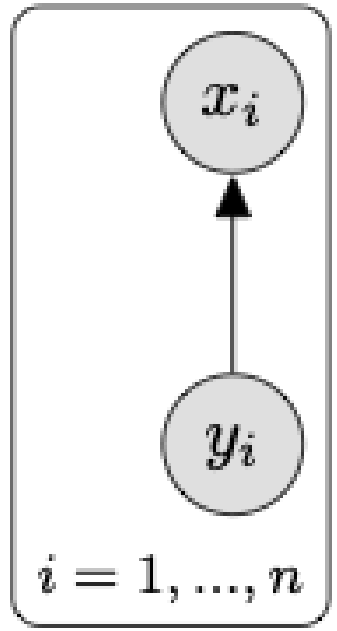
$$p_\theta(x_1, \ldots, x_n, y_1, \ldots, y_n) = \prod_{i=1}^{n} p_\theta(y_i) p_\theta(x_i|y_i) \text{ with } p_\theta(y) = \mathcal{M}(y|1, \pi) \text{ and } p_\theta(x|y) = \mathcal{B}(x|\mu_{yj})$$



$$\theta = \left(\pi, (\mu_{yj})_{y \in \{0,\ldots,9\}, j \leq 784}\right)$$

Adding a node that represents the class improves the model!

$$p_\theta(x_1, \ldots, x_n, y_1, \ldots, y_n) = \prod_{i=1}^{n} p_\theta(y_i) p_\theta(x_i|y_i) \text{ with } p_\theta(y) = \mathcal{M}(y|1, \pi) \text{ and } p_\theta(x|y) = \mathcal{B}(x|\mu_{yj})$$

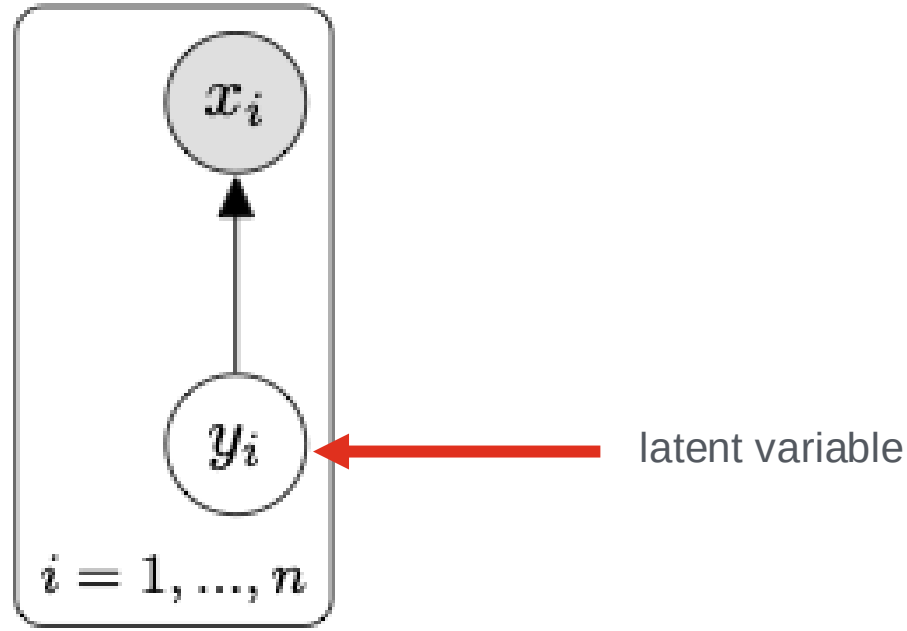


But this is a supervised model… How to make it unsupervised?

$$\theta = \left(\pi, (\mu_{yj})_{y \in \{0,\ldots,9\}, j \leq 784}\right)$$

Latent class analysis (aka mixtures of products of Bernoullis)

$$p_\theta(x_1, \ldots, x_n, y_1, \ldots, y_n) = \prod_{i=1}^{n} p_\theta(y_i) p_\theta(x_i|y_i) \text{ with } p_\theta(y) = \mathcal{M}(y|1, \pi) \text{ and } p_\theta(x|y) = \mathcal{B}(x|\mu_{yj})$$



$$\theta = \left(\pi, (\mu_{yj})_{y \in \{0,\ldots,9\}, j \leq 784}\right)$$

Two very similar models, two very different likelihoods…

The likelihood of the supervised model is just a sum of log-densities of simple distributions:

$$\log p_\theta(x_1, \ldots, x_n, y_1, \ldots, y_n) = \sum_{i=1}^{n} (\log p_\theta(y_i) + \log p_\theta(x_i | y_i))$$

This is very easy to maximise, it has a simple closed-form solution.

Two very similar models, two very different likelihoods…

The likelihood of the supervised model is just a sum of log-densities of simple distributions:

$$\log p_\theta(x_1, \ldots, x_n, y_1, \ldots, y_n) = \sum_{i=1}^{n} (\log p_\theta(y_i) + \log p_\theta(x_i|y_i))$$

This is very easy to maximise, it has a simple closed-form solution.

On the other hand, **the likelihood of the unsupervised model is way trickier**

$$\log p_\theta(x_1, \ldots, x_n) = \sum_{i=1}^{n} \log \left(\sum_{y} p_\theta(y) p_\theta(x_i|y) \right)$$

Two very similar models, two very different likelihoods…

The likelihood of the supervised model is just a sum of log-densities of simple distributions:

$$\log p_\theta(x_1, \ldots, x_n, y_1, \ldots, y_n) = \sum_{i=1}^{n} (\log p_\theta(y_i) + \log p_\theta(x_i|y_i))$$

This is very easy to maximise, it has a simple closed-form solution.

On the other hand, **the likelihood of the unsupervised model is way trickier**

$$\log p_\theta(x_1, \ldots, x_n) = \sum_{i=1}^{n} \log \left(\sum_{y} p_\theta(y) p_\theta(x_i|y) \right)$$

What would be ways to maximise it?

Two very similar models, two very different likelihoods…

The likelihood of the supervised model is just a sum of log-densities of simple distributions:

$$\log p_\theta(x_1, \ldots, x_n, y_1, \ldots, y_n) = \sum_{i=1}^{n} (\log p_\theta(y_i) + \log p_\theta(x_i|y_i))$$

This is very easy to maximise, it has a simple closed-form solution.

On the other hand, **the likelihood of the unsupervised model is way trickier**

$$\log p_\theta(x_1, \ldots, x_n) = \sum_{i=1}^{n} \log \left(\sum_{y} p_\theta(y) p_\theta(x_i|y) \right)$$

What would be ways to maximise it? EM, gradient descent, etc.

3

Technical aparté 1

Sampling with graphical models: ancestral sampling

Why sampling from a graphical model

1. The samples are useful by themselves

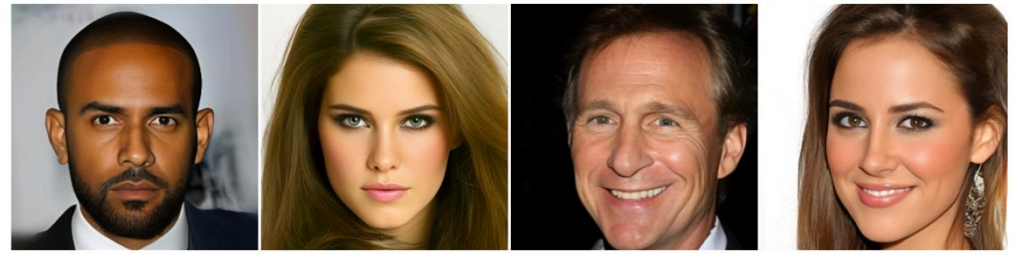
Images generated by a GAN

Brock, Donahue, and Simonyan (ICLR 2019)

2. Samples allow us to appximate integrals using Monte Carlo

Why sampling from a graphical model

1. The samples are useful by themselves



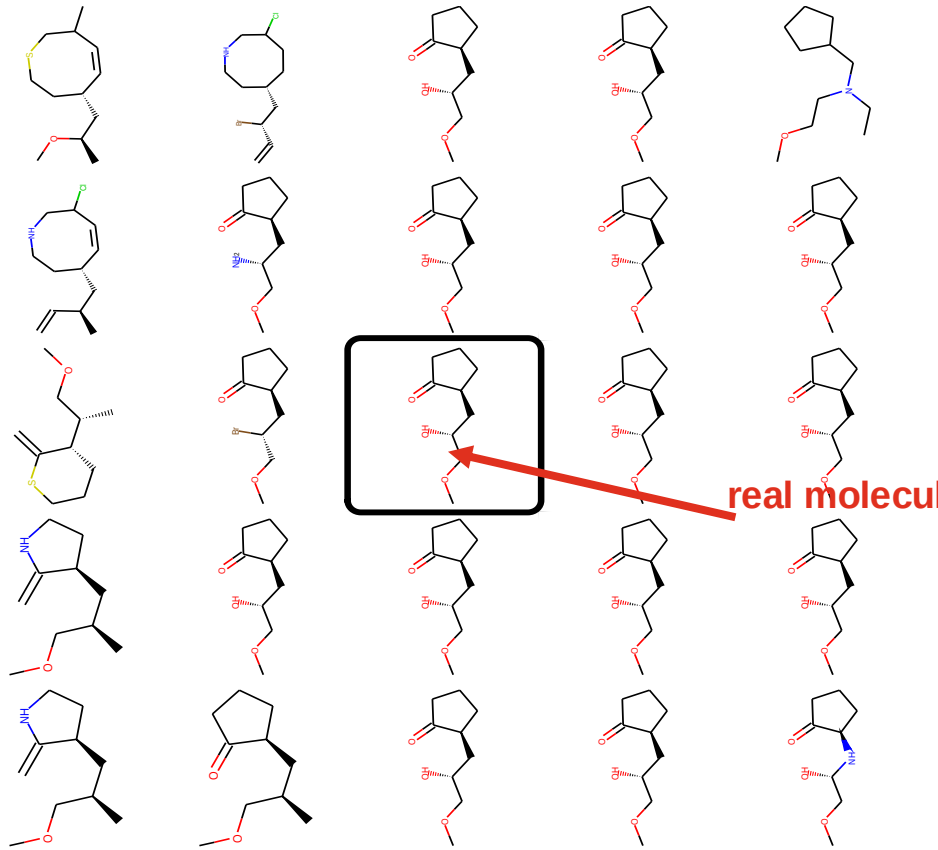
Images generated by a VAE

Vahdat and Kautz (NeurIPS 2020)

2. Samples allow us to appximate integrals using Monte Carlo

Why sampling from a graphical model

1. The samples are useful by themselves

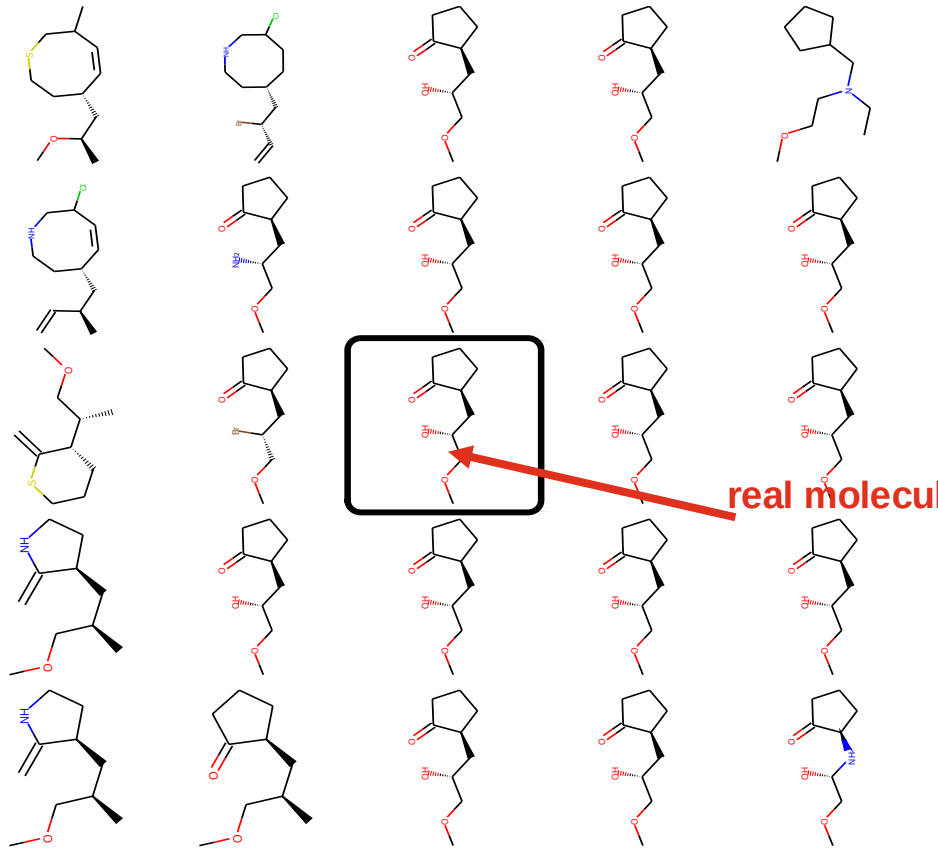


Kusner, Paige, and Hernández-Lobato (ICML 2017)

2. Samples allow us to appximate integrals using Monte Carlo

Why sampling from a graphical model

1. The samples are useful by themselves



Kusner, Paige, and Hernández-Lobato (ICML 2017)

2. Samples allow us to appximate integrals using Monte Carlo

$$\int f(x)p(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

Monte Carlo basics

- Until now, we have written integrals as sums that we can compute exhaustively, but this is quite rarely the case.

- When this integral is an expected value over something we know how to sample from, we can use **simple Monte Carlo**

Algorithm 1 Simple Monte Carlo

1: Draw $X^{(1)}, ..., X^{(n)} \overset{i.i.d.}{\sim} p$
2: $\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} f(X^{(i)})$

- Why are we expecting this to work?

Monte Carlo basics

Algorithm 1 Simple Monte Carlo

1: Draw $X^{(1)}, ..., X^{(n)} \overset{i.i.d.}{\sim} p$
2: $\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} f(X^{(i)})$

Proposition (Law of Large Numbers)**.**

$$\hat{\mu} \xrightarrow{a.s.} \mu \quad \textit{iff} \quad \mu \textit{ is finite}$$

Proposition (Central Limit Theorem)**.** *If* $\mathbb{V}ar(f(X)) = \sigma^2 < \infty$*, then*

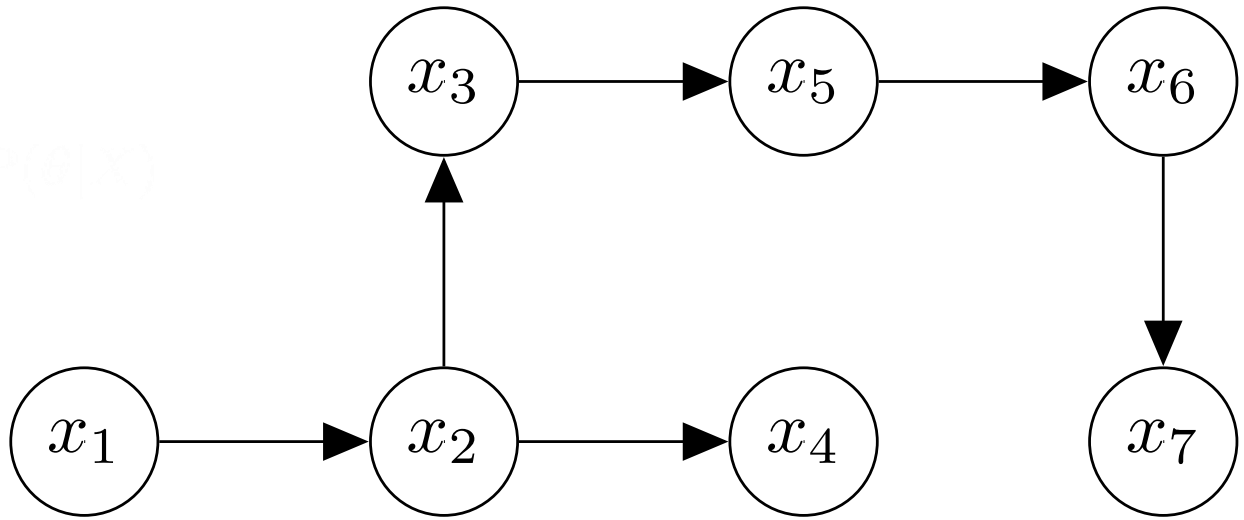
$$\sqrt{n}(\hat{\mu} - \mu) \xrightarrow{\mathcal{D}} \mathcal{N}(0, \sigma^2)$$

How to sample from a directed graph?

Algorithm 1 Ancestral sampling with topological ordering

for $i = 1$ to d **do**
  Draw z_i from $p(X_i = .|X_{\mathrm{pa}_i} = z_{\mathrm{pa}_i})$
end for
  return $(z_1, \ldots, z_d)$

We follow the ordering, sampling one descendant after the other…

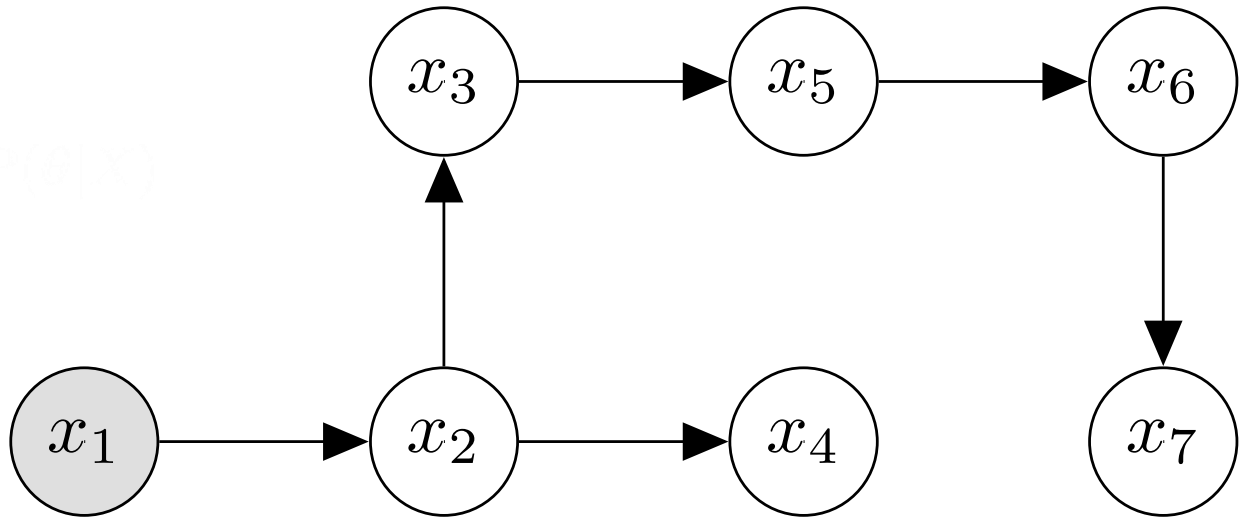


How to sample from a directed graph?

Algorithm 1 Ancestral sampling with topological ordering

for $i = 1$ to d **do**
  Draw z_i from $p(X_i = .|X_{\mathrm{pa}_i} = z_{\mathrm{pa}_i})$
end for
  return $(z_1, \ldots, z_d)$

We follow the ordering, sampling one descendant after the other…

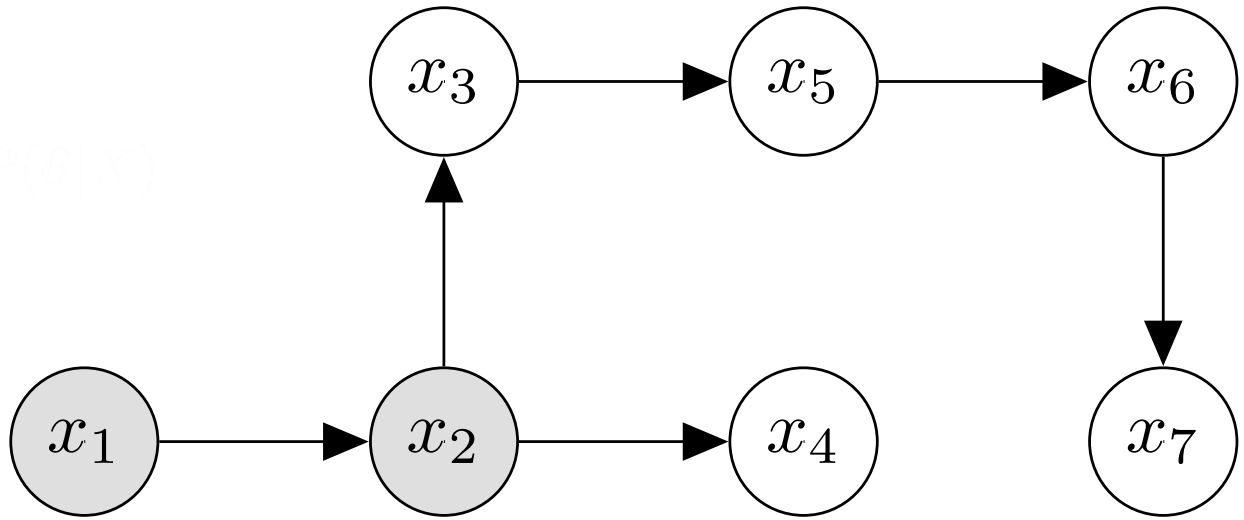


How to sample from a directed graph?

Algorithm 1 Ancestral sampling with topological ordering

for $i = 1$ to d **do**
  Draw z_i from $p(X_i = .|X_{\mathrm{pa}_i} = z_{\mathrm{pa}_i})$
end for
  return $(z_1, \ldots, z_d)$

We follow the ordering, sampling one descendant after the other…

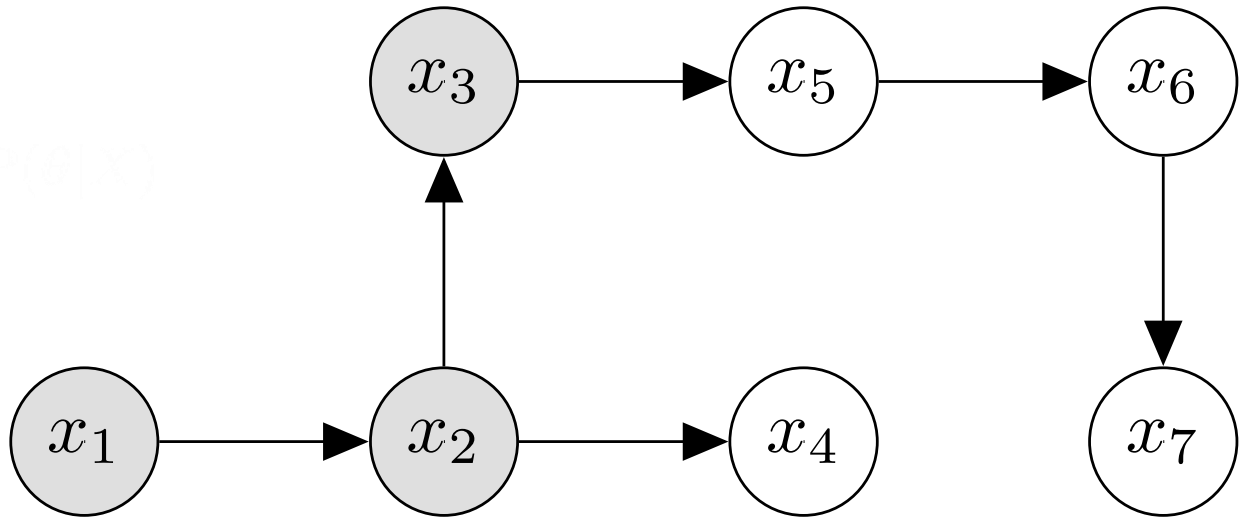


How to sample from a directed graph?

Algorithm 1 Ancestral sampling with topological ordering

for $i = 1$ to d **do**
  Draw z_i from $p(X_i = .|X_{\mathrm{pa}_i} = z_{\mathrm{pa}_i})$
end for
  return $(z_1, \ldots, z_d)$

We follow the ordering, sampling one descendant after the other…

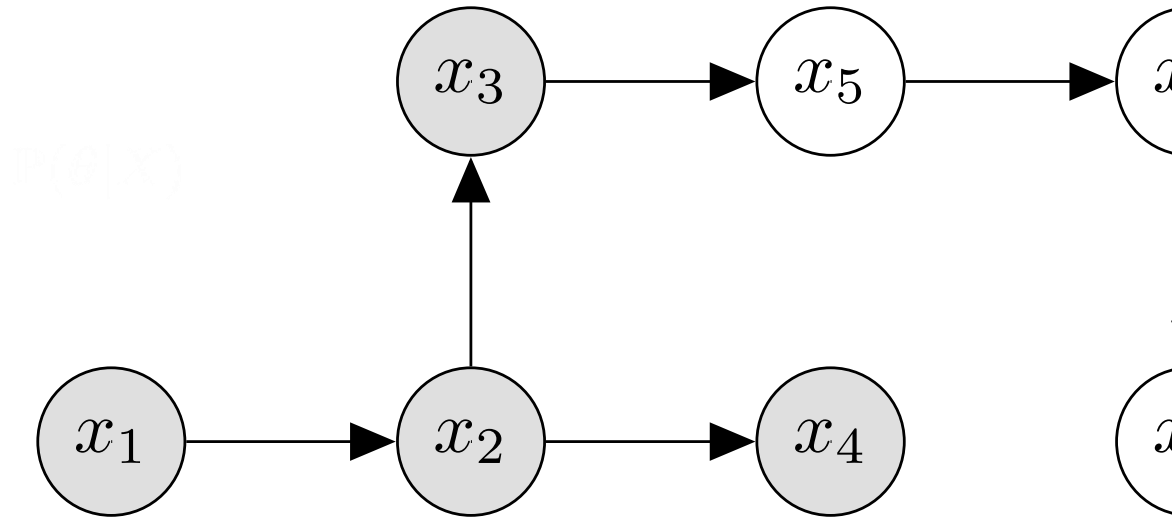


How to sample from a directed graph?

Algorithm 1 Ancestral sampling with topological ordering

for $i = 1$ to d **do**
  Draw z_i from $p(X_i = .|X_{\mathrm{pa}_i} = z_{\mathrm{pa}_i})$
end for
  return $(z_1, \ldots, z_d)$

We follow the ordering, sampling one descendant after the other…

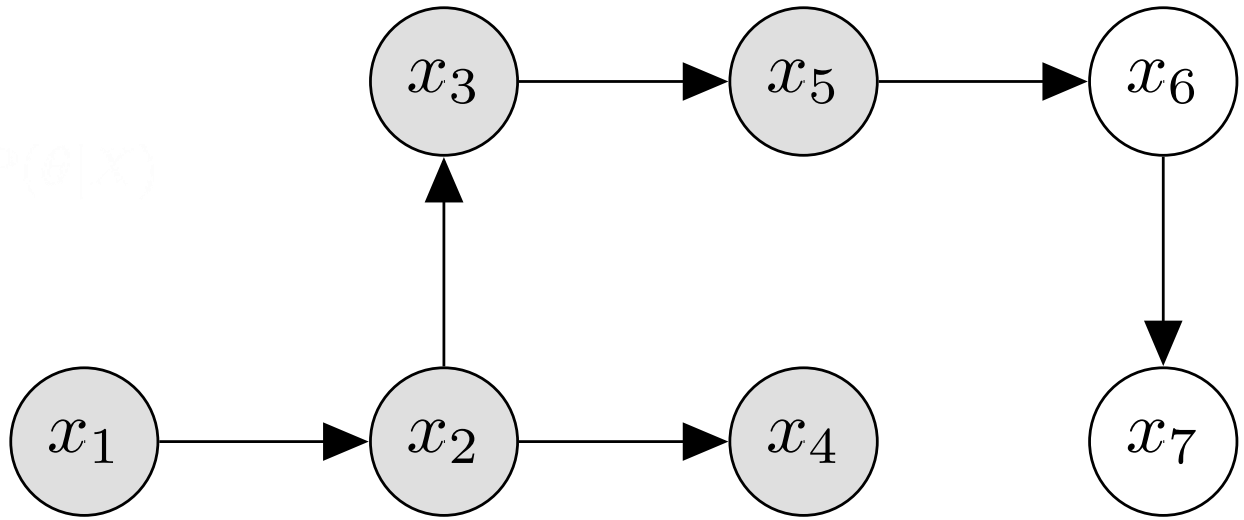


How to sample from a directed graph?

Algorithm 1 Ancestral sampling with topological ordering

for $i = 1$ to d **do**
  Draw z_i from $p(X_i = .|X_{\mathrm{pa}_i} = z_{\mathrm{pa}_i})$
end for
  return $(z_1, \ldots, z_d)$

We follow the ordering, sampling one descendant after the other…

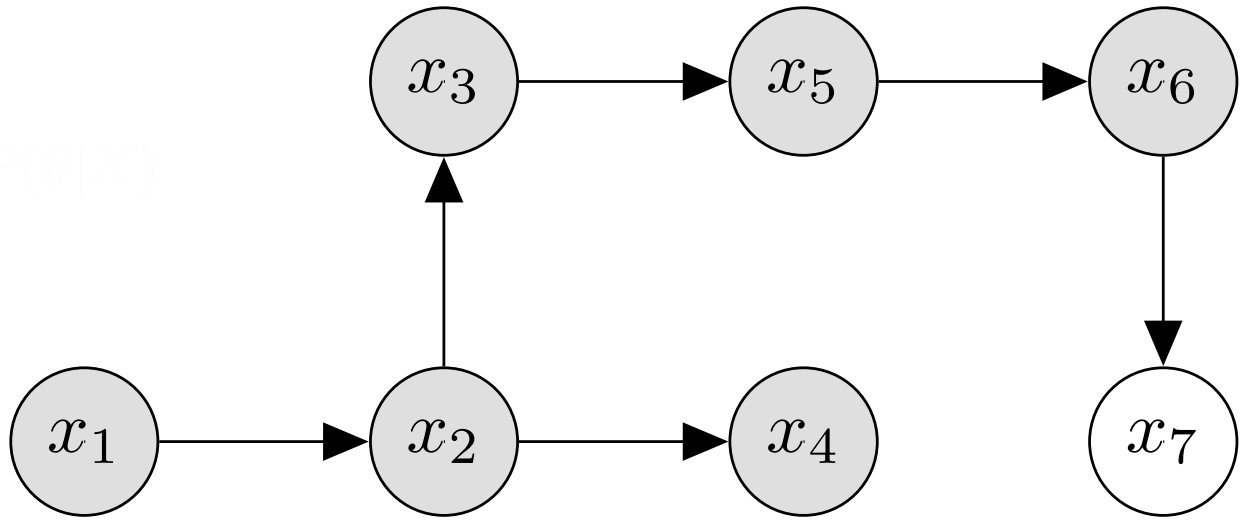


How to sample from a directed graph?

Algorithm 1 Ancestral sampling with topological ordering

for $i = 1$ to d **do**
  Draw z_i from $p(X_i = .|X_{\mathrm{pa}_i} = z_{\mathrm{pa}_i})$
end for
  return $(z_1, \ldots, z_d)$

We follow the ordering, sampling one descendant after the other…

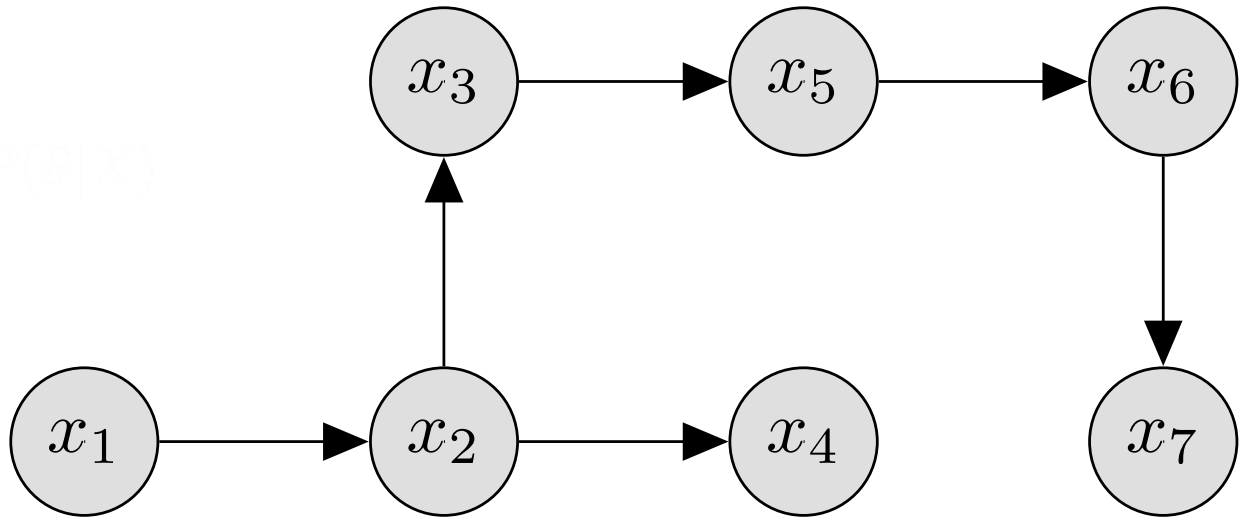


How to sample from a directed graph?

Algorithm 1 Ancestral sampling with topological ordering

for $i = 1$ to d **do**
  Draw z_i from $p(X_i = .|X_{\mathrm{pa}_i} = z_{\mathrm{pa}_i})$
end for
  return $(z_1, \ldots, z_d)$

We follow the ordering, sampling one descendant after the other…

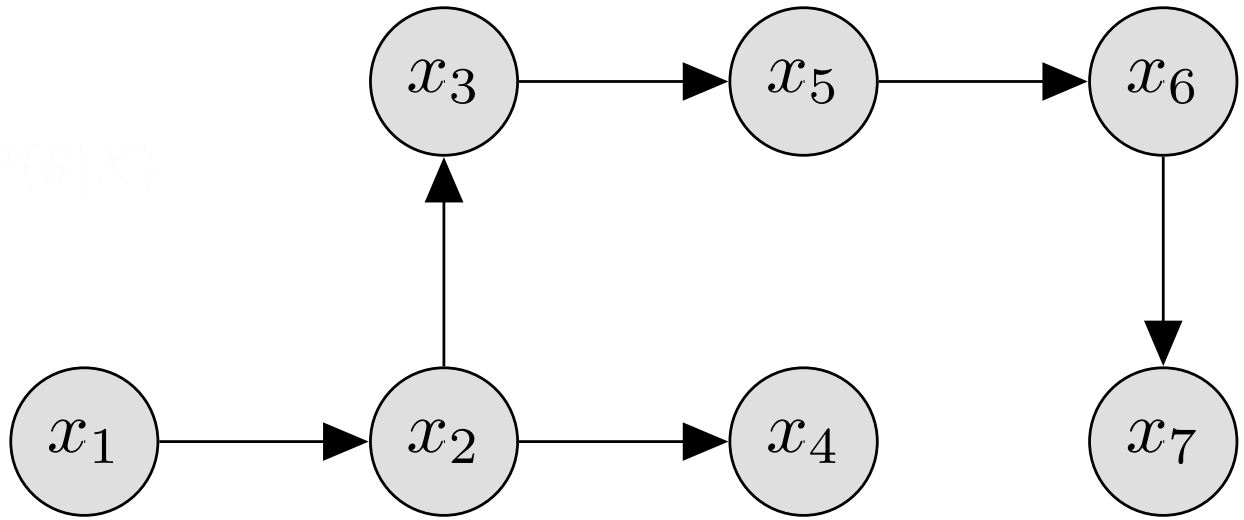


How to sample from a directed graph?

Algorithm 1 Ancestral sampling with topological ordering

for $i = 1$ to d **do**
  Draw z_i from $p(X_i = .|X_{\mathrm{pa}_i} = z_{\mathrm{pa}_i})$
end for
  return $(z_1, \ldots, z_d)$

Of course, we could have chosen **any topological ordering!**

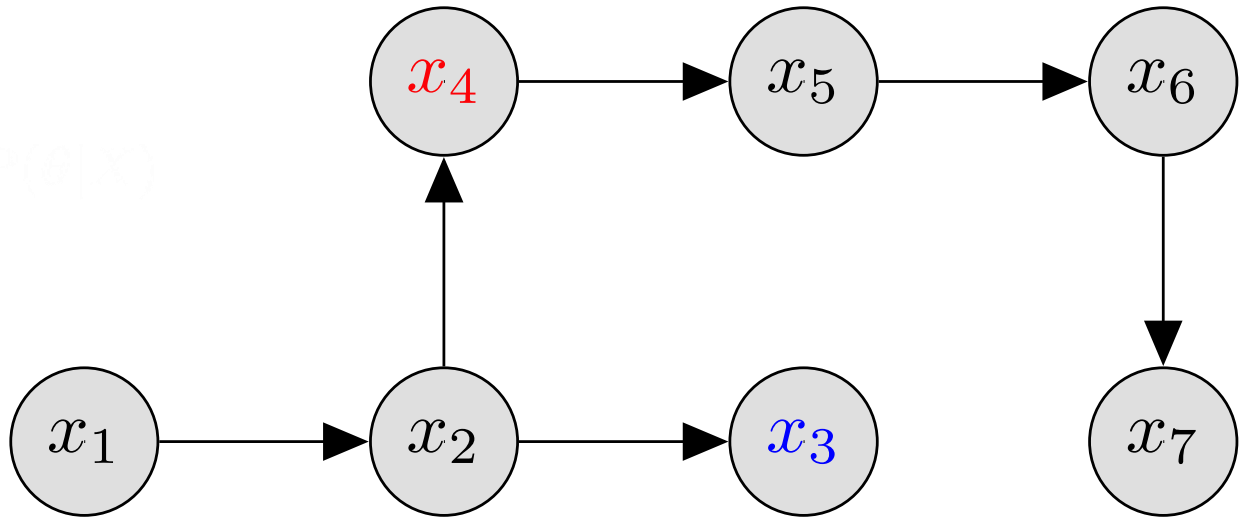


How to sample from a directed graph?

Algorithm 1 Ancestral sampling with topological ordering

for $i = 1$ to d **do**
  Draw z_i from $p(X_i = .|X_{\mathrm{pa}_i} = z_{\mathrm{pa}_i})$
end for
  return $(z_1, \ldots, z_d)$

Of course, we could have chosen **any topological ordering!**



How to sample from a directed graph?

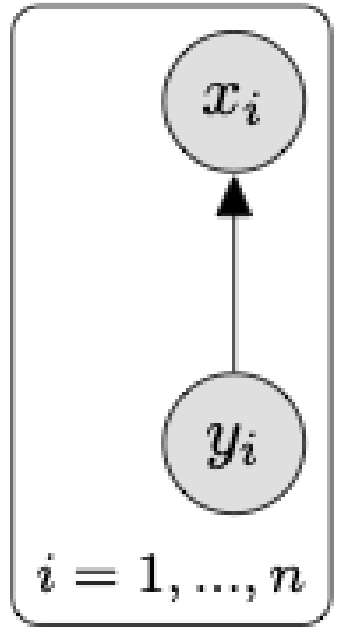
Algorithm 1 Ancestral sampling with topological ordering

for $i = 1$ to d **do**
  Draw z_i from $p(X_i = .|X_{\mathrm{pa}_i} = z_{\mathrm{pa}_i})$
end for
  return $(z_1, \ldots, z_d)$

Proposition (ancestral sampling)**.** *Algorithm 1 provides samples from the joint distribution* $p(x_1, ..., x_d)$*.*

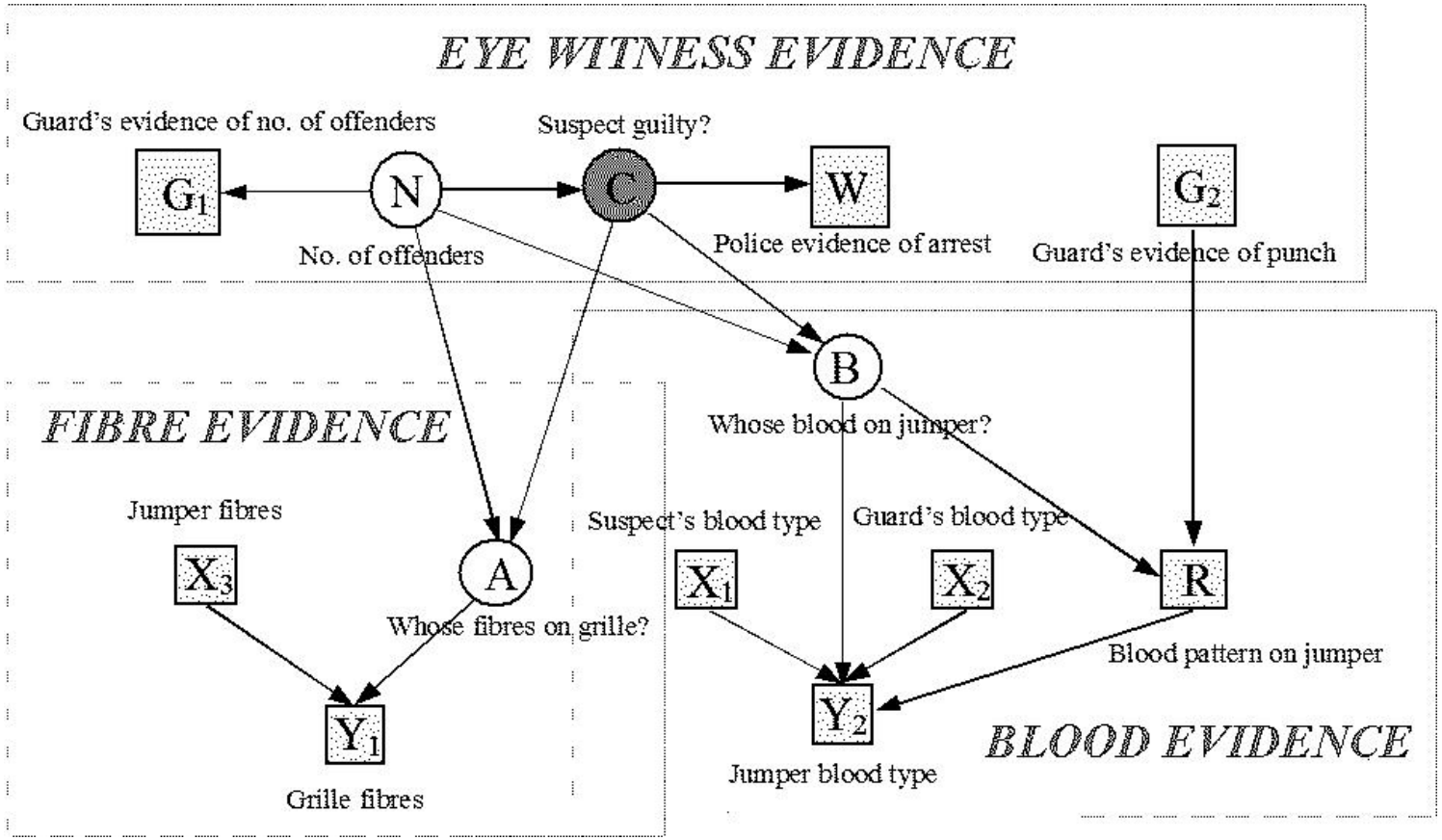
Proof. By induction on the topological ordering. □

Think of ways to implement this for some of the graphs we saw



Think of ways to implement this for some of the graphs we saw

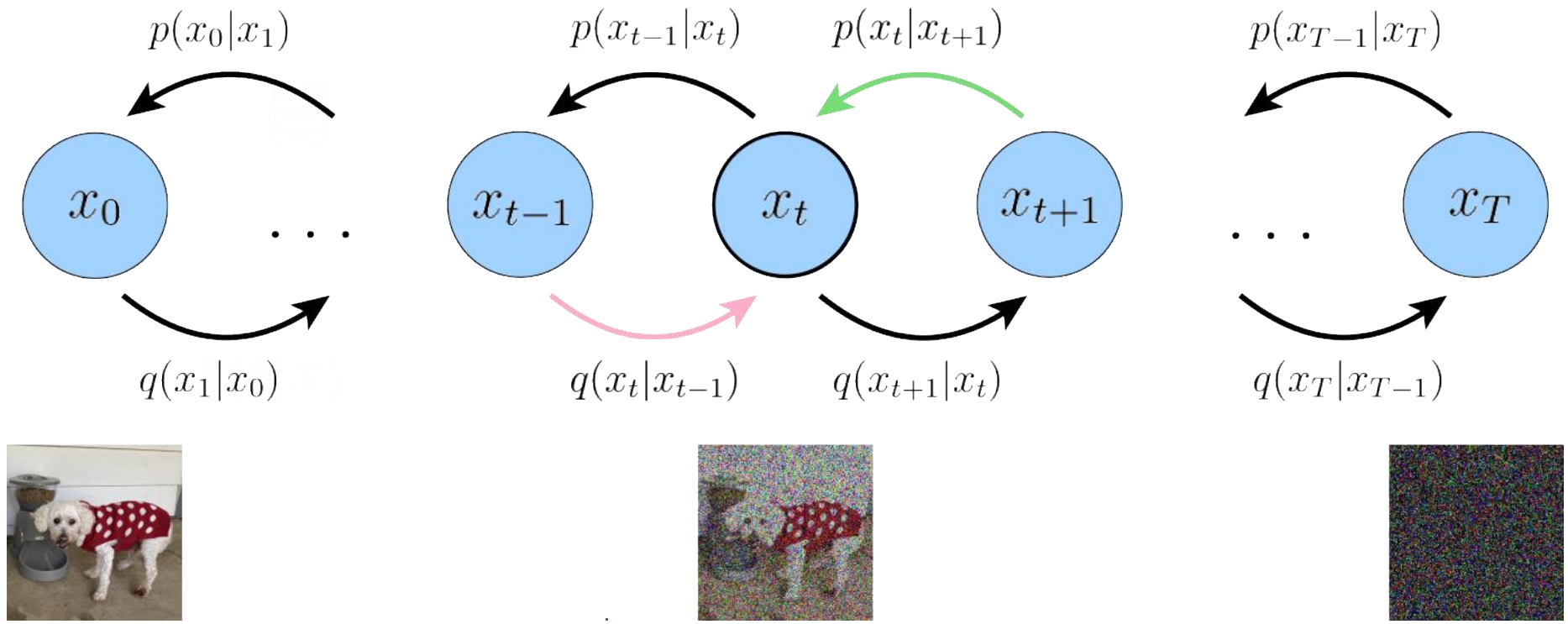
- The big graph from the forensic science example… Here the topological ordering is related to the time of the events…



Dawid and Evett, Using a graphical method to assist the evaluation of complicated patterns of evidence. Journal of Forensic Sciences, 1997

Think of ways to implement this for some of the graphs we saw

- Diffusion model (again, two DAGs!)



Luo, Understanding Diffusion Models: A Unified Perspective, arXiv preprint, 2022

4

Linear latent variable models
PPCA and friends

Why latent-variable models?

Let's assume that we have i.i.d. data $\mathbf{x}_1, \ldots, \mathbf{x}_n$ 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** $\mathbf{z}_1, \ldots, \mathbf{z}_n$ **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. $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^p$).

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 a slightly less general model

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

Linear latent-variable models

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

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 a slightly less general model

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

- 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

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

...into "something deep"?

From linear (« shallow ») to deep variable models

How do we transform this kind of model

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

...into "something deep"?

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

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).

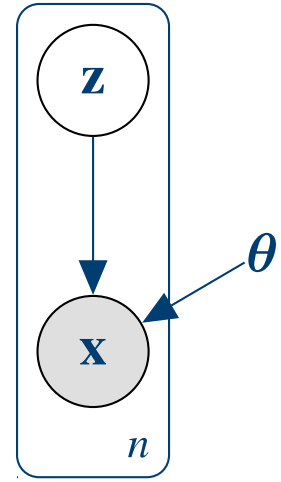
4

Deep latent variable models

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:

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



where

- $\mathbf{z} \in \mathbb{R}^d$ is the **latent** variable,
- $\mathbf{x} \in \mathcal{X}$ is the **observed** variable.

  - the function $f_{\boldsymbol{\theta}} : \mathbb{R}^d \to H$ is a **(deep) neural network** called the **decoder**
  - $(\Phi(\cdot \mid \boldsymbol{\eta}))_{\boldsymbol{\eta} \in H}$ 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).

Deep latent variable models: the role of the observation model

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

Its parameters are the output of the decoder.

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

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

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

Deep latent variable models: back to PPCA

$$p(z) = \mathcal{N}(z|0, I_Q)$$

$$p_\theta(x|z) = \mathcal{N}(x|Wz + \mu, \sigma^2 I_D)$$

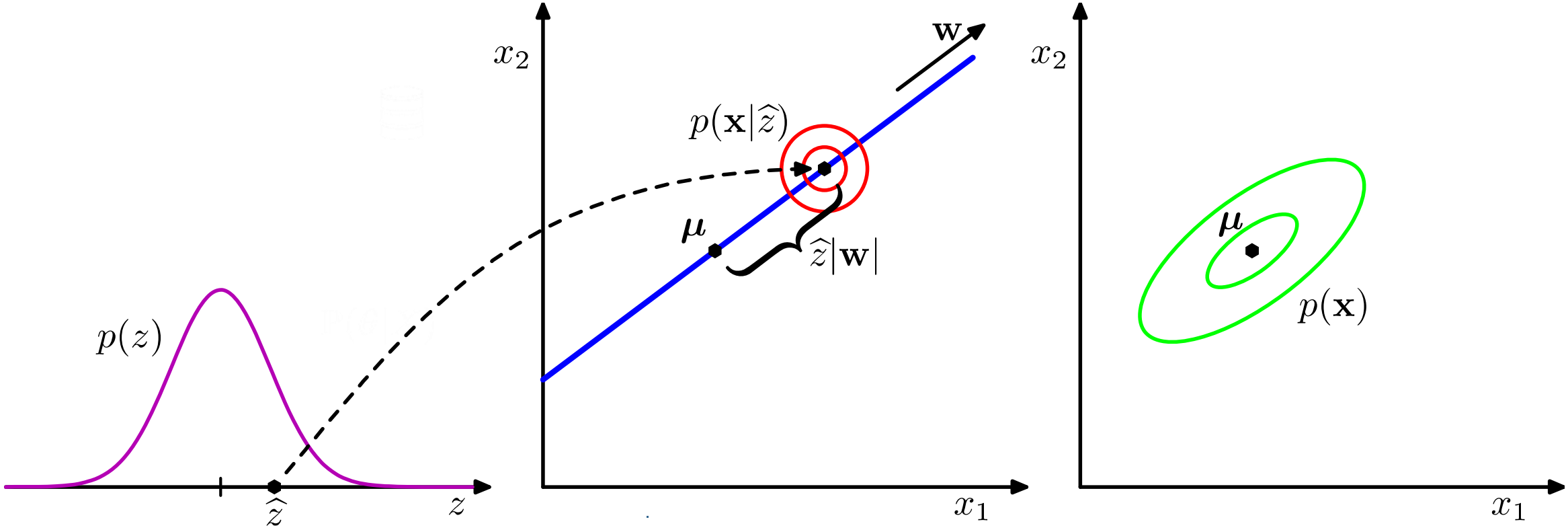


Fig. 12.9 in Bishop's Pattern Recognition and Machine Learning

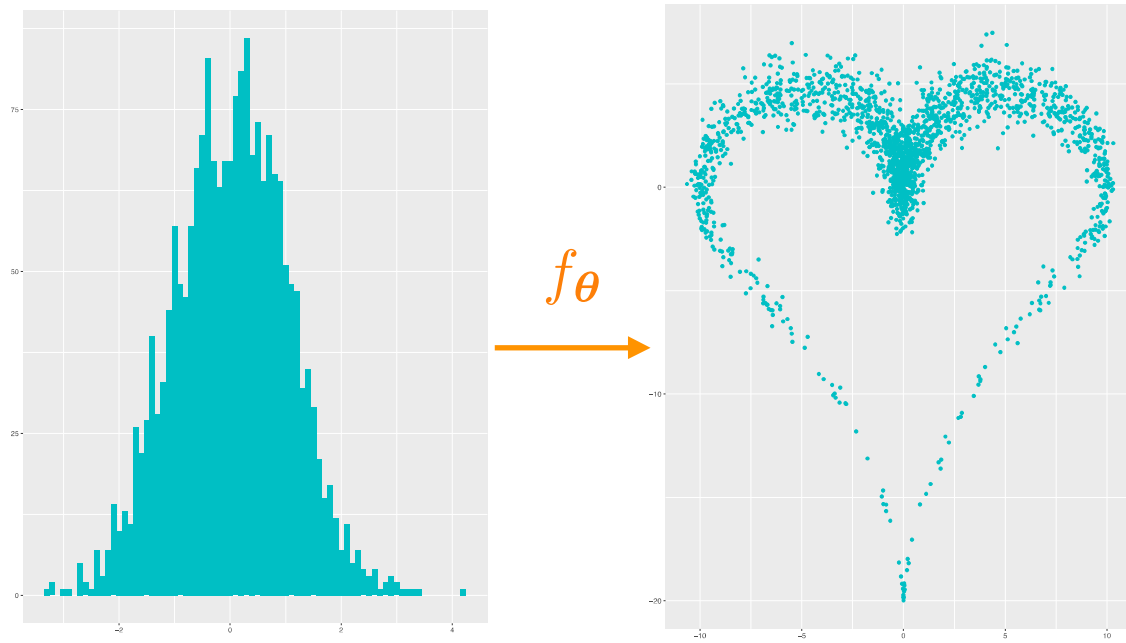
Deep latent variable models: the role of the decoder

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

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

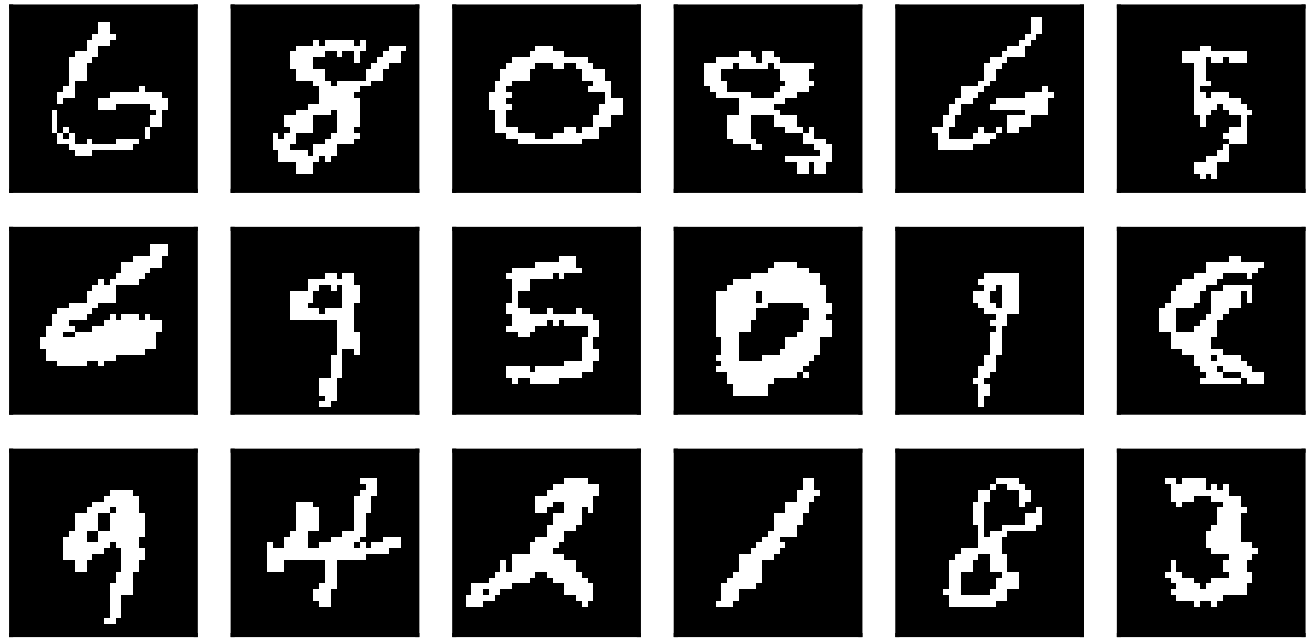
Simple non-linear decoder **($d = 1$, $p = 2$)**: $f_{\boldsymbol{\theta}}(z) = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{z}), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\mathbf{z})$ with, for all $z \in \mathbb{R}$,

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



Illustrative example of a DLVM for binary images

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



Illustrative example of a DLVM for binary images

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

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

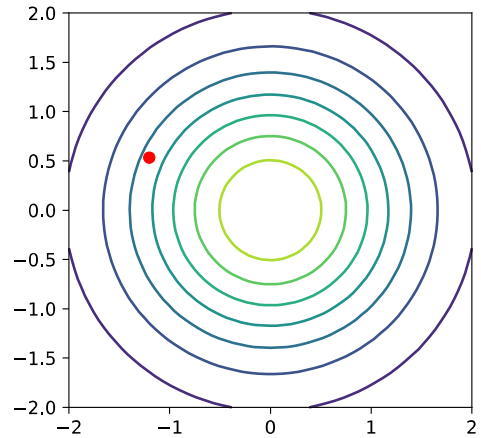
Decoder network

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

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 $\mathbf{x} \in \{0,1\}^{28\times 28}$

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

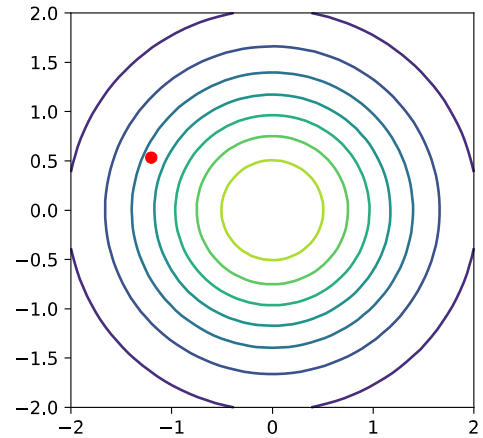
Decoder network

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

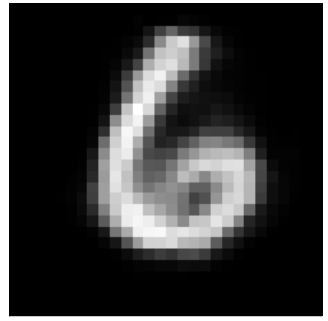
Generation

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

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



$f(\mathbf{z})$



Illustrative example of a DLVM for binary images

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

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

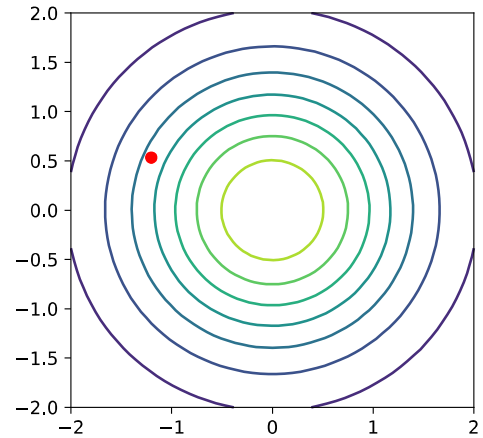
Decoder network

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

Generation

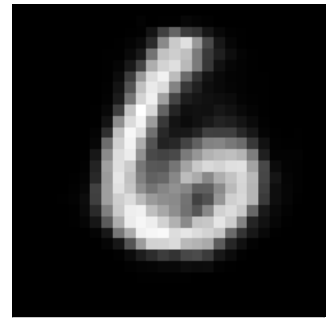
$$\mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right)$$

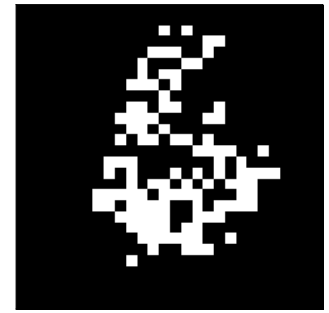
$$\mathbf{z} = (-1.2033, 0.5340)$$



$f(\mathbf{z})$

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





Illustrative example of a DLVM for binary images

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

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

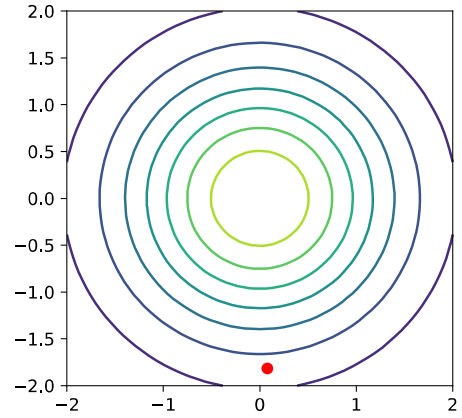
Decoder network

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

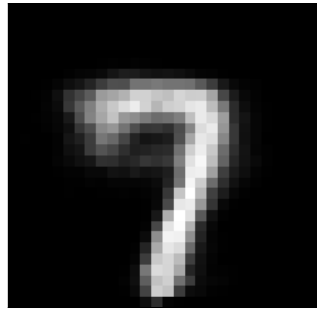
Generation

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

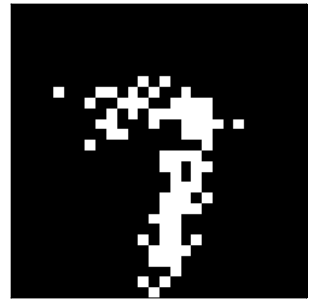
$$\mathbf{z} = (0.0791, -1.8165)$$



$f(z)$



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



Maximum likelihood for DLVM

Given a data matrix $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\intercal \in \mathcal{X}^n$, 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

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

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

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

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

Aparté: Beyond simple MC: importance sampling

- We have to use Monte Carlo! Our goal is to approximate an integral

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

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

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

Beyond simple MC: importance sampling

- We have to use Monte Carlo! Our goal is to approximate an integral

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

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

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

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

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 $\mathbb{V}[\hat{I}_K] = \frac{1}{K}\mathbb{V}[f(x_1)]$

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 $\mathbb{V}[f(x_1)]$
- **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**

$$x_1, \ldots, .x_K \sim q$$

- 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*?**

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**

$$x_1, \ldots, .x_K \sim q$$

- 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!**

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

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

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?

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 aslo be unbiased, consistent, and asymptotically normal.
- The variance will be $\mathbb{V}_{x\sim q}[f(x)p(x)/q(x)]/K$

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 aslo 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 exaclty 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

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

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

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

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

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

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

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

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** $q_1, \ldots, q_n$ (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 $q_1, \ldots, q_n$?

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 $q_1, \ldots, q_n$?

$$q_i^*(\mathbf{z}) = p_\theta(\mathbf{z}|\mathbf{x}_i)$$

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 γ:**

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

How to parametrise this conditional distribution? The key idea is that **its parameters are the output of a neural net g_γ:**

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

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

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

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

Maximum likelihood for DLVM

All of this leads to the following approximation of the likelihood

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

Rather than maximising $\ell(\boldsymbol{\theta})$, **we'll maximise $\mathcal{L}_K(\theta, \gamma)$ 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 $\ell(\theta)$** (exercise !)
- The bounds get **tighter and tighter!**

$$\mathcal{L}_1(\boldsymbol{\theta}, \boldsymbol{\gamma}) \leq \mathcal{L}_2(\boldsymbol{\theta}, \boldsymbol{\gamma}) \leq \ldots \leq \mathcal{L}_K(\boldsymbol{\theta}, \boldsymbol{\gamma}) \xrightarrow[K \to \infty]{} \ell(\boldsymbol{\theta}).$$

$\mathcal{L}_K(\boldsymbol{\theta}, \boldsymbol{\gamma})$ 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 $\mathcal{L}_1(\boldsymbol{\theta}, \boldsymbol{\gamma})$, which is the loosest bound!

The VAE bound can be interestingly rewritten

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

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

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

What about VAEs?

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

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

What about VAEs?

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

The VAE bound can be interestingly rewritten

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

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

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

What about VAEs?

- The VAE bound can also be rewritten

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

What about VAEs?

- The VAE bound can also be rewritten

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

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

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

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?

This motivates the name variational auto-encoder, as we can see the loss as a KL-regularised auto-encoder loss!

- The VAE bound can also be rewritten

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

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

Why is this a reconstruction error?

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

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

Why is this a reconstruction error?

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

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

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

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

Why is this a reconstruction error?

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

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

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

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

Why is this a reconstruction error?

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

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = -\sum_{j=1}^{d} \mathrm{XEnt}(\mathbf{x}, \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{z}))$$

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 $\mathrm{KL}\left(\prod_{i=1}^{n} q_{\boldsymbol{\gamma}}(\mathbf{z}_i|\mathbf{x}_i) \middle|\middle| \prod_{i=1}^{n} p_{\boldsymbol{\theta}}(\mathbf{z}_i)\right)$, which is the

divergence between the approximate posterior (aka encoding) and the prior.

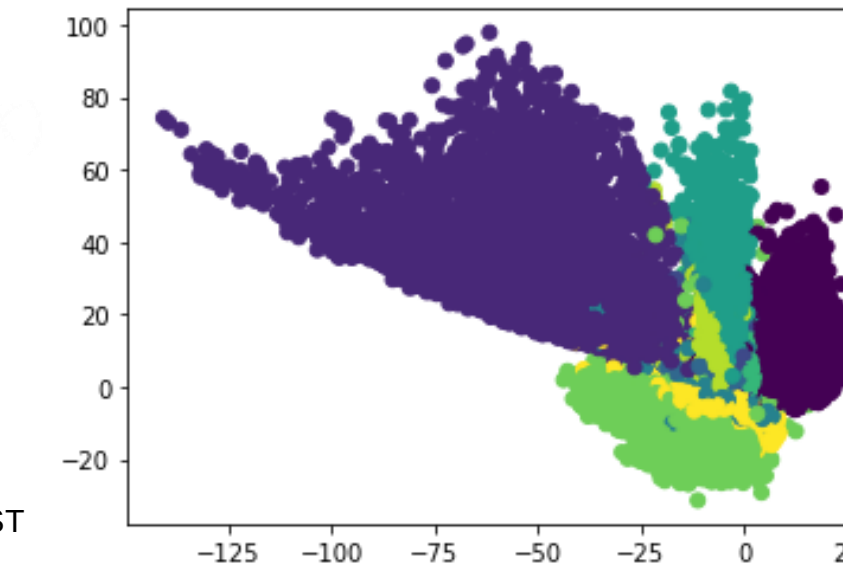
- Minimising the VAE bound will therefore lead to solutions such that **encodings are somewhat collectively similar to the prior**.

Why is the KL term a regulariser?

- The KL part of the VAE bound is equal to $\mathrm{KL}\left(\prod_{i=1}^{n} q_{\boldsymbol{\gamma}}(\mathbf{z}_i|\mathbf{x}_i) \middle|\middle| \prod_{i=1}^{n} p_{\boldsymbol{\theta}}(\mathbf{z}_i)\right)$, 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…



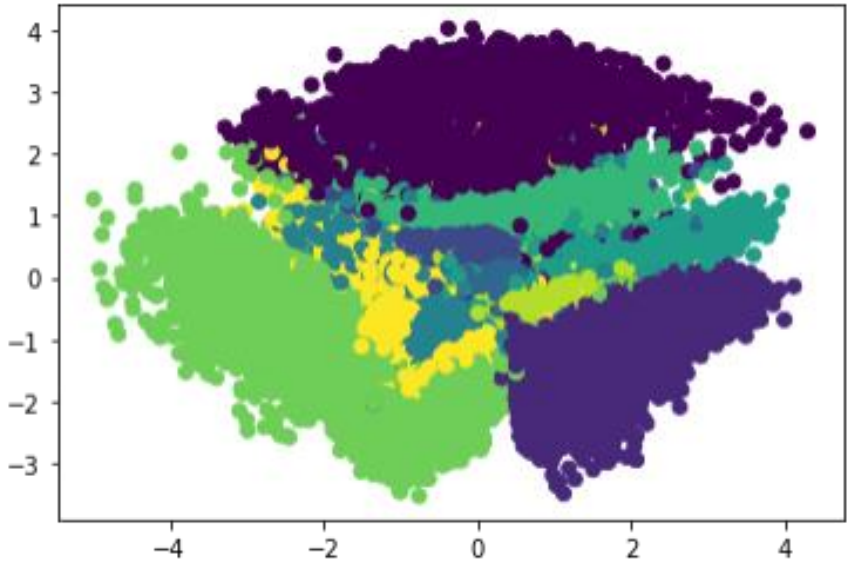
2D code space of a standard AE on MNIST

Why is the KL term a regulariser?

- The KL part of the VAE bound is equal to $\mathrm{KL}\left(\prod_{i=1}^{n} q_{\boldsymbol{\gamma}}(\mathbf{z}_i|\mathbf{x}_i) \middle|\middle| \prod_{i=1}^{n} p_{\boldsymbol{\theta}}(\mathbf{z}_i)\right)$, 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!**



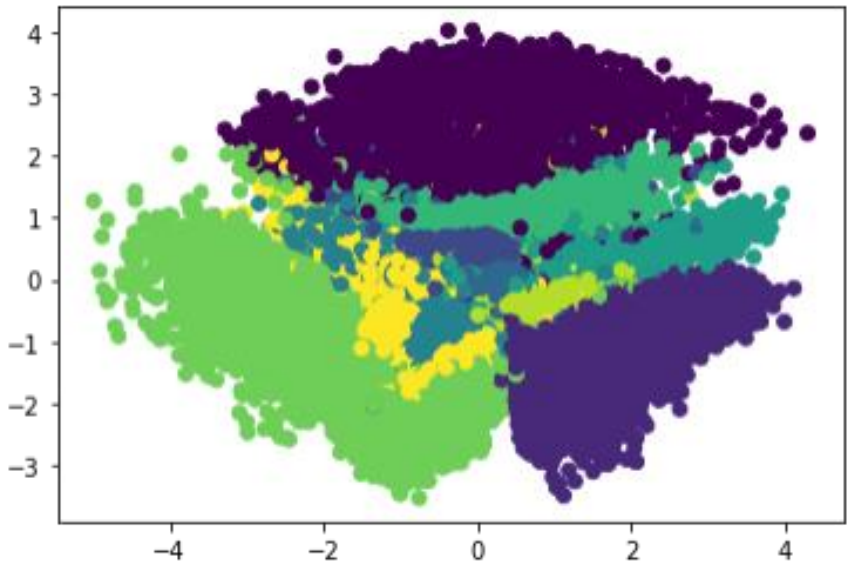
2D code space of a VAE on MNIST

Why is the KL term a regulariser?

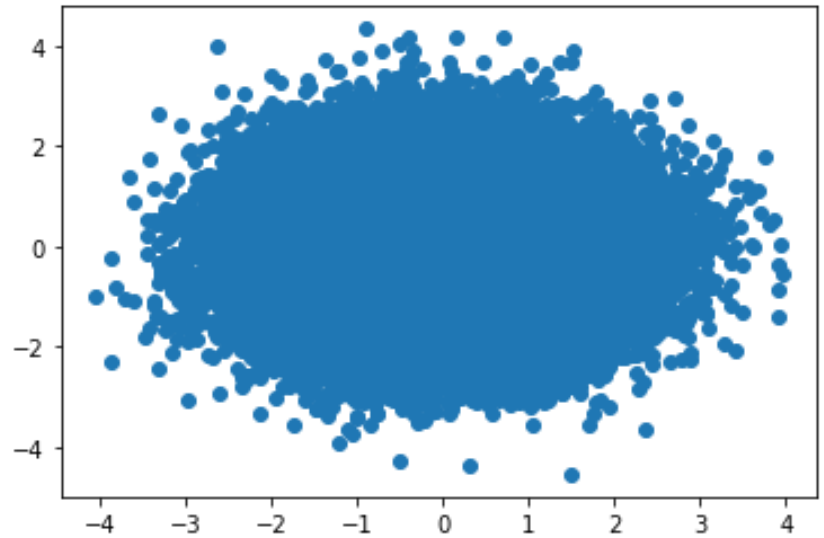
- The KL part of the VAE bound is equal to $\mathrm{KL}\left(\prod_{i=1}^{n} q_{\boldsymbol{\gamma}}(\mathbf{z}_i|\mathbf{x}_i) \middle|\middle| \prod_{i=1}^{n} p_{\boldsymbol{\theta}}(\mathbf{z}_i)\right)$, which is the

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

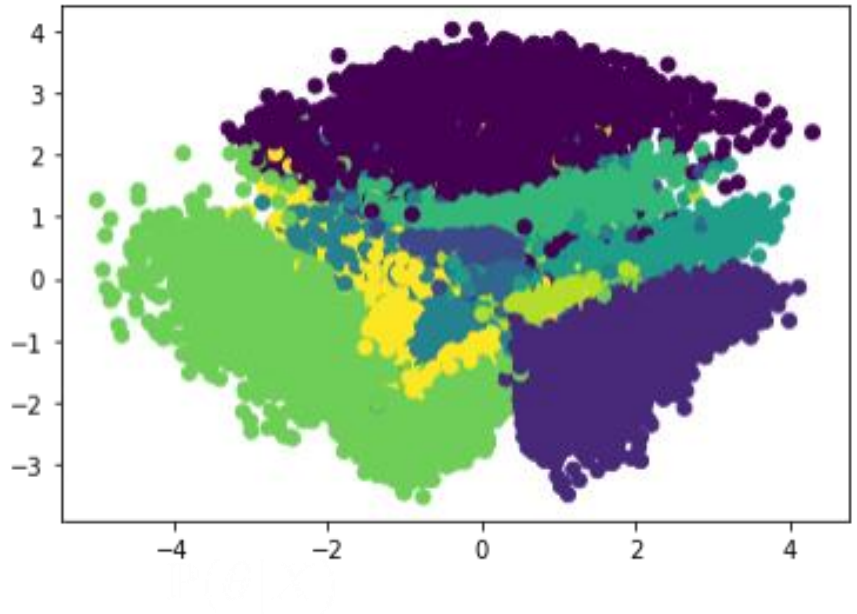


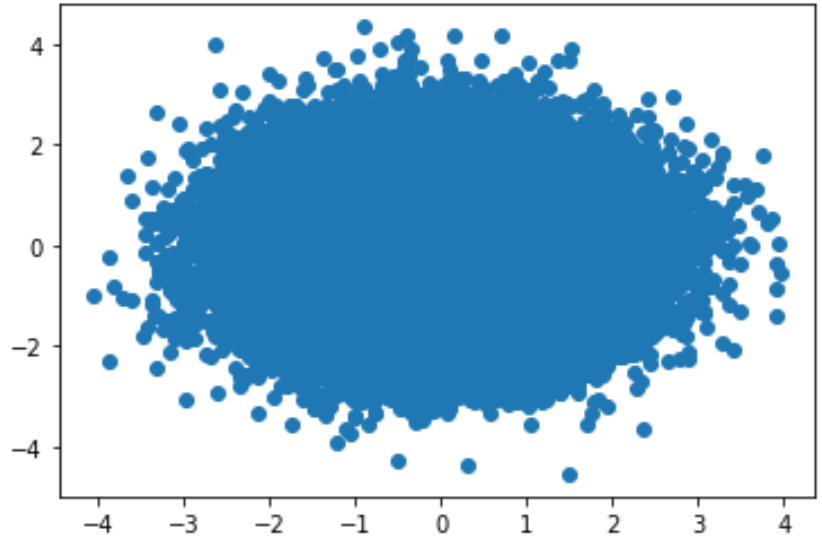
Standard Gaussian samples

More on this KL regulariser

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

2D code space of a VAE on MNIST





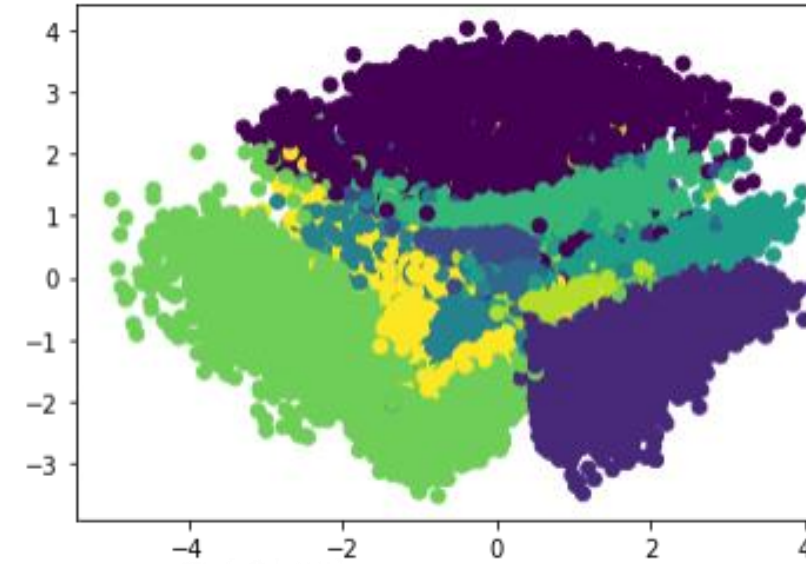
Standard Gaussian samples

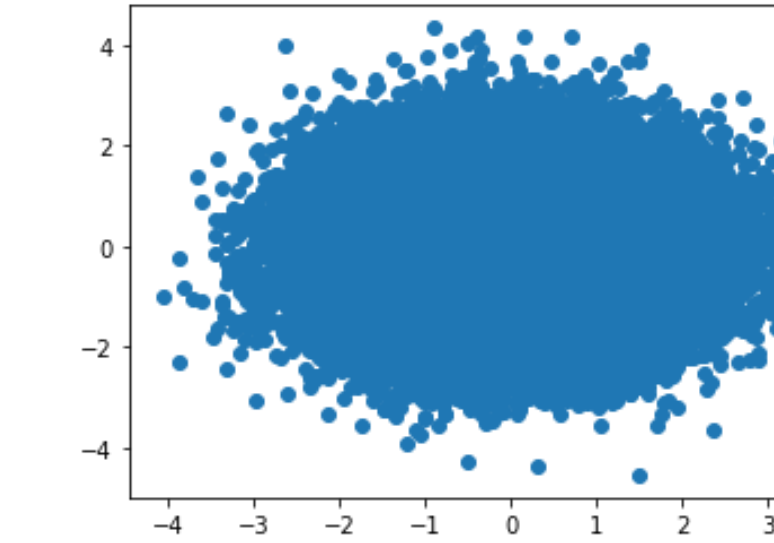
- **Is it a good or a bad thing?**

More on this KL regulariser

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

2D code space of a VAE on MNIST





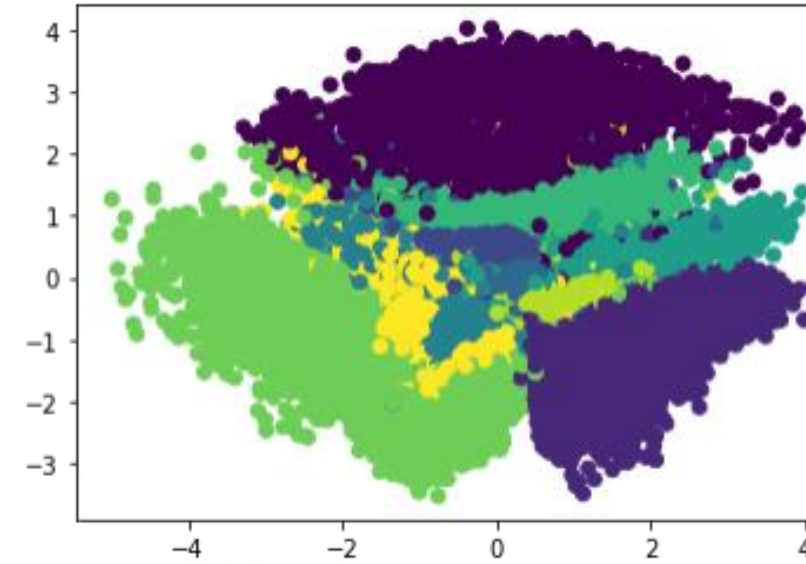
Standard Gaussian samples

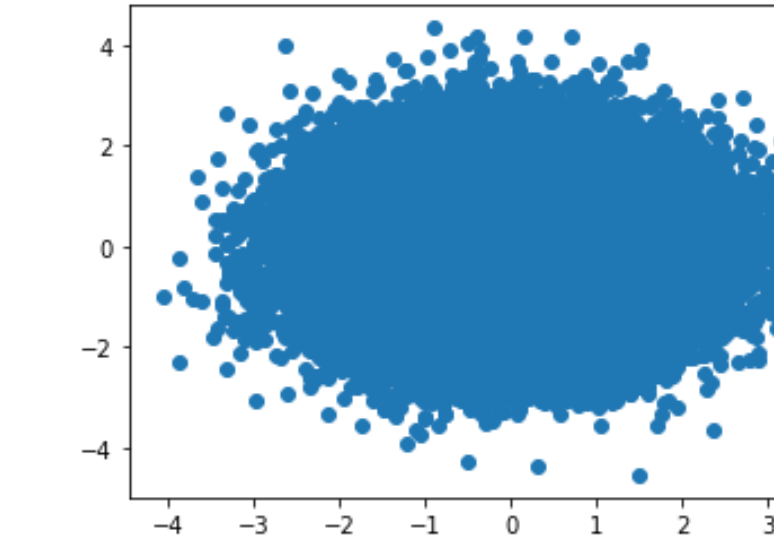
- **Is it a good or a bad thing?**
  - **Good: Gaussian data are well-behaved** (in particular, Euclidean geometry makes sense), the latent space has no nonlinear structure

More on this KL regulariser

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

2D code space of a VAE on MNIST





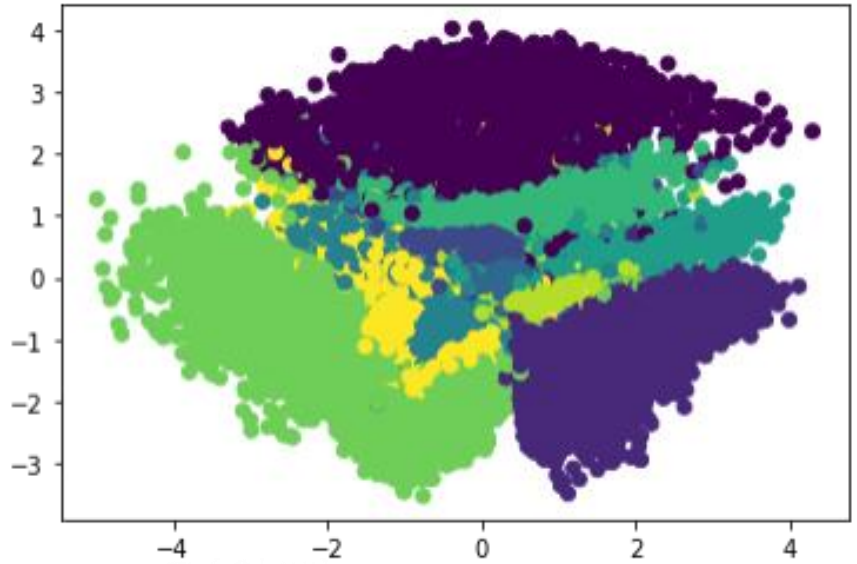
Standard Gaussian samples

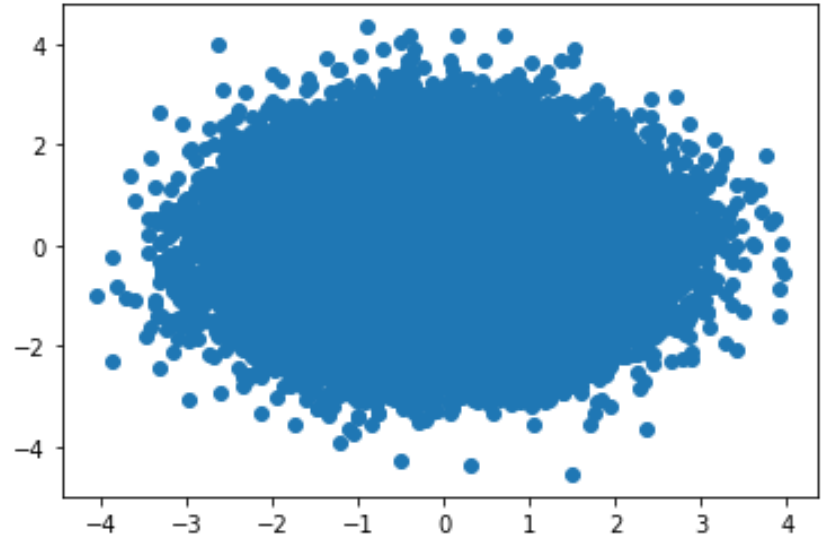
- **Is it a good or a bad thing?**
  - **Good: Gaussian data are well-behaved** (in particular, Euclidean geometry makes sense), the latent space has no nonlinear structure
  - **Bad:** The latent space is not discriminative! **Any idea how to simply solve this?**

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

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

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

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

- It's easy to backprop through this, ans this was used in the seminal VAE papers!

- 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

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

- **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

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

- **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

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

- **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 $\nabla_{\boldsymbol{\theta},\boldsymbol{\gamma}} \mathcal{L}_K(\boldsymbol{\theta}, \boldsymbol{\gamma})$

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

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

- 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[*1] SHAKIR@GOOGLE.COM
Mihaela Rosca[*1 2] MIHAELACR@GOOGLE.COM
Michael Figurnov[*1] MFIGURNOV@GOOGLE.COM
Andriy Mnih[*1] AMNIH@GOOGLE.COM

Let's start with the easy part: ∇_θ

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

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

Let's start with the easy part: ∇_θ

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

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

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

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

Let's start with the easy part: ∇_θ

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

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

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

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

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

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

Let's start with the easy part: ∇_θ

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

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

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

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

which means that we can get an unbiased estimate of $\nabla_\theta f(\boldsymbol{\theta}, \gamma)$ by simply sampling $\mathbf{w}_1, \ldots, \mathbf{w}_K \sim \pi_{\boldsymbol{\gamma}}(\mathbf{w})$ 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: ∇_γ

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

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

Now the tricky part: ∇_γ

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

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

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

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

Now the tricky part: ∇_γ

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

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

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

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

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

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

Now the tricky part: ∇_γ

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

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

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

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

Now the tricky part: ∇_γ

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

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

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

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

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

Now the tricky part: ∇_γ

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

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

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

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

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

$$\int \nabla_\gamma \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \right] \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w} \approx \frac{1}{K} \sum_{k=1}^{K} \nabla_\gamma \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}_k) \right]$$

Now the tricky part: ∇_γ

- The real tricky part is what's left: the second term $\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w}$

that is not an expected value!

Now the tricky part: ∇_γ

- The real tricky part is what's left: the second term $\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w}$

that is not an expected value!

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

$$\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \frac{\nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]}{\pi_{\boldsymbol{\gamma}}(\mathbf{w})} \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w}$$

Now the tricky part: ∇_γ

- The real tricky part is what's left: the second term $\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w}$

that is not an expected value!

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

$$\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \frac{\nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]}{\pi_{\boldsymbol{\gamma}}(\mathbf{w})} \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w}$$

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

Now the tricky part: ∇_γ

- The real tricky part is what's left: the second term $\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w}$

that is not an expected value!

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

$$\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \frac{\nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]}{\pi_{\boldsymbol{\gamma}}(\mathbf{w})} \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w}$$

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

$$\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w} = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \log \pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]$$

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

Now the tricky part: ∇_γ

- The estimate $\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w} = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \log \pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]$

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

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

Now the tricky part: ∇_γ

- The estimate $\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w} = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \log \pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]$

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

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.

Now the tricky part: ∇_γ

- The estimate $\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w} = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \log \pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]$

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

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.

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

Now the tricky part: ∇_γ

- The estimate $\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w} = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \log \pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]$

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

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.

- 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 ∇_γ

- Our goal is still to estimate our tricky term $\nabla_\gamma \int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w}$

- Again, the main issue is that the density depends on the parameter of interest. Can we destroy this dependence? Can we **push γ away from the density** we're integrating against?

Reparametrisation trick for ∇_γ

- Our goal is still to estimate our tricky term $\nabla_\gamma \int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w}$

- Again, the main issue is that the density depends on the parameter of interest. Can we destroy this dependence? Can we **push γ away from the density** we're integrating against?

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

Reparametrisation trick for ∇_γ

- Our goal is still to estimate our tricky term $\nabla_\gamma \int g(\boldsymbol{\theta}, \gamma, \mathbf{w}) \pi_\gamma(\mathbf{w}) d\mathbf{w}$

- Again, the main issue is that the density depends on the parameter of interest. Can we destroy this dependence? Can we **push γ away from the density** we're integrating against?

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

- In this setting, it is clear that sampling $\mathbf{w} \sim \pi_\gamma$ can be done by computing

$$\mathbf{w} = \mu_\gamma + \underbrace{C_\gamma \varepsilon}_{\sim \mathcal{N}(0, \Sigma)}$$

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

Reparametrisation trick for ∇_γ

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

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

Reparametrisation trick for ∇_γ

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

where $C_{\boldsymbol{\gamma}}$ is the Cholesky decomposition of the covariance, 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 $\boldsymbol{\varepsilon}$

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

Reparametrisation trick for ∇_γ

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

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

- But now, the only random thing is ε. This means we can rewrite our expectation as an expectation over ε

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

…and finally use Leibniz's rule

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

$$\approx \frac{1}{K} \sum_{k=1}^{K} \nabla_\gamma \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mu_{\boldsymbol{\gamma}} + C_{\boldsymbol{\gamma}} \boldsymbol{\varepsilon}_k) \right]$$

Reparametrisation trick for ∇_γ

- The estimate

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

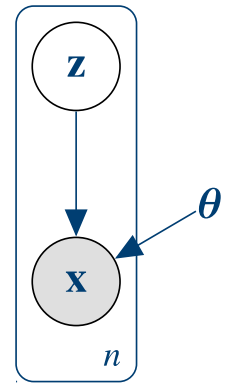
$$\approx \frac{1}{K} \sum_{k=1}^{K} \nabla_\gamma \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mu_{\boldsymbol{\gamma}} + C_{\boldsymbol{\gamma}} \boldsymbol{\varepsilon}_k) \right]$$

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

- 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).