More on latent variable models

Pierre-Alexandre Mattei

Menu for this lecture

1. Recap on VAEs + implementing variational bounds
2. How to optimise variational bounds? + implementation
3. Variational bounds beyond VAEs: Bayesian models, topic models, diffusion models…
4. On d-separation and VAEs

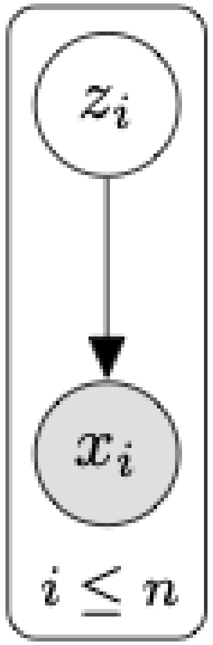
1

Recap on VAEs/IWAEs

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.

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

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

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

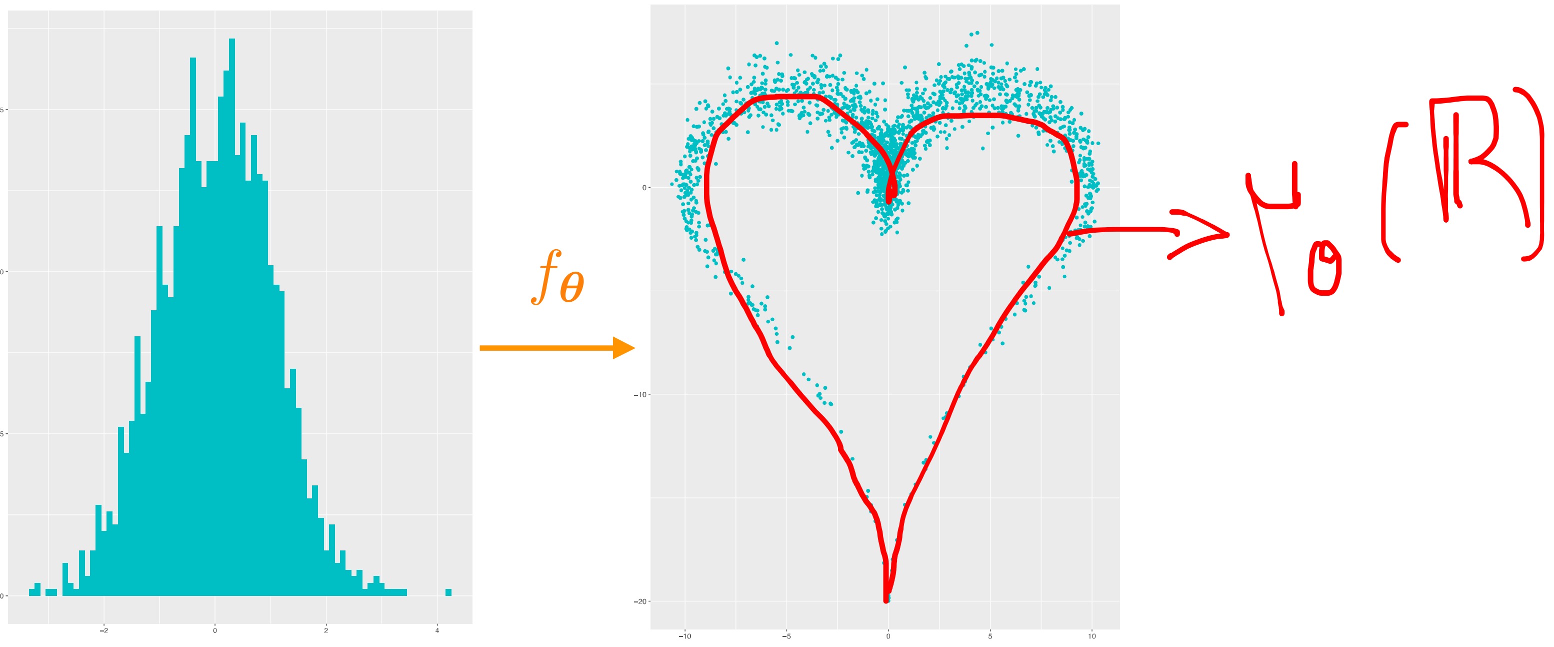
Deep latent variable models: the role of the decoder

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.

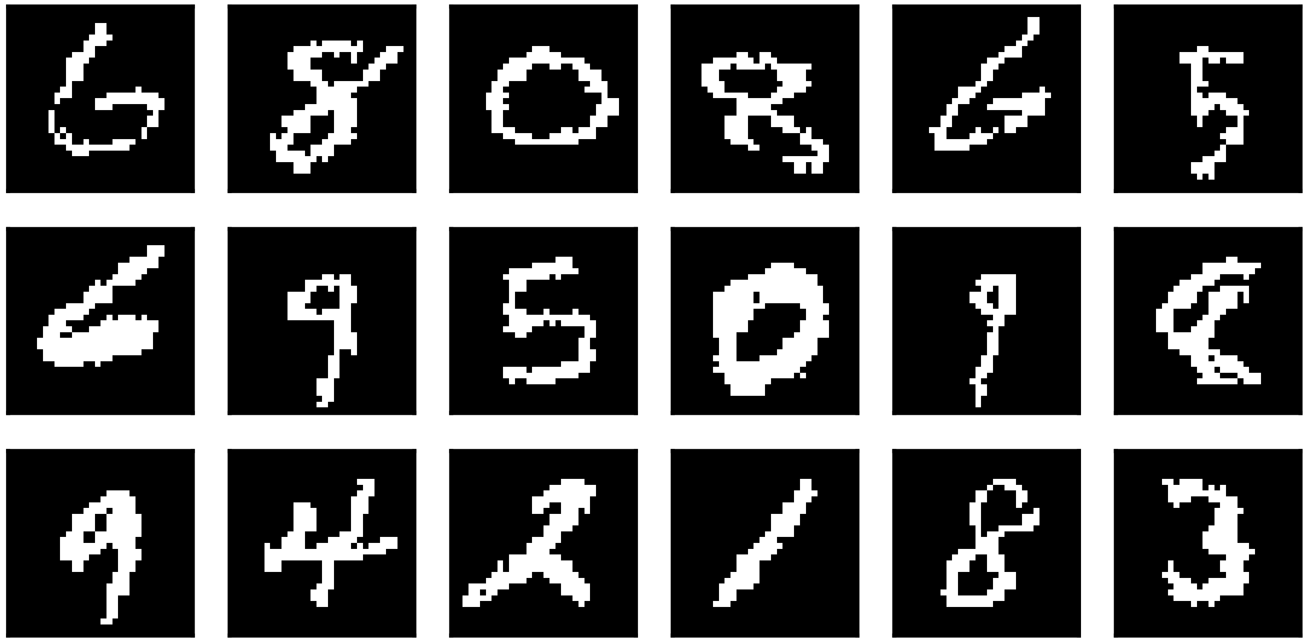
Simple non-linear decoder ($d = 1$**,** $p = 2$): $f_{\boldsymbol\theta}(z) = \boldsymbol\mu_{\boldsymbol\theta}(\mathbf{z}), \boldsymbol\Sigma_{\boldsymbol\theta}(\mathbf{z})$ with, for all $z \in \mathbb{R}$,

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



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

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

Decoder network

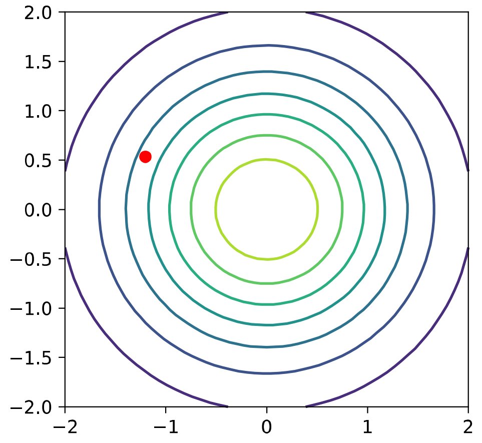
$$f(\mathbf{z}) = \mathrm{Sigmoid}(\mathbf{V} \tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \boldsymbol{\beta})$$

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

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

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

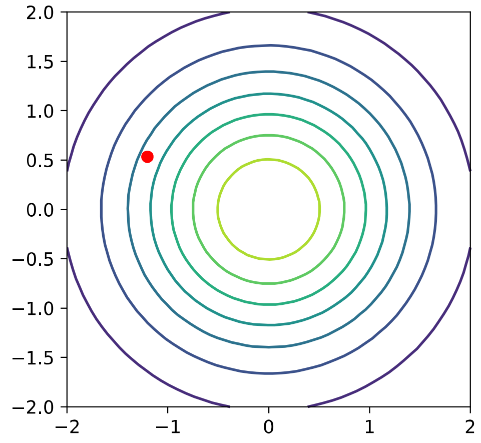
Decoder network

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

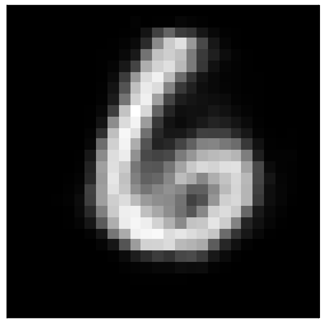
Generation

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

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



$f(\mathbf{z})$



Illustrative example of a DLVM for binary images

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

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

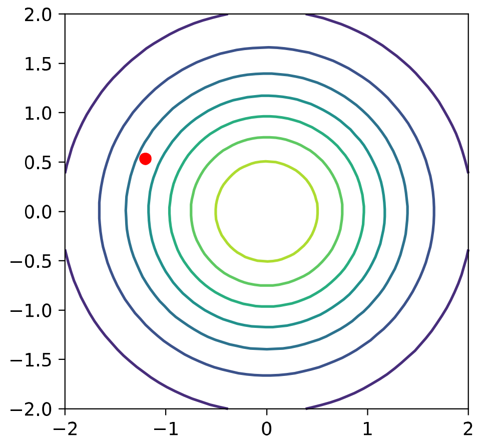
Decoder network

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

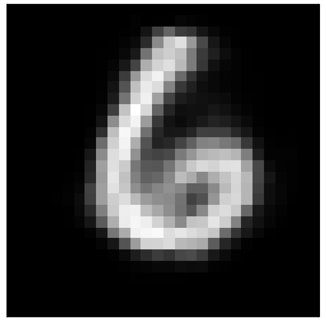
Generation

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

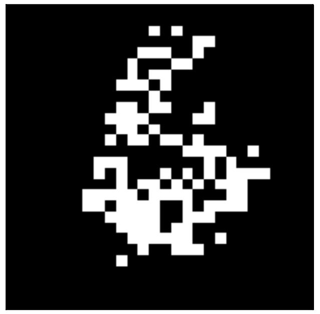
$$\mathbf{z} = (-1.2033, 0.5340)$$



$f(\mathbf{z})$



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



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

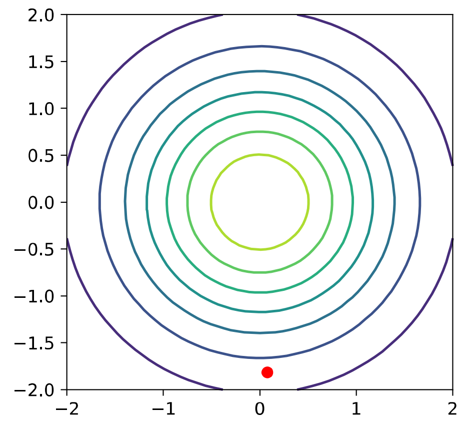
$$f(\mathbf{z}) = \text{Sigmoid}(\mathbf{V} \tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \boldsymbol{\beta})$$

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

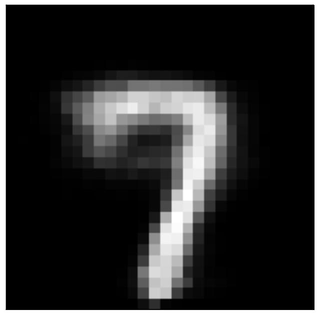
Generation

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

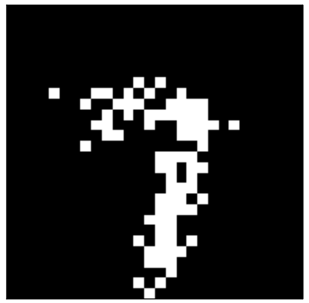
$$\mathbf{z} = (0.0791, -1.8165)$$



$f(z)$



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



Maximum likelihood for DLVM

The log-likelihood function of our dataset $(x_1, \dots, x_n)$ is

$$\ell(\theta) = \sum_{i=1}^{n} \log p(x_i) = \sum_{i=1}^{n} \log \int p(x_i|z)p(z)dz$$

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

Maximum likelihood for DLVM

The log-likelihood function of our dataset $(x_1, \dots, x_n)$ is

$$\ell(\theta) = \sum_{i=1}^{n} \log p(x_i) = \sum_{i=1}^{n} \log \int p(x_i|z)p(z)dz$$

$p(x_i)$ **is actually a difficult integral**

Maximum likelihood for DLVM

The log-likelihood function of our dataset $(x_1, \dots, x_n)$ is

$$\ell(\theta) = \sum_{i=1}^{n} \log p(x_i) = \sum_{i=1}^{n} \log \int p(x_i|z)p(z)dz$$

$p(x_i)$ is actually a difficult integral

Our solution from last lecture : **use importance sampling to attack the integral!**

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: $\hat{I}_K \overset{a.s.}{\to} I$
  - 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**

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

- 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!**

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

Beyond simple MC: importance sampling

- This new estimate $\hat{I}^q_K$ is called an **importance sampling** estimate.
- Now, is $\hat{I}^q_K$ 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 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

The likelihood is $\ell(\theta) = \sum_{i=1}^{n} \log p(x_i) = \sum_{i=1}^{n} \log \int p(x_i|z)p(z)dz$

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

The likelihood is $\ell(\theta) = \sum_{i=1}^{n} \log p(x_i) = \sum_{i=1}^{n} \log \int p(x_i|z)p(z)dz$

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

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

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** $q_1, \ldots, q_n$ (and n is usually large in deep learning...).

Maximum likelihood for DLVMs: choosing proposals

For the importance sampling problem

$$p(x_i) = \int p(x_i|z)p(z)dz \approx \frac{1}{K}\sum_{k=1}^{K}\frac{p(x_i|z_{ik})p(z_{ik})}{q_i(z_{ik})}$$

the optimal proposal will be

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

Maximum likelihood for DLVMs: choosing proposals

For the importance sampling problem

$$p(x_i) = \int p(x_i|z)p(z)dz \approx \frac{1}{K}\sum_{k=1}^{K} \frac{p(x_i|z_{ik})p(z_{ik})}{q_i(z_{ik})}$$

the optimal proposal will be

$$q_i^\star(z) = \frac{p(x_i|z)p(z)}{p(x_i)}$$

Maximum likelihood for DLVMs: choosing proposals

For the importance sampling problem

$$p(x_i) = \int p(x_i|z)p(z)dz \approx \frac{1}{K}\sum_{k=1}^{K} \frac{p(x_i|z_{ik})p(z_{ik})}{q_i(z_{ik})}$$

the optimal proposal will be

$$q_i^\star(z) = \frac{p(x_i|z)p(z)}{p(x_i)} = p(z|x_i)$$

aka the posterior (again, not in a Bayesian sense).

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:

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

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

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

2

Properties of the bound

Properties of the bound

Theorem. $\mathcal{L}_K(\theta, \gamma) \leq \ell(\theta)$.

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

Properties of the bound

Theorem. $\mathcal{L}_K(\theta, \gamma) \leq \ell(\theta)$.

Proof. Without loss of generality, we assume that there is a single point x in our dataset. Let $z_1, \ldots, z_K \sim q_i$, and let

$$w_k = \frac{p(x|z_k)p(z_k)}{q_i(z_k)}, \text{ and } \bar{w}_K = (w_1 + \ldots + w_K)/K.$$

Properties of the bound

Theorem. $\mathcal{L}_K(\theta, \gamma) \leq \ell(\theta)$.

Proof. Without loss of generality, we assume that there is a single point x in our dataset. Let $z_1, \ldots, z_K \sim q_i$, and let

$$w_k = \frac{p(x|z_k)p(z_k)}{q_i(z_k)}, \text{ and } \bar{w}_K = (w_1 + \ldots + w_K)/K.$$

With these notations, $\mathcal{L}_K = \mathbb{E}[\log \bar{w}_K]$. Because IS is unbiased, $\mathbb{E}[\bar{w}_K] = p(x)$.

Properties of the bound

Theorem. $\mathcal{L}_K(\theta, \gamma) \leq \ell(\theta)$.

Proof. Without loss of generality, we assume that there is a single point x in our dataset. Let $z_1, \ldots, z_K \sim q_i$, and let

$$w_k = \frac{p(x|z_k)p(z_k)}{q_i(z_k)}, \text{ and } \bar{w}_K = (w_1 + \ldots + w_K)/K.$$

With these notations, $\mathcal{L}_K = \mathbb{E}[\log \bar{w}_K]$. Because IS is unbiased, $\mathbb{E}[\bar{w}_K] = p(x)$. Now, since the log is concave, we can use Jensen's inequatity to get

$$\mathcal{L}_K \leq \mathbb{E}[\log \bar{w}_K] \leq \log \mathbb{E}[\bar{w}_K] = \log p(x)$$

$\square$

Properties of the bound

Theorem. $\mathcal{L}_1(\theta, \gamma) \leq \ldots \leq \mathcal{L}_K(\theta, \gamma) \leq \ell(\theta).$

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

Properties of the bound

Theorem. $\mathcal{L}_1(\theta, \gamma) \leq \ldots \leq \mathcal{L}_K(\theta, \gamma) \leq \ell(\theta)$.

Proof. Same notations as before. The key idea is to notice that

$$\frac{1}{K}\sum_{k=1}^{K} w_k = \frac{1}{K}\sum_{j=1}^{K}\frac{1}{K-1}\sum_{k\neq j} w_k$$

Properties of the bound

Theorem. $\mathcal{L}_1(\theta,\gamma) \leq \ldots \leq \mathcal{L}_K(\theta,\gamma) \leq \ell(\theta)$.

Proof. Same notations as before. The key idea is to notice that

$$\frac{1}{K}\sum_{k=1}^{K} w_k = \frac{1}{K}\sum_{j=1}^{K}\frac{1}{K-1}\sum_{k\neq j} w_k$$

Then, using Jensen's leads to

$$\log\left(\frac{1}{K}\sum_{k=1}^{K} w_k\right) \geq \frac{1}{K}\sum_{j=1}^{K}\log\left(\frac{1}{K-1}\sum_{k\neq j} w_k\right)$$

and to the desired result. $\square$

What about VAEs?

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

The VAE bound can be interestingly rewritten

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

which means that, for a given $\boldsymbol{\theta}$, **the optimal $q_\gamma(\mathbf{z}_i|\mathbf{x}_i)$ 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 $q_\gamma(\mathbf{z}_i|\mathbf{x}_i)$ 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.

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

What about VAEs?

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

The VAE bound can be interestingly rewritten

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

which means that, for a given $\boldsymbol{\theta}$, **the optimal $q_\gamma(\mathbf{z}_i|\mathbf{x}_i)$ 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 $q_\gamma(\mathbf{z}_i|\mathbf{x}_i)$ 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

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

What about VAEs?

- The VAE bound can also be rewritten

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

Reconstruction error:
mismatch between **x**$_i$ and its reconstruction (obtained by auto-encoding it)

What about VAEs?

- The VAE bound can also be rewritten

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

Reconstruction error:
mismatch between **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

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

Reconstruction error:
mismatch between **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 **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 $\mathbf{z} \sim q_{\gamma}(\mathbf{z}|\mathbf{x})$

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

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

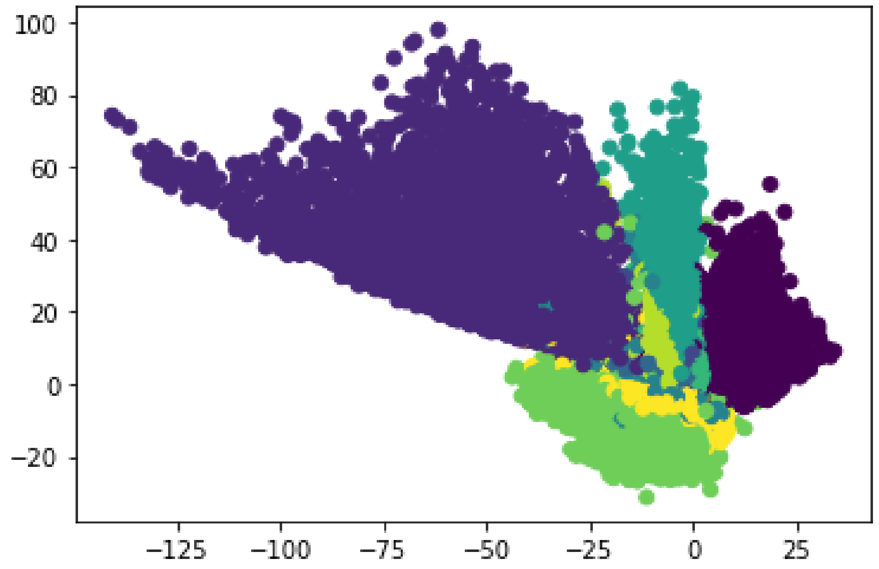
- Minimising the VAE bound will therefore lead to solutions such that **encodings are somewhat collectively similar to the prior**.

Why is the KL term a regulariser?

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

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

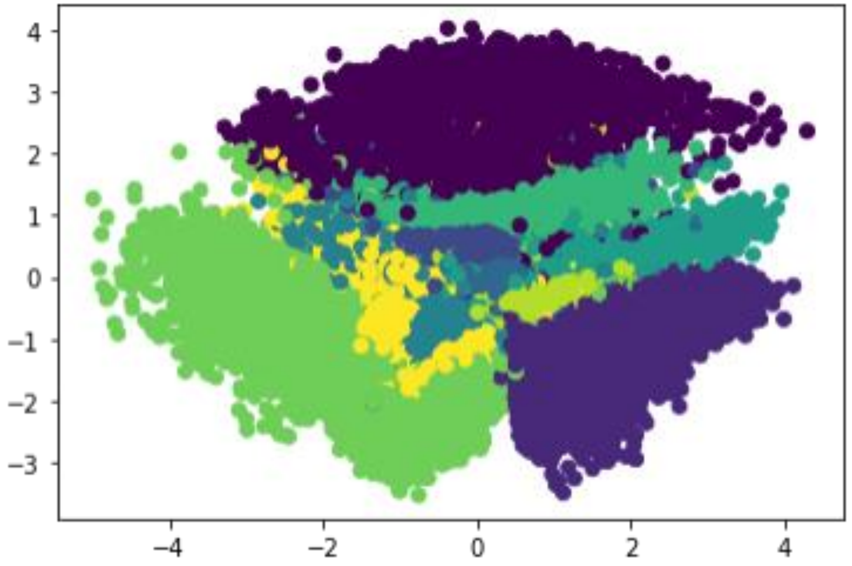
- Minimising the VAE bound will therefore lead to solutions such that **encodings are somewhat collectively similar to the prior**.
- Typically, remember that the prior is often simply standard Gaussian! In practice, the encodings of traditional VAEs have no reason to be standard Gaussian…



2D code space of a standard AE on MNIST

Why is the KL term a regulariser?

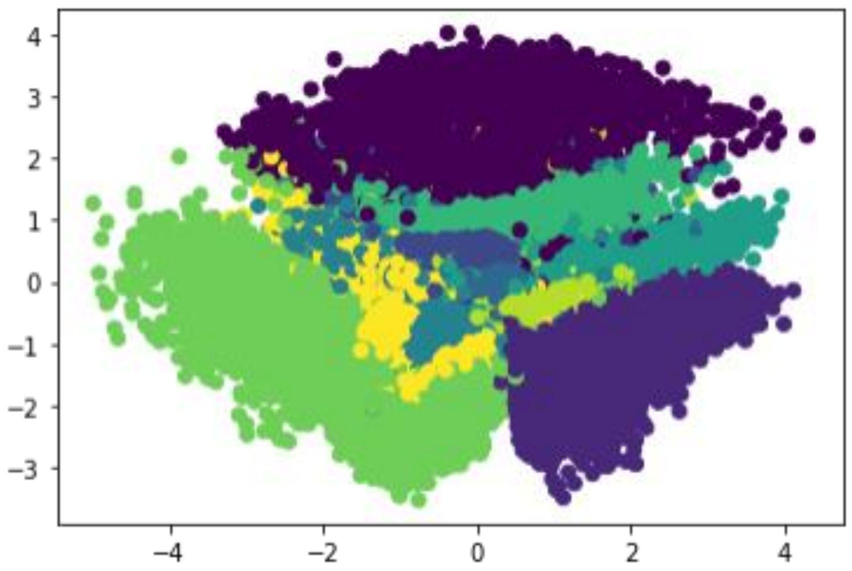
- The KL part of the VAE bound is equal to $\mathrm{KL}\left(\prod_{i=1}^{n} q_{\boldsymbol{\gamma}}(\mathbf{z}_i|\mathbf{x}_i) \middle|\middle| \prod_{i=1}^{n} p_{\boldsymbol{\theta}}(\mathbf{z}_i)\right)$, which is the **divergence between the approximate posterior (aka encoding) and the prior.**
-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



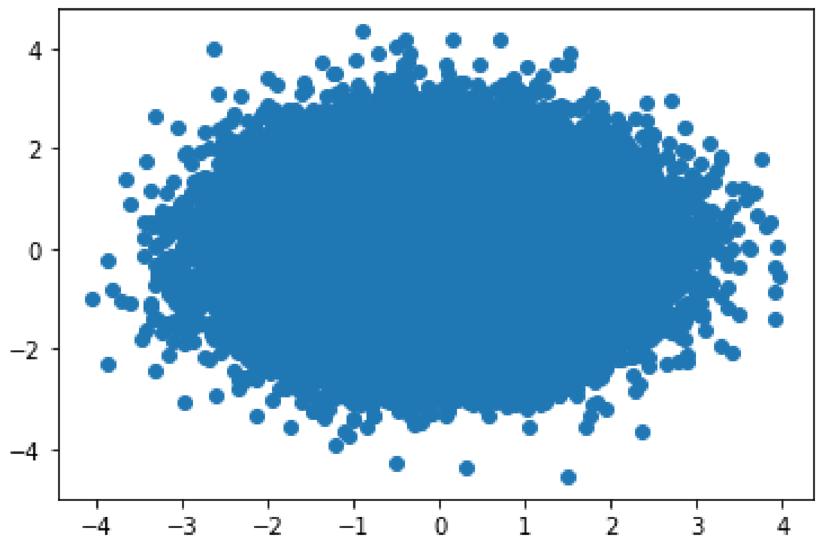
2D code space of a VAE on MNIST

Why is the KL term a regulariser?

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



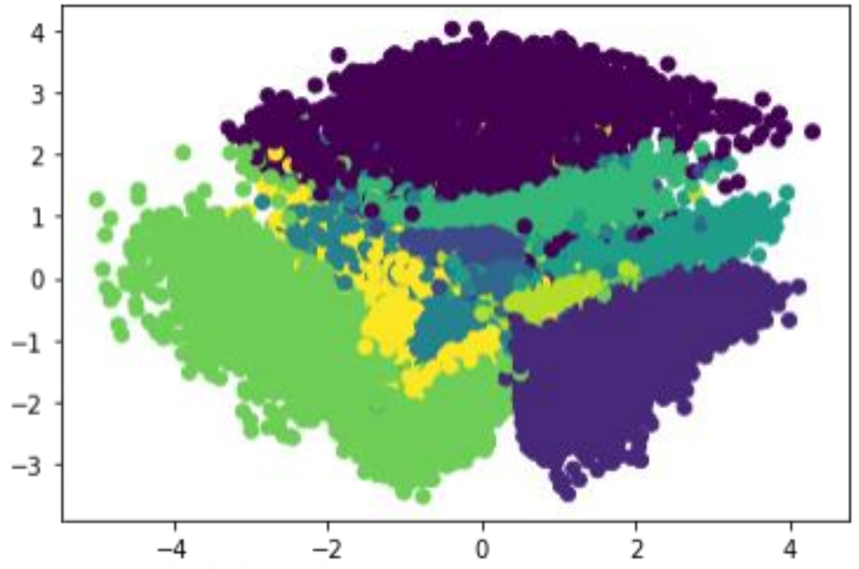
2D code space of a VAE on MNIST



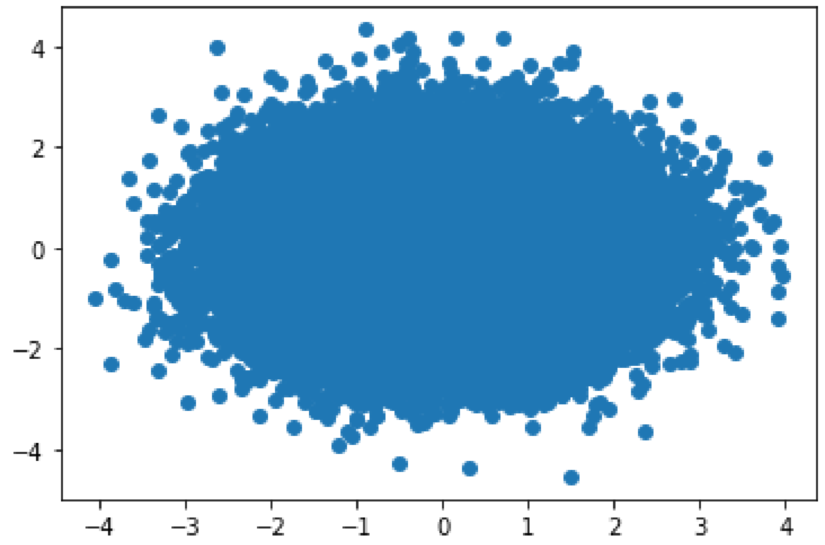
Standard Gaussian samples

More on this KL regulariser

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



2D code space of a VAE on MNIST

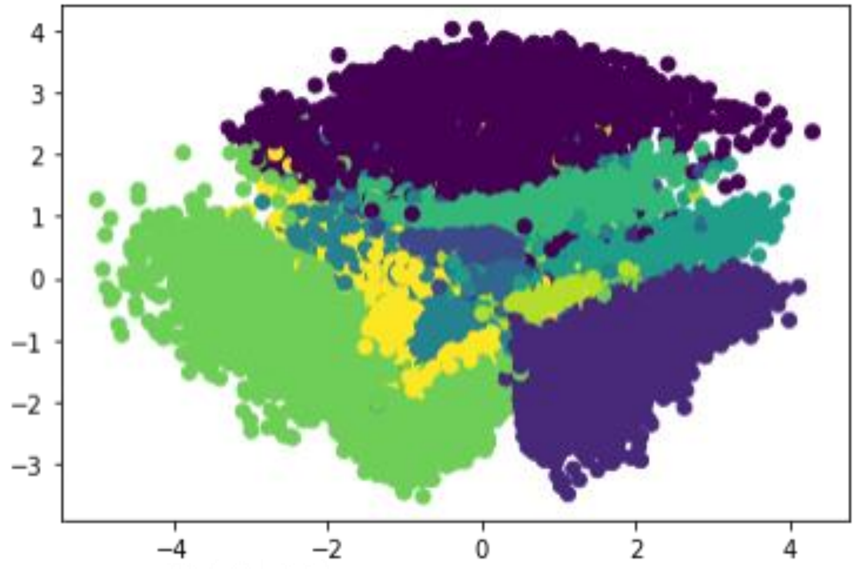


Standard Gaussian samples

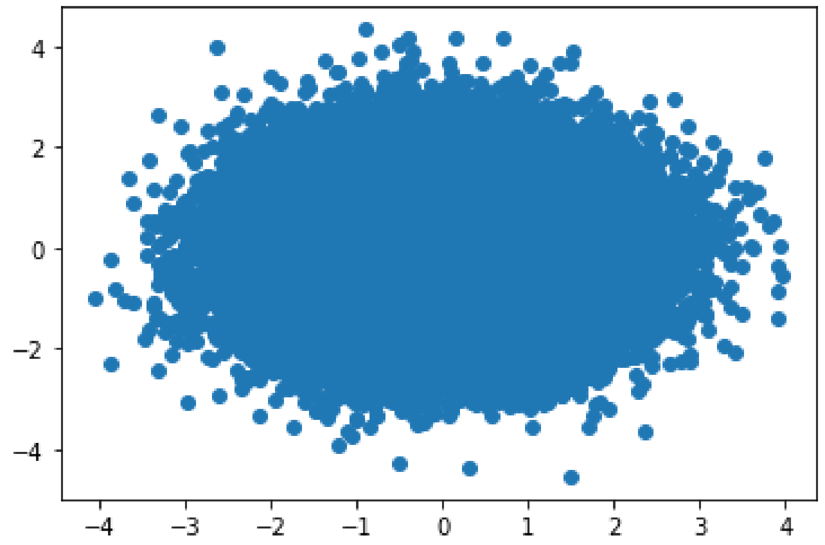
- **Is it a good or a bad thing?**

More on this KL regulariser

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



2D code space of a VAE on MNIST

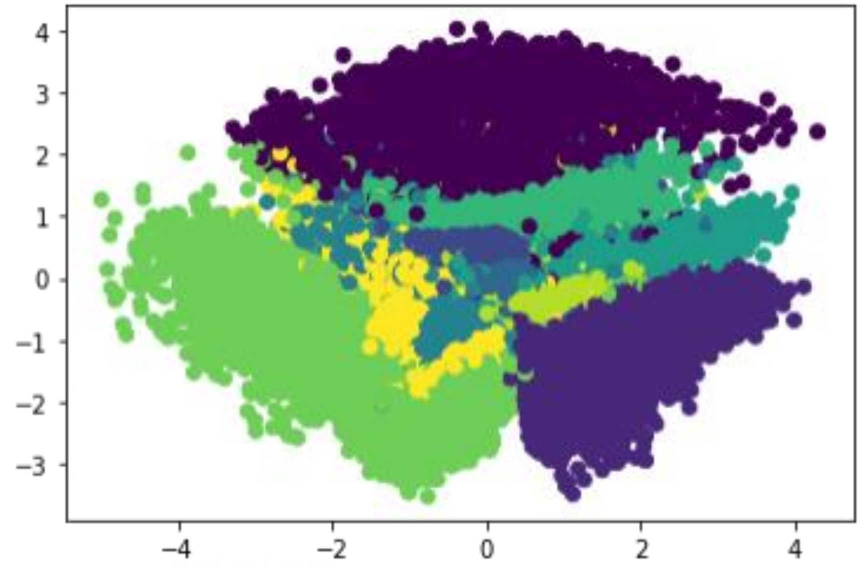


Standard Gaussian samples

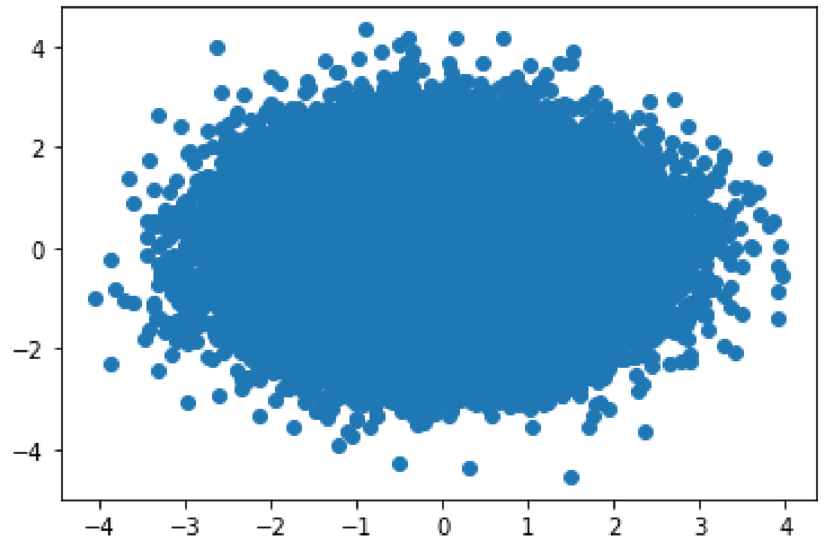
- **Is it a good or a bad thing?**
  - **Good: Gaussian data are well-behaved** (in particular, Euclidean geometry makes sense), the latent space has no nonlinear structure

More on this KL regulariser

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



2D code space of a VAE on MNIST

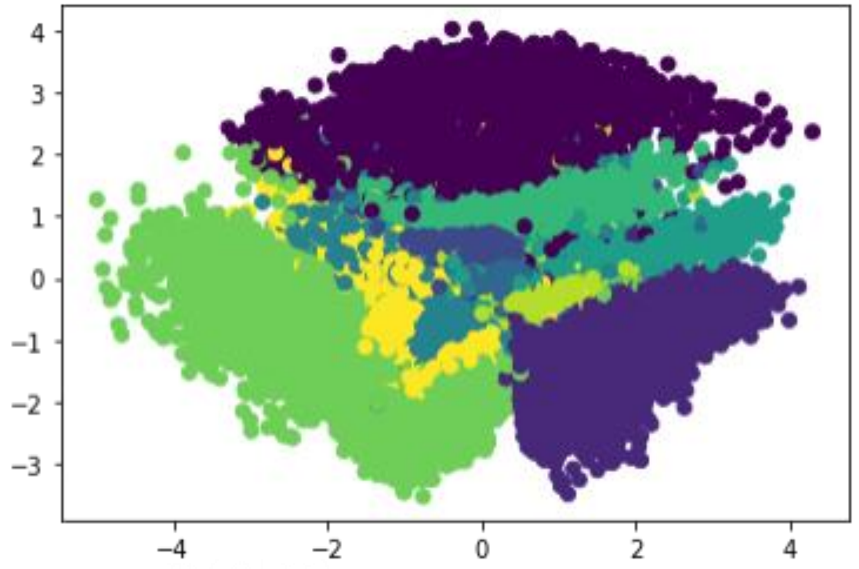


Standard Gaussian samples

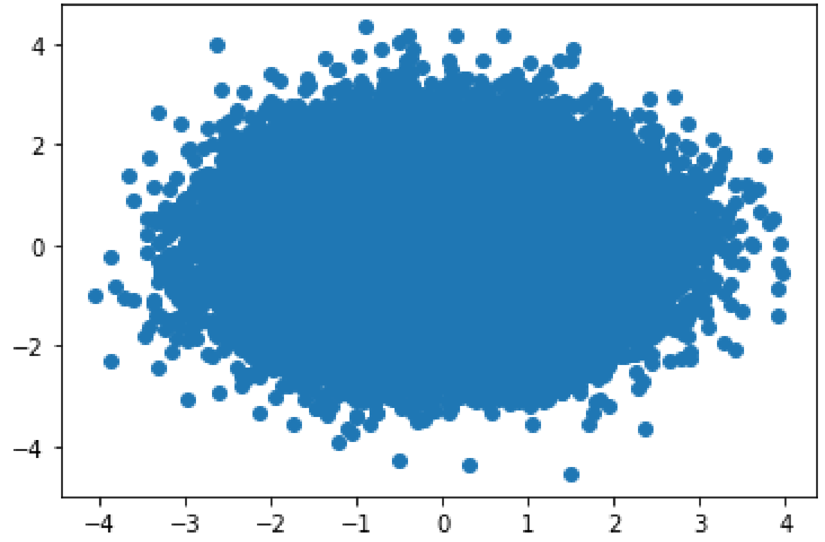
- **Is it a good or a bad thing?**
  - **Good: Gaussian data are well-behaved** (in particular, Euclidean geometry makes sense), the latent space has no nonlinear structure
  - **Bad:** The latent space is not discriminative! **Any idea how to simply solve this?**

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

$$\textbf{KL}(\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) || \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)) =$$

$$\frac{1}{2}\left(\textbf{tr}(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\Sigma}_0) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^T \boldsymbol{\Sigma}_1^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) + \log\frac{\det \boldsymbol{\Sigma}_1}{\det \boldsymbol{\Sigma}_0} - p\right)$$

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

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

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

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

- Not easy to generalise beyond Gaussians…

3

How to train VAEs?

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

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

- **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

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

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

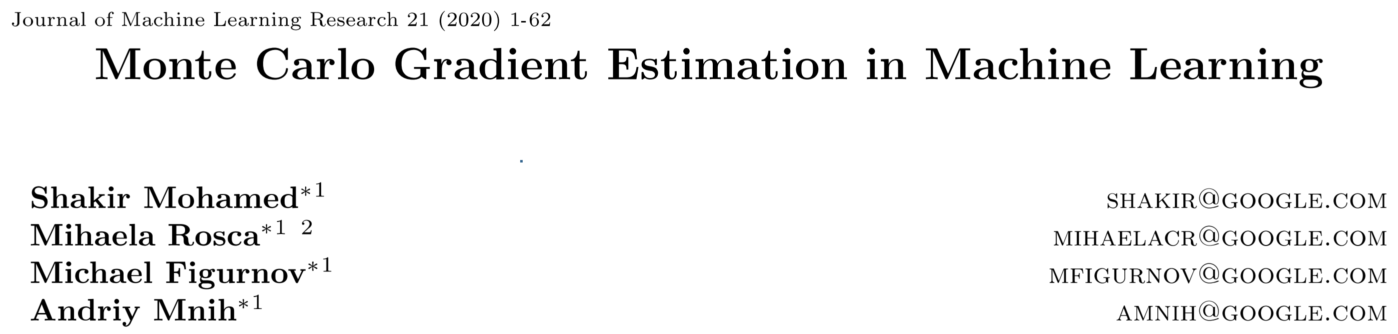
- The idea is to use stochastic gradient descent (SGD, or one of its variants). What do we need to compute to perform SGD on an objective? **Unbiased estimates of the gradients!**
- As we’ll see, it is doable to compute unbiased estimates of $\nabla_{\theta,\gamma} \mathcal{L}_K(\boldsymbol{\theta}, \gamma)$

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

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

- 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

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

- 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

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

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

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

Let’s start with the easy part: ∇_θ

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

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

- 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

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

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

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

Now the tricky part: ∇_{γ}

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

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

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

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

Now the tricky part: ∇_{γ}

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

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

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

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

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

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

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!

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

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

where C_γ 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

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

Reparametrisation trick for ∇_γ

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

where C_γ 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 $\boldsymbol{\varepsilon}$

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

…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

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

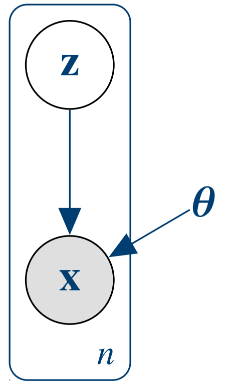
is often called the **reparametrisation trick** estimate. It has considerably less variance in practice than the score gradient, but can be less generally applied.

- 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).

4

On d-separation and latent variable models

D-separation

- D-separation means **directed separation.**

- It’s a general framework for answering questions à la «$X_A \perp\!\!\!\perp X_B | X_C?$» where A, B, C are three subsets of the set of nodes.

D-separation

- D-separation means **directed separation.**

- It's a general framework for answering questions à la «$X_A \perp\!\!\!\perp X_B | X_C ?$» where A, B, C are three subsets of the set of nodes.

- **D-separation recipe:** we consider all chains between any node in A and any node in B. Any of these chains is said to be **blocked by C** if it includes a node such that either
  - the chain is a v-structure at the node, and neither the node, nor its descendants, are in C
  - the arrows on the chain meet either head-to-tail or tail-to-tail at the node, and the node belongs to C

D-separation

- D-separation means **directed separation.**

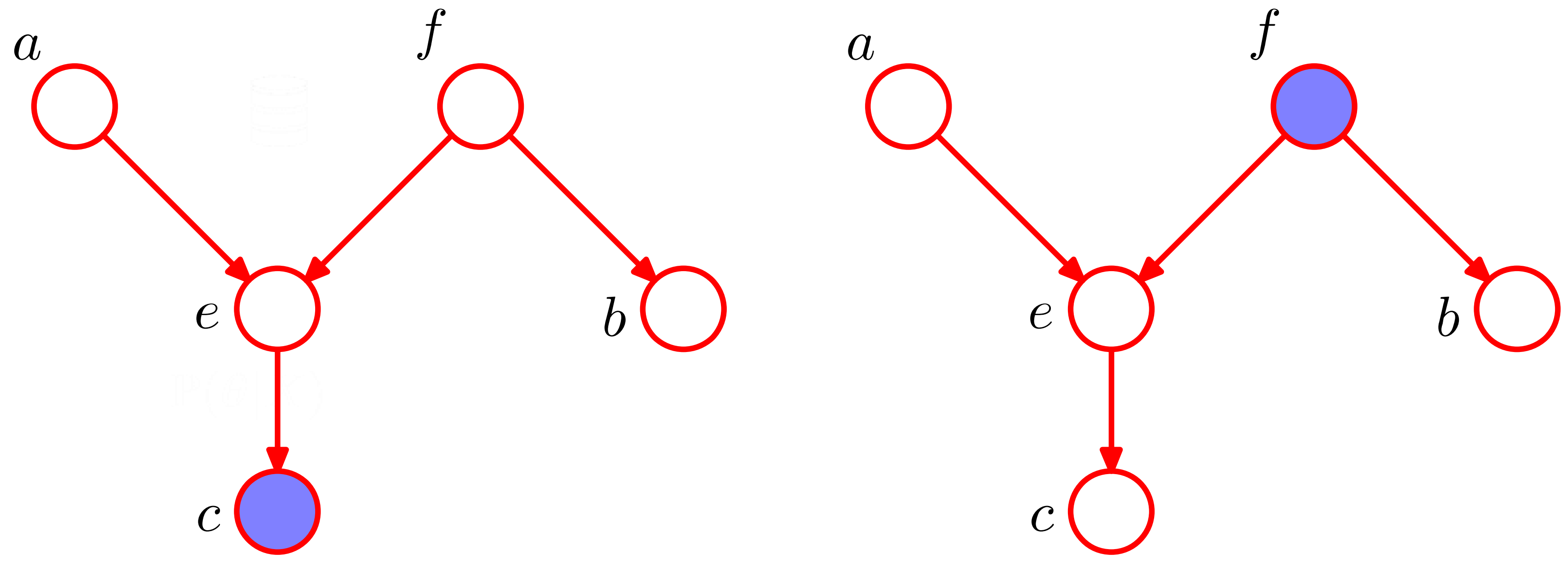
- It's a general framework for answering questions à la « $X_A \perp\!\!\!\perp X_B | X_C ?$ » where A, B, C are three subsets of the set of nodes.

- **D-separation recipe:** we consider all chains between any node in A and any node in B. Any of these chains is said to be **blocked by C** if it includes a node such that either
    - the chain is a v-structure at the node, and neither the node, nor its descendants, are in C
    - the arrows on the chain meet either head-to-tail or tail-to-tail at the node, and the node belongs to C

Definition. *We say that A* ***and*** *B* ***are d-separated by*** *C if all chains between A and B are blocked by C.*

Examples of d-separation

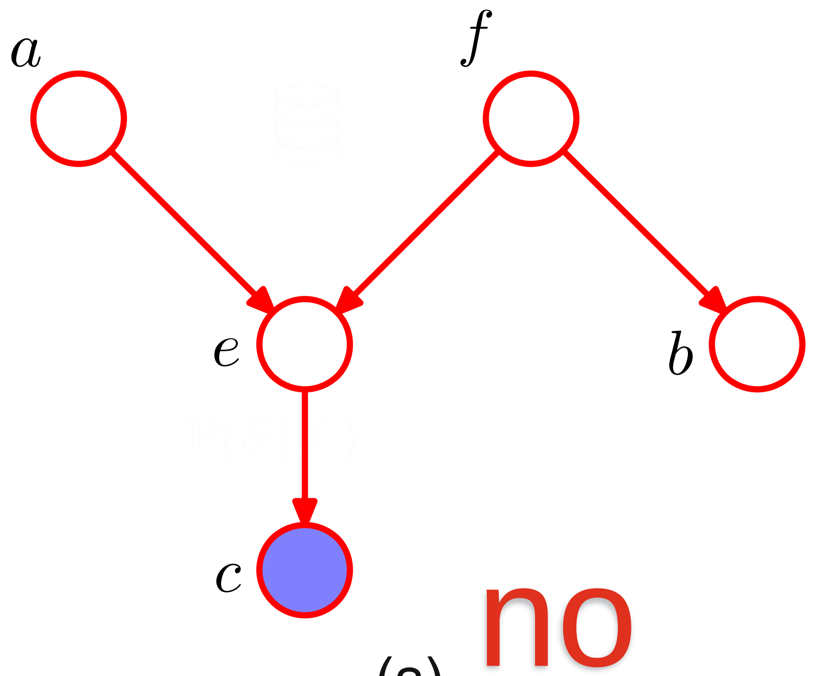
- I took this figure from [Bishop, Fig. 8.22]. Does the blue node d-separate a and b?



(a) (b)

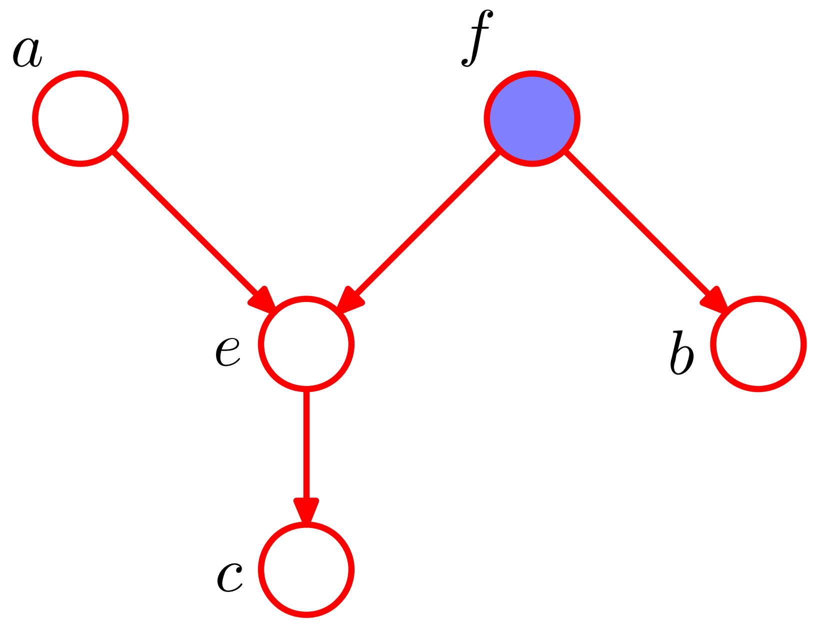
Examples of d-separation

- I took this figure from [Bishop, Fig. 8.22]. Does the blue node d-separate a and b?



(a)

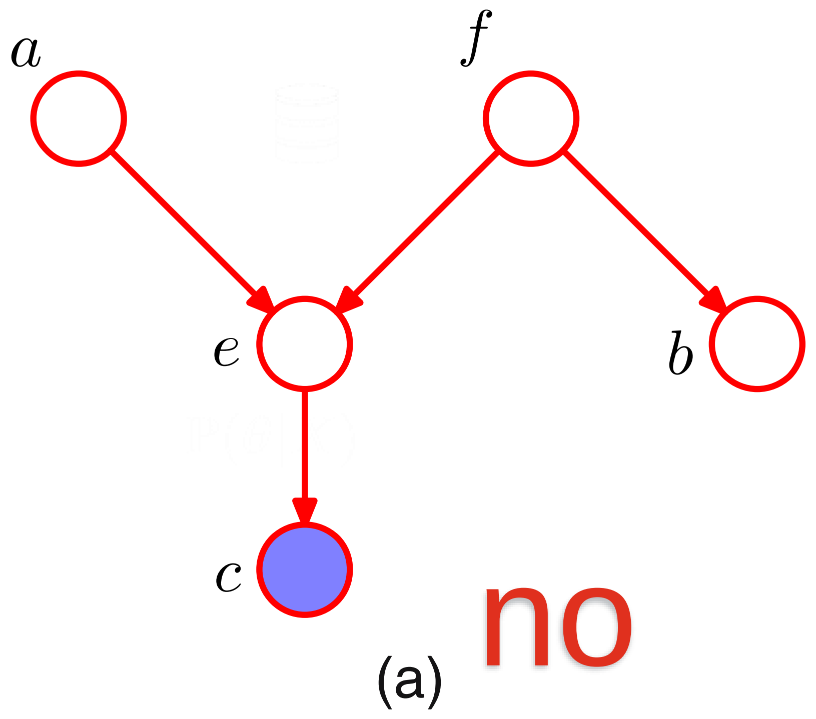
no

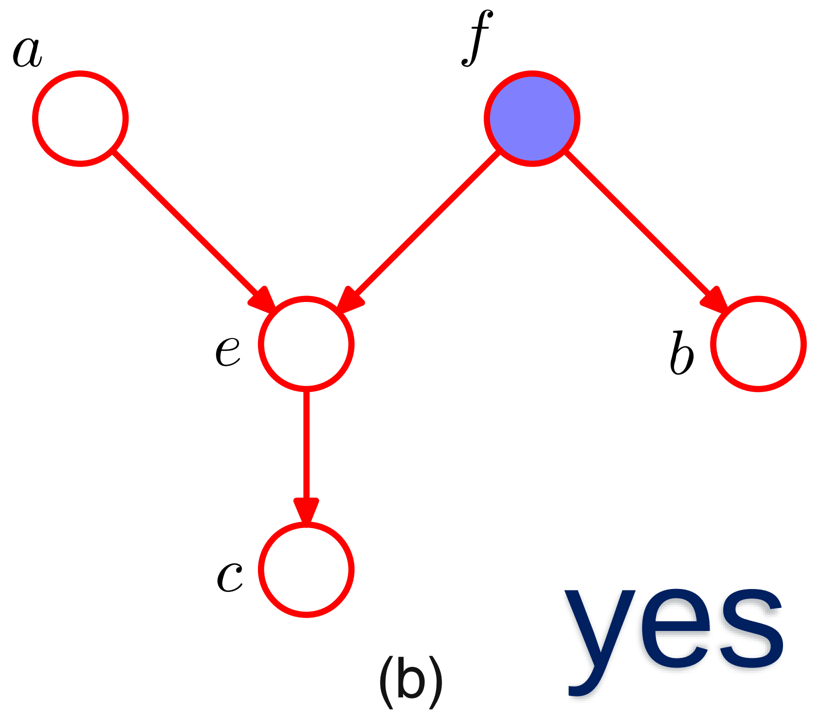


(b)

Examples of d-separation

- I took this figure from [Bishop, Fig. 8.22]. Does the blue node d-separate a and b?





Properties of d-separation

Theorem (soundness of d-separation). *If A and B are d-separated by C and $p \in \mathcal{L}(G)$, then $X_A \perp\!\!\!\perp X_B | X_C$.*

Theorem (completeness of d-separation). *If A and B are not d-separated by C, then there exist $p \in \mathcal{L}(G)$ such that $X_A \not\perp\!\!\!\perp X_B | X_C$.*

For more details, including proofs, see [PGM, Sec. 3.3.2].

Applications of d-separation

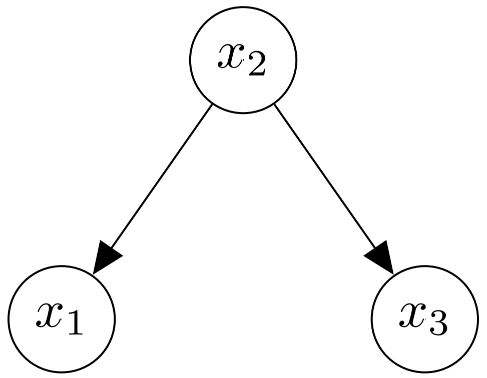
- We previously mentioned conditional independence properties. Try to prove them using d-separation:

$$\begin{tikzcd} x_1 \arrow[r] & x_2 \arrow[r] & x_3 \end{tikzcd}$$

$$x_3 \perp\!\!\!\perp x_1 | x_2$$

Applications of d-separation

- We previously mentioned conditional independence properties. Try to prove them using d-separation:



$$x_3 \perp\!\!\!\perp x_1 | x_2$$