

# Variational Autoencoders: From Theory to Practice

Clément Chadebec

Université de Paris - INRIA (HeKA team) - INSERM

NeurIPS 2022

February 28, 2023

# Overview

## 1 Variational Autoencoder - The Idea

- Autoencoder
- VAE framework
- Mathematical foundations
- Some use cases

## 2 Enhancing the model

- Tweaking the variational distribution
- Building better estimators
- Questioning our priors

## 3 VAE in Practice with Pythae

# Autoencoder

- The objective  $\Rightarrow$  Dimensionnality Reduction

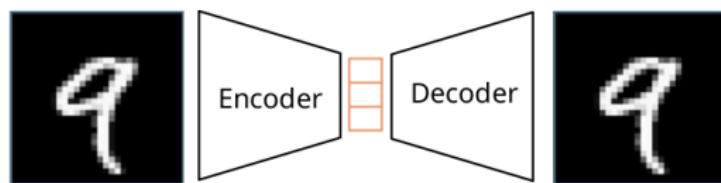


Figure: Simple Autoencoder

- Need for a representation of the image  $\Rightarrow$  vectors

# Autoencoder

- The objective  $\Rightarrow$  Dimensionality Reduction

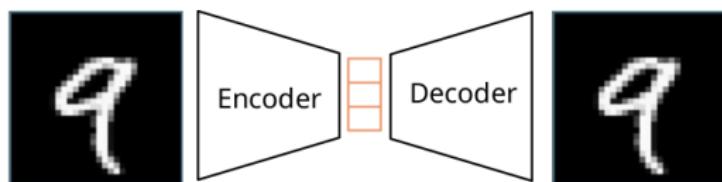


Figure: Simple Autoencoder

- Need for a representation of the image  $\Rightarrow$  vectors

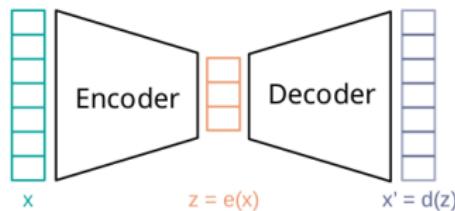


Figure: Simple Autoencoder

# AutoEncoder

Assumptions:

- Let  $x \in \mathcal{X}$  be a set a data. We assume that there exists  $z \in \mathcal{Z}$  such that  $z$  is a low dimensional representation of  $x$
- The encoder  $e_\theta$  and decoder  $d_\phi$  are functions modelled by neural networks (NNs) such that  $\theta$  and  $\phi$  are the weights of the NNs
- Let  $x'$  be the reconstructed samples, the objective is to have  $x \simeq x'$

The Objective function writes:

$$\mathcal{L} = \|x - x'\|^2 = \|x - d_\phi(z)\|^2 = \|x - d_\phi(e_\theta(x))\|^2$$

⇒ The networks are optimised using stochastic gradient descent

$$\begin{aligned}\phi &\leftarrow \phi - \varepsilon \cdot \nabla_\phi \mathcal{L} \\ \theta &\leftarrow \theta - \varepsilon \cdot \nabla_\theta \mathcal{L}\end{aligned}$$

# AutoEncoder

Assumptions:

- Let  $x \in \mathcal{X}$  be a set a data. We assume that there exists  $z \in \mathcal{Z}$  such that  $z$  is a low dimensional representation of  $x$
- The encoder  $e_\theta$  and decoder  $d_\phi$  are functions modelled by neural networks (NNs) such that  $\theta$  and  $\phi$  are the weights of the NNs
- Let  $x'$  be the reconstructed samples, the objective is to have  $x \simeq x'$

The Objective function writes:

$$\mathcal{L} = \|x - x'\|^2 = \|x - d_\phi(z)\|^2 = \|x - d_\phi(e_\theta(x))\|^2$$

⇒ The networks are optimised using stochastic gradient descent

$$\begin{aligned}\phi &\leftarrow \phi - \varepsilon \cdot \nabla_\phi \mathcal{L} \\ \theta &\leftarrow \theta - \varepsilon \cdot \nabla_\theta \mathcal{L}\end{aligned}$$

# AutoEncoder

Assumptions:

- Let  $x \in \mathcal{X}$  be a set a data. We assume that there exists  $z \in \mathcal{Z}$  such that  $z$  is a low dimensional representation of  $x$
- The encoder  $e_\theta$  and decoder  $d_\phi$  are functions modelled by neural networks (NNs) such that  $\theta$  and  $\phi$  are the weights of the NNs
- Let  $x'$  be the reconstructed samples, the objective is to have  $x \simeq x'$

The Objective function writes:

$$\mathcal{L} = \|x - x'\|^2 = \|x - d_\phi(z)\|^2 = \|x - d_\phi(e_\theta(x))\|^2$$

⇒ The networks are optimised using stochastic gradient descent

$$\begin{aligned}\phi &\leftarrow \phi - \varepsilon \cdot \nabla_\phi \mathcal{L} \\ \theta &\leftarrow \theta - \varepsilon \cdot \nabla_\theta \mathcal{L}\end{aligned}$$

# AutoEncoder

Assumptions:

- Let  $x \in \mathcal{X}$  be a set a data. We assume that there exists  $z \in \mathcal{Z}$  such that  $z$  is a low dimensional representation of  $x$
- The encoder  $e_\theta$  and decoder  $d_\phi$  are functions modelled by neural networks (NNs) such that  $\theta$  and  $\phi$  are the weights of the NNs
- Let  $x'$  be the reconstructed samples, the objective is to have  $x \simeq x'$

The Objective function writes:

$$\mathcal{L} = \|x - x'\|^2 = \|x - d_\phi(z)\|^2 = \|x - d_\phi(e_\theta(x))\|^2$$

⇒ The networks are optimised using stochastic gradient descent

$$\begin{aligned}\phi &\leftarrow \phi - \varepsilon \cdot \nabla_\phi \mathcal{L} \\ \theta &\leftarrow \theta - \varepsilon \cdot \nabla_\theta \mathcal{L}\end{aligned}$$

# AutoEncoder

Assumptions:

- Let  $x \in \mathcal{X}$  be a set a data. We assume that there exists  $z \in \mathcal{Z}$  such that  $z$  is a low dimensional representation of  $x$
- The encoder  $e_\theta$  and decoder  $d_\phi$  are functions modelled by neural networks (NNs) such that  $\theta$  and  $\phi$  are the weights of the NNs
- Let  $x'$  be the reconstructed samples, the objective is to have  $x \simeq x'$

The Objective function writes:

$$\mathcal{L} = \|x - x'\|^2 = \|x - d_\phi(z)\|^2 = \|x - d_\phi(e_\theta(x))\|^2$$

⇒ The networks are optimised using stochastic gradient descent

$$\begin{aligned}\phi &\leftarrow \phi - \varepsilon \cdot \nabla_\phi \mathcal{L} \\ \theta &\leftarrow \theta - \varepsilon \cdot \nabla_\theta \mathcal{L}\end{aligned}$$

# AutoEncoder - Shortcomings

- How to generate new data ?

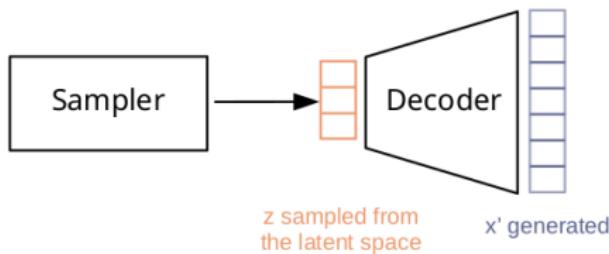


Figure: Generation procedure ?

- How to sample form the latent space?
- The AutoEncoder was just trained to **encode and decode the input data** without information on its structure or distribution.

⇒ Need for a new framework

# AutoEncoder - Shortcomings

- How to generate new data ?

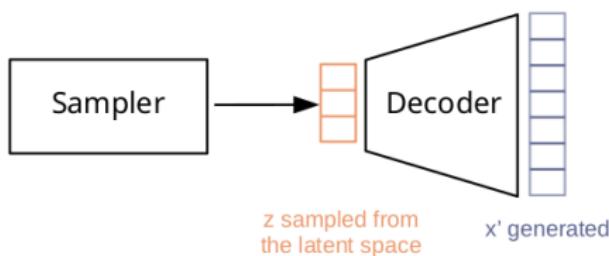


Figure: Generation procedure ?

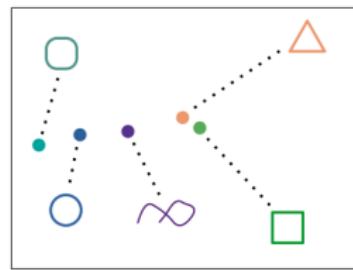


Figure: Potential latent space

- How to sample form the latent space?
- The AutoEncoder was just trained to encode and decode the input data without information on its structure or distribution.

⇒ Need for a new framework

# AutoEncoder - Shortcomings

- How to generate new data ?

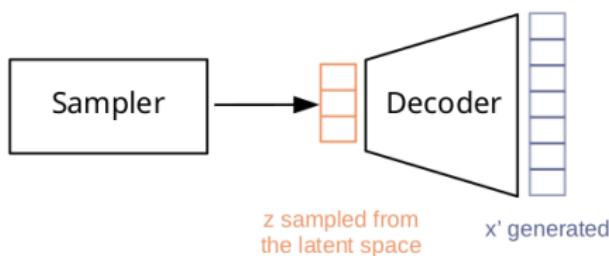


Figure: Generation procedure ?

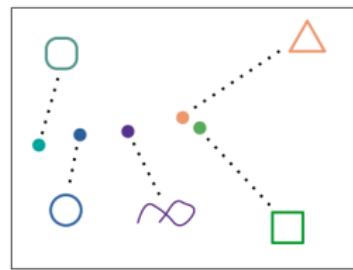


Figure: Potential latent space

- How to sample form the latent space?
- The AutoEncoder was just trained to **encode and decode the input data** without information on its structure or distribution.

⇒ Need for a new framework

# AutoEncoder - Shortcomings

- How to generate new data ?

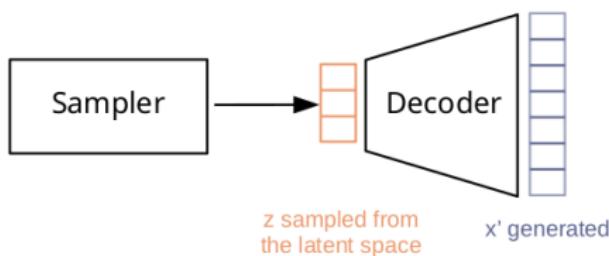


Figure: Generation procedure ?

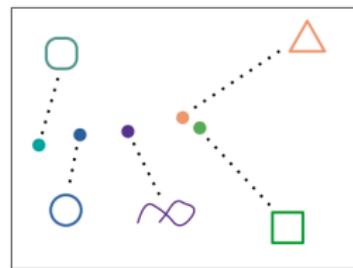


Figure: Potential latent space

- How to sample form the latent space?
- The AutoEncoder was just trained to **encode and decode the input data** without information on its structure or distribution.

⇒ Need for a new framework

# VAE - The Idea

- An autoencoder based model...

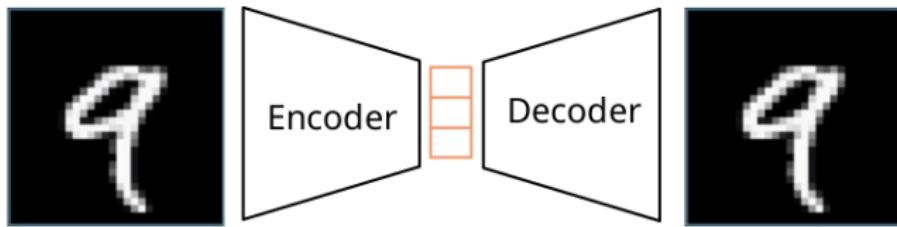


Figure: Simple autoencoder

- ... but where an input data point is encoded as a **distribution** defined over the latent space [16, 26]

# VAE - The Idea

- An autoencoder based model...

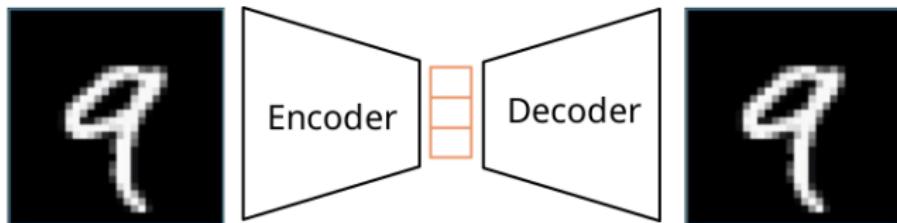


Figure: Simple autoencoder

- ... but where an input data point is encoded as a **distribution** defined over the latent space [16, 26]

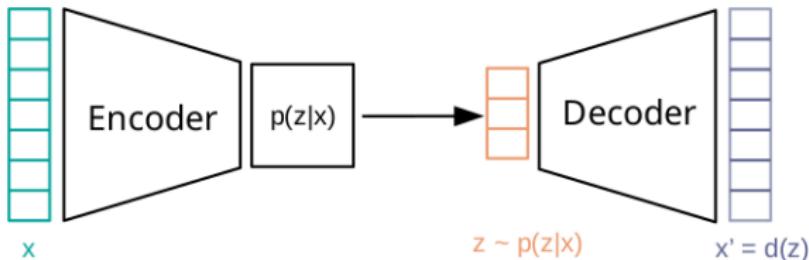


Figure: VAE framework

# VAE - Mathematical Considerations

- Let  $x \in \mathcal{X}$  be a set of data and  $\{P_\theta, \theta \in \Theta\}$  be a parametric model
- We assume there exists latent variables  $z \in \mathcal{Z}$  living in a smaller space such that the marginal likelihood writes

$$p_\theta(x) = \int p_\theta(x|z) q_{\text{prior}}(z) dz,$$

where  $q_{\text{prior}}$  is a prior distribution over the latent variables and  $p_\theta(x|z)$  is referred to as the decoder

- Example:

$$q_{\text{prior}} = \mathcal{N}(0, I), \quad p_\theta(x|z) = \prod_{i=1}^D \mathcal{B}(\pi_{\theta_i(z)})$$

## Objective:

- Maximizing the likelihood of the model

Problem: The integral is often intractable.

# VAE - Mathematical Considerations

- Let  $x \in \mathcal{X}$  be a set of data and  $\{P_\theta, \theta \in \Theta\}$  be a parametric model
- We assume there exists latent variables  $z \in \mathcal{Z}$  living in a smaller space such that the marginal likelihood writes

$$p_\theta(x) = \int p_\theta(x|z) q_{\text{prior}}(z) dz,$$

where  $q_{\text{prior}}$  is a prior distribution over the latent variables and  $p_\theta(x|z)$  is referred to as the decoder

- Example:

$$q_{\text{prior}} = \mathcal{N}(0, I), \quad p_\theta(x|z) = \prod_{i=1}^D \mathcal{B}(\pi_{\theta_i(z)})$$

## Objective:

- Maximizing the likelihood of the model

Problem: The integral is often intractable.

# VAE - Mathematical Considerations

- Let  $x \in \mathcal{X}$  be a set of data and  $\{P_\theta, \theta \in \Theta\}$  be a parametric model
- We assume there exists latent variables  $z \in \mathcal{Z}$  living in a smaller space such that the marginal likelihood writes

$$p_\theta(x) = \int p_\theta(x|z) q_{\text{prior}}(z) dz,$$

where  $q_{\text{prior}}$  is a prior distribution over the latent variables and  $p_\theta(x|z)$  is referred to as the decoder

- Example:

$$q_{\text{prior}} = \mathcal{N}(0, I), \quad p_\theta(x|z) = \prod_{i=1}^D \mathcal{B}(\pi_{\theta_i(z)})$$

## Objective:

- Maximizing the likelihood of the model

Problem: The integral is often intractable.

# VAE - Mathematical Considerations

- Let  $x \in \mathcal{X}$  be a set of data and  $\{P_\theta, \theta \in \Theta\}$  be a parametric model
- We assume there exists latent variables  $z \in \mathcal{Z}$  living in a smaller space such that the marginal likelihood writes

$$p_\theta(x) = \int p_\theta(x|z) q_{\text{prior}}(z) dz,$$

where  $q_{\text{prior}}$  is a prior distribution over the latent variables and  $p_\theta(x|z)$  is referred to as the decoder

- Example:

$$q_{\text{prior}} = \mathcal{N}(0, I), \quad p_\theta(x|z) = \prod_{i=1}^D \mathcal{B}(\pi_{\theta_i(z)})$$

## Objective:

- Maximizing the likelihood of the model

Problem: The integral is often intractable.

# VAE - Mathematical Considerations

- Let  $x \in \mathcal{X}$  be a set of data and  $\{P_\theta, \theta \in \Theta\}$  be a parametric model
- We assume there exists latent variables  $z \in \mathcal{Z}$  living in a smaller space such that the marginal likelihood writes

$$p_\theta(x) = \int p_\theta(x|z) q_{\text{prior}}(z) dz,$$

where  $q_{\text{prior}}$  is a prior distribution over the latent variables and  $p_\theta(x|z)$  is referred to as the decoder

- Example:

$$q_{\text{prior}} = \mathcal{N}(0, I), \quad p_\theta(x|z) = \prod_{i=1}^D \mathcal{B}(\pi_{\theta_i(z)})$$

## Objective:

- Maximizing the likelihood of the model

Problem: The integral is often intractable.

# VAE - Mathematical Considerations

One may write:

$$\begin{aligned}\log p_\theta(x) &= \log \left( \int p_\theta(x|z) q_{\text{prior}}(z) dz \right) \\ &= \log \left( \int p_\theta(x, z) dz \right) \\ &= \log \left( \int p_\theta(x, z) \frac{q(z)}{q(z)} dz \right), \text{ for any pdf } q \\ &\geq \int \left( \log \frac{p_\theta(x, z)}{q(z)} \right) q(z) dz, \text{ using Jensen's inequality} \\ &\geq \int (\log p_\theta(x, z)) q(z) dz - H(q(z))\end{aligned}$$

with  $H$  the entropy of  $q(z)$ .

The equality holds for  $q(z) = p_\theta(z|x)$ .

# VAE - Mathematical Considerations

One may write:

$$\begin{aligned}\log p_\theta(x) &= \log \left( \int p_\theta(x|z) q_{\text{prior}}(z) dz \right) \\ &= \log \left( \int p_\theta(x, z) dz \right) \\ &= \log \left( \int p_\theta(x, z) \frac{q(z)}{q(z)} dz \right), \text{ for any pdf } q \\ &\geq \int \left( \log \frac{p_\theta(x, z)}{q(z)} \right) q(z) dz, \text{ using Jensen's inequality} \\ &\geq \int (\log p_\theta(x, z)) q(z) dz - H(q(z))\end{aligned}$$

with  $H$  the entropy of  $q(z)$ .

The equality holds for  $q(z) = p_\theta(z|x)$ .

# VAE - Mathematical Considerations

One may write:

$$\begin{aligned}\log p_\theta(x) &= \log \left( \int p_\theta(x|z) q_{\text{prior}}(z) dz \right) \\ &= \log \left( \int p_\theta(x, z) dz \right) \\ &= \log \left( \int p_\theta(x, z) \frac{q(z)}{q(z)} dz \right), \text{ for any pdf } q \\ &\geq \int \left( \log \frac{p_\theta(x, z)}{q(z)} \right) q(z) dz, \text{ using Jensen's inequality} \\ &\geq \int (\log p_\theta(x, z)) q(z) dz - H(q(z))\end{aligned}$$

with  $H$  the entropy of  $q(z)$ .

The equality holds for  $q(z) = p_\theta(z|x)$ .

# VAE - Mathematical Considerations

One may write:

$$\begin{aligned}
 \log p_\theta(x) &= \log \left( \int p_\theta(x|z) q_{\text{prior}}(z) dz \right) \\
 &= \log \left( \int p_\theta(x, z) dz \right) \\
 &= \log \left( \int p_\theta(x, z) \frac{q(z)}{q(z)} dz \right), \text{ for any pdf } q \\
 &\geq \int \left( \log \frac{p_\theta(x, z)}{q(z)} \right) q(z) dz, \text{ using Jensen's inequality} \\
 &\geq \int (\log p_\theta(x, z)) q(z) dz - H(q(z))
 \end{aligned}$$

with  $H$  the entropy of  $q(z)$ .

The equality holds for  $q(z) = p_\theta(z|x)$ .

# VAE - Mathematical Considerations

One may write:

$$\begin{aligned}\log p_\theta(x) &= \log \left( \int p_\theta(x|z) q_{\text{prior}}(z) dz \right) \\ &= \log \left( \int p_\theta(x, z) dz \right) \\ &= \log \left( \int p_\theta(x, z) \frac{q(z)}{q(z)} dz \right), \text{ for any pdf } q \\ &\geq \int \left( \log \frac{p_\theta(x, z)}{q(z)} \right) q(z) dz, \text{ using Jensen's inequality} \\ &\geq \int (\log p_\theta(x, z)) q(z) dz - H(q(z))\end{aligned}$$

with  $H$  the entropy of  $q(z)$ .

The equality holds for  $q(z) = p_\theta(z|x)$ .

# VAE - Mathematical Considerations

One may write:

$$\begin{aligned}\log p_\theta(x) &= \log \left( \int p_\theta(x|z) q_{\text{prior}}(z) dz \right) \\ &= \log \left( \int p_\theta(x, z) dz \right) \\ &= \log \left( \int p_\theta(x, z) \frac{q(z)}{q(z)} dz \right), \text{ for any pdf } q \\ &\geq \int \left( \log \frac{p_\theta(x, z)}{q(z)} \right) q(z) dz, \text{ using Jensen's inequality} \\ &\geq \int (\log p_\theta(x, z)) q(z) dz - H(q(z))\end{aligned}$$

with  $H$  the entropy of  $q(z)$ .

The equality holds for  $q(z) = p_\theta(z|x)$ .

# The ELBO

- Well-known issue: the posterior  $q(z) = p_\theta(z|x)$  is intractable.  
→ use Expectation-Maximisation algorithms (up to the MCMC-SAEM version)
- OR** approximate this posterior with amortised variational inference → ELBO
- Introduce a parametric approximation:

$$q_\phi(z|x) \simeq p_\theta(z|x),$$

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

- This leads to an unbiased estimate of the log-likelihood

$$\hat{p}_\theta(x) = \frac{p_\theta(x, z)}{q_\phi(z|x)}, \quad \mathbb{E}_{z \sim q_\phi(z|x)}[\hat{p}_\theta(x)] = p_\theta(x),$$

- and the definition of the **Evidence Lower Bound (ELBO)**:

$$\begin{aligned} \log p_\theta(x) &\geq \mathbb{E}_{z \sim q_\phi(z|x)}[\log(p_\theta(x, z)) - \log(q_\phi(z|x))] \\ &\geq \text{ELBO} \end{aligned}$$

# The ELBO

- Well-known issue: the posterior  $q(z) = p_\theta(z|x)$  is intractable.  
→ use Expectation-Maximisation algorithms (up to the MCMC-SAEM version)
- OR** approximate this posterior with amortised variational inference → ELBO
- Introduce a parametric approximation:

$$q_\phi(z|x) \simeq p_\theta(z|x),$$

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

- This leads to an unbiased estimate of the log-likelihood

$$\hat{p}_\theta(x) = \frac{p_\theta(x, z)}{q_\phi(z|x)}, \quad \mathbb{E}_{z \sim q_\phi(z|x)}[\hat{p}_\theta(x)] = p_\theta(x),$$

- and the definition of the **Evidence Lower Bound (ELBO)**:

$$\begin{aligned} \log p_\theta(x) &\geq \mathbb{E}_{z \sim q_\phi(z|x)}[\log(p_\theta(x, z)) - \log(q_\phi(z|x))] \\ &\geq \text{ELBO} \end{aligned}$$

# The ELBO

- Well-known issue: the posterior  $q(z) = p_\theta(z|x)$  is intractable.  
→ use Expectation-Maximisation algorithms (up to the MCMC-SAEM version)
- OR** approximate this posterior with amortised variational inference → ELBO
- Introduce a parametric approximation:

$$q_\phi(z|x) \simeq p_\theta(z|x),$$

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

- This leads to an unbiased estimate of the log-likelihood

$$\hat{p}_\theta(x) = \frac{p_\theta(x, z)}{q_\phi(z|x)}, \quad \mathbb{E}_{z \sim q_\phi(z|x)}[\hat{p}_\theta(x)] = p_\theta(x),$$

- and the definition of the **Evidence Lower Bound (ELBO)**:

$$\begin{aligned} \log p_\theta(x) &\geq \mathbb{E}_{z \sim q_\phi(z|x)}[\log(p_\theta(x, z)) - \log(q_\phi(z|x))] \\ &\geq \text{ELBO} \end{aligned}$$

# The ELBO

- Well-known issue: the posterior  $q(z) = p_\theta(z|x)$  is intractable.  
→ use Expectation-Maximisation algorithms (up to the MCMC-SAEM version)
- OR** approximate this posterior with amortised variational inference → ELBO
- Introduce a parametric approximation:

$$q_\phi(z|x) \simeq p_\theta(z|x),$$

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

- This leads to an unbiased estimate of the log-likelihood

$$\hat{p}_\theta(x) = \frac{p_\theta(x, z)}{q_\phi(z|x)}, \quad \mathbb{E}_{z \sim q_\phi(z|x)}[\hat{p}_\theta(x)] = p_\theta(x),$$

- and the definition of the **Evidence Lower Bound** (ELBO):

$$\begin{aligned} \log p_\theta(x) &\geq \mathbb{E}_{z \sim q_\phi(z|x)}[\log(p_\theta(x, z)) - \log(q_\phi(z|x))] \\ &\geq \text{ELBO} \end{aligned}$$

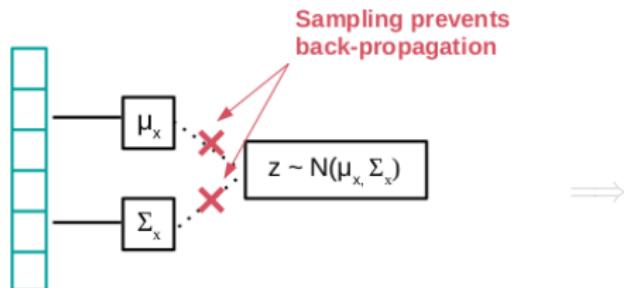
# Variational inference: The ELBO

Objective:

1. Optimise the ELBO **as a function** instead of the target distribution  
Use stochastic gradient descent in both  $\theta$  and  $\phi$

# The Reparametrisation Trick for stochastic gradient descent

- Since  $z \sim \mathcal{N}(\mu_\phi(x), \Sigma_\phi(x))$ , the model is not amenable to gradient descent w.r.t  $\phi$



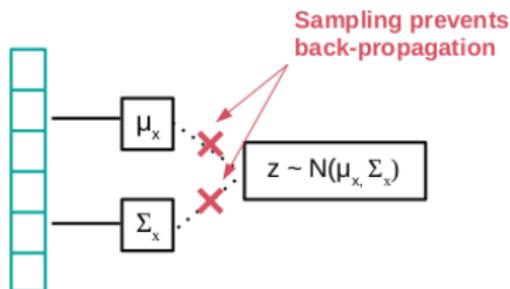
(a) Back-propagation impossible

⇒ Optimisation with respect to encoder and decoder parameters made possible !

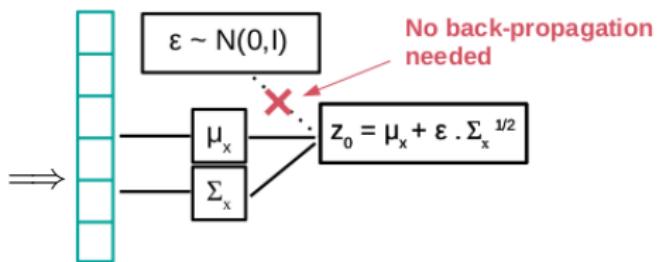
Objective ⇒ OK

# The Reparametrisation Trick for stochastic gradient descent

- Since  $z \sim \mathcal{N}(\mu_\phi(x), \Sigma_\phi(x))$ , the model is not amenable to gradient descent w.r.t  $\phi$



(a) Back-propagation impossible



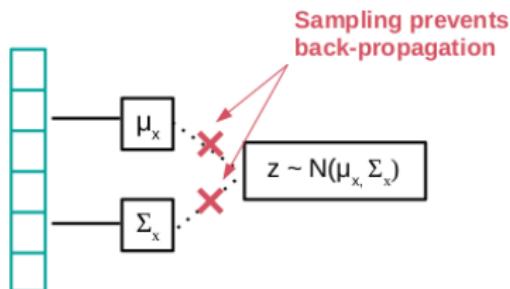
(b) Back-propagation possible

$\implies$  Optimisation with respect to encoder and decoder parameters made possible !

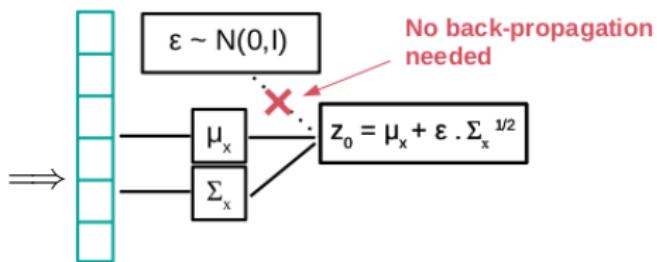
Objective  $\implies$  OK

# The Reparametrisation Trick for stochastic gradient descent

- Since  $z \sim \mathcal{N}(\mu_\phi(x), \Sigma_\phi(x))$ , the model is not amenable to gradient descent w.r.t  $\phi$



(a) Back-propagation impossible



(b) Back-propagation possible

$\implies$  Optimisation with respect to encoder and decoder parameters made possible !

**Objective  $\implies$  OK**

# Generating new samples

- We only need to sample  $z \sim \mathcal{N}(0, I)$  and feed it to the decoder.

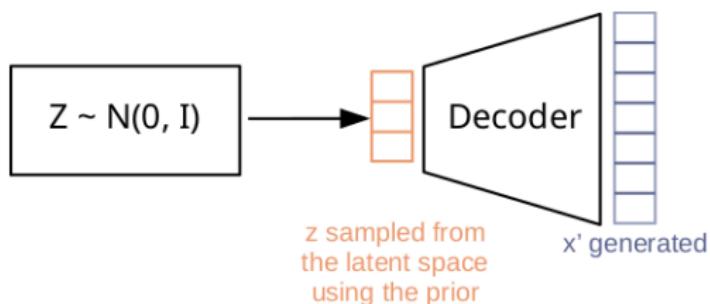


Figure: Generation procedure using prior

## Pros:

- Very simple to use in practice

## Cons:

- The prior and posterior are not expressive enough to capture complex distributions
- Poor latent space prospecting

# Generating new samples

- We only need to sample  $z \sim \mathcal{N}(0, I)$  and feed it to the decoder.

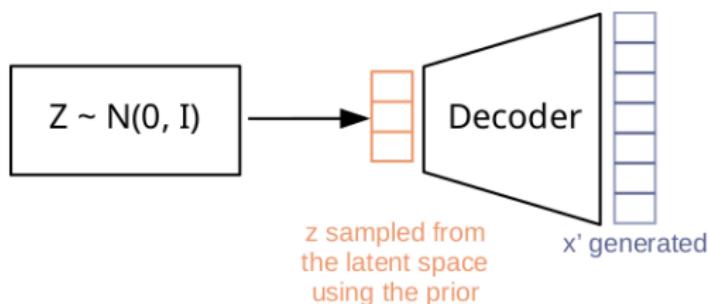


Figure: Generation procedure using prior

## Pros:

- Very simple to use in practice

## Cons:

- The prior and posterior are not expressive enough to capture complex distributions
- Poor latent space prospecting

# Generating new samples

- We only need to sample  $z \sim \mathcal{N}(0, I)$  and feed it to the decoder.

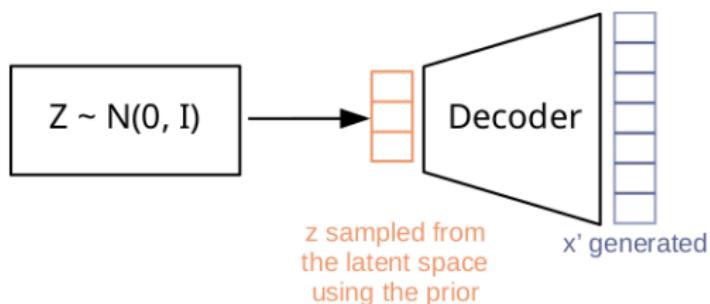


Figure: Generation procedure using prior

## Pros:

- Very simple to use in practice

## Cons:

- The prior and posterior are **not expressive enough** to capture complex distributions
- Poor latent space prospecting**

# VAE - Use Cases

- The VAE is a very versatile model that can be used to model complex distributions [17] such as *images* [32, 31], time series [5, 13], natural language [2], chemical structures [29], shapes [4] ...
- It can be used for various tasks as well

# Image Synthesis

- VAE as a generative model for image data



Figure: Samples from NVAE [31] on FFHQ [15]

# Data Augmentation

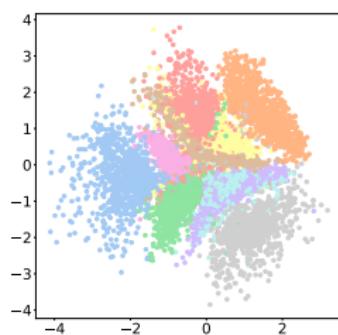
- VAE for Data Augmentation of 3D MRIs to enhance Alzheimer's automatic diagnosis [7]

training set	data set	ADNI			AIBL		
		sensitivity	specificity	balanced accuracy	sensitivity	specificity	balanced accuracy
train-50	real	70.3 ± 12.2	62.4 ± 11.5	66.3 ± 2.4	60.7 ± 13.7	73.8 ± 7.2	67.2 ± 4.1
	real (high-resolution)	78.5 ± 9.4	57.4 ± 8.8	67.9 ± 2.3	57.2 ± 11.2	75.8 ± 7.0	66.5 ± 3.0
	500 synthetic + real	71.9 ± 5.3	67.0 ± 4.5	69.4 ± 1.6	55.9 ± 6.8	81.1 ± 3.1	68.5 ± 2.5
	2000 synthetic + real	72.2 ± 4.4	70.3 ± 4.3	71.2 ± 1.6	66.6 ± 7.1	79.0 ± 4.1	72.8 ± 2.2
	5000 synthetic + real	<b>74.7 ± 5.3</b>	<b>73.5 ± 4.8</b>	<b>74.1 ± 2.2</b>	<b>71.7 ± 10.0</b>	80.5 ± 4.4	<b>76.1 ± 3.6</b>
	10000 synthetic + real	74.7 ± 7.0	73.4 ± 6.1	74.0 ± 2.7	69.1 ± 9.9	<b>80.7 ± 5.1</b>	74.9 ± 3.2

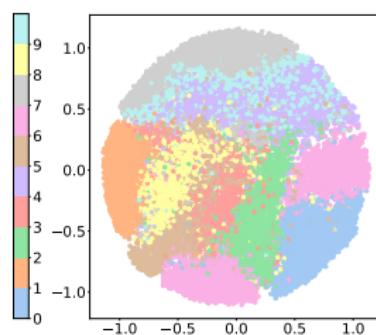
Figure: Classification results with State of the art CNN for Alzheimer disease from [7]

# Clustering

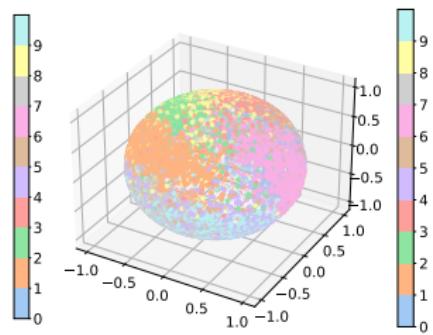
- VAE for clustering



$\mathcal{N}$ -VAE



$\mathcal{P}$ -VAE



$\mathcal{S}$ -VAE

**Figure:** 2-dimensional latent spaces learned by a vanilla VAE ( $\mathcal{N}$ -VAE), Poincaré VAE ( $\mathcal{P}$ -VAE) and Hyperspherical VAE ( $\mathcal{S}$ -VAE) on MNIST. The colors represent the digits. Plots are made using [8]

# Feature Extraction

- VAE used as feature extractor (e.g. Stable diffusion) [27]

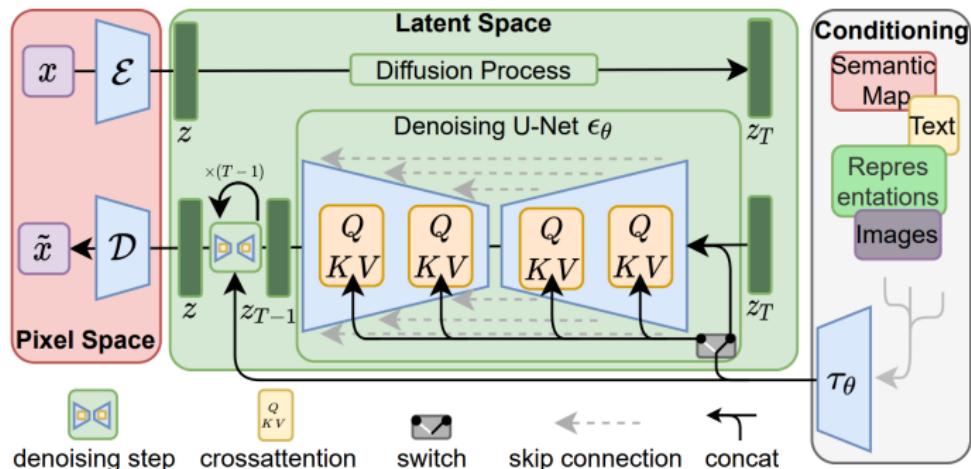


Figure: Latent Diffusion Model architecture [27]

## Improving the model

Can we do better?

# Tweaking the Approximate Posterior Distribution

- The ELBO can be written as

$$\text{ELBO} = \log p_\theta(x) - \underbrace{\text{KL}(q_\phi(z|x) || p_\theta(z|x))}_{\approx 0 \text{ if } q_\phi(z|x) \approx p_\theta(z|x)} .$$

- Since the Kullback-Leibler divergence is always non-negative, the objective is to try to make it vanish by tweaking the approximate posterior  $q_\phi(z|x)$

# Tweaking the Approximate Posterior Distribution

- The ELBO can be written as

$$\text{ELBO} = \log p_\theta(x) - \underbrace{\text{KL}(q_\phi(z|x)||p_\theta(z|x))}_{\approx 0 \text{ if } q_\phi(z|x) \approx p_\theta(z|x)} .$$

- Since the Kullback-Leibler divergence is always non-negative, the objective is to try to make it vanish by tweaking the approximate posterior  $q_\phi(z|x)$

# Normalizing Flows

- The idea is to use smooth invertible parameterised mappings  $f_\psi$  to “sample”  $z$  [25]
- $K$  transformations are then applied to a latent variable  $z_0$  drawn from an initial distribution  $q$  (here  $q = q_\phi$ ) leading to a final random variable  $z_K = f_x^K \circ \dots \circ f_x^1(z_0)$  whose density writes

$$q_\phi(z_K|x) = q_\phi(z_0|x) \prod_{k=1}^K |\det \mathbf{J}_{f_x^k}|^{-1},$$

- E.g. Planar flows [25], NICE [10], radial flows [25], RealNVP [11], Masked Autoregressive Flows (MAF) [22] or Inverse Autoregressive Flows (IAF) [18]

# Normalizing Flows

- The idea is to use smooth invertible parameterised mappings  $f_\psi$  to “sample”  $z$  [25]
- $K$  transformations are then applied to a latent variable  $z_0$  drawn from an initial distribution  $q$  (here  $q = q_\phi$ ) leading to a final random variable  $z_K = f_x^K \circ \dots \circ f_x^1(z_0)$  whose density writes

$$q_\phi(z_K|x) = q_\phi(z_0|x) \prod_{k=1}^K |\det \mathbf{J}_{f_x^k}|^{-1},$$

- E.g. Planar flows [25], NICE [10], radial flows [25], RealNVP [11], Masked Autoregressive Flows (MAF) [22] or Inverse Autoregressive Flows (IAF) [18]

# Normalizing Flows

- The idea is to use smooth invertible parameterised mappings  $f_\psi$  to “sample”  $z$  [25]
- $K$  transformations are then applied to a latent variable  $z_0$  drawn from an initial distribution  $q$  (here  $q = q_\phi$ ) leading to a final random variable  $z_K = f_x^K \circ \dots \circ f_x^1(z_0)$  whose density writes

$$q_\phi(z_K|x) = q_\phi(z_0|x) \prod_{k=1}^K |\det \mathbf{J}_{f_x^k}|^{-1},$$

- E.g. Planar flows [25], NICE [10], radial flows [25], RealNVP [11], Masked Autoregressive Flows (MAF) [22] or Inverse Autoregressive Flows (IAF) [18]

# Auxiliary Latent Variables

- Idea: Work with an extended space by adding an *auxiliary* continuous random variable  $u \in \mathcal{U}$  and consider an augmented inference model [28, 19, 23]

$$q_\phi(u, z|x) = q_\phi(u|x)q_\phi(z|u, x).$$

- $u$  allows to access to a potentially richer class of  $q_\phi(z|x)$  since

$$q_\phi(z|x) = \int_{\mathcal{U}} q_\phi(u, z|x)du.$$

- The extended generative model follows

$$p_\theta(x, z, u) = p_\theta(u|x, z)p_\theta(x, z).$$

# Auxiliary Latent Variables

- Idea: Work with an extended space by adding an *auxiliary* continuous random variable  $u \in \mathcal{U}$  and consider an augmented inference model [28, 19, 23]

$$q_\phi(u, z|x) = q_\phi(u|x)q_\phi(z|u, x).$$

- $u$  allows to access to a potentially richer class of  $q_\phi(z|x)$  since

$$q_\phi(z|x) = \int_{\mathcal{U}} q_\phi(u, z|x)du.$$

- The extended generative model follows

$$p_\theta(x, z, u) = p_\theta(u|x, z)p_\theta(x, z).$$

# Auxiliary Latent Variables

- Idea: Work with an extended space by adding an *auxiliary* continuous random variable  $u \in \mathcal{U}$  and consider an augmented inference model [28, 19, 23]

$$q_\phi(u, z|x) = q_\phi(u|x)q_\phi(z|u, x).$$

- $u$  allows to access to a potentially richer class of  $q_\phi(z|x)$  since

$$q_\phi(z|x) = \int_{\mathcal{U}} q_\phi(u, z|x)du.$$

- The extended generative model follows

$$p_\theta(x, z, u) = p_\theta(u|x, z)p_\theta(x, z).$$

# Auxiliary Latent Variables

- In a similar fashion as Eq. (1), one can build an unbiased estimator of the marginal likelihood  $p_\theta(x)$

$$\hat{p}_\theta(x) = \frac{p_\theta(x, z, u)}{q_\phi(u, z|x)} \text{ and } \mathbb{E}_{(u,z) \sim q_\phi} [\hat{p}_\theta] = p_\theta(x).$$

- This allows to derive an ELBO

$$\begin{aligned} \log p_\theta(x) &= \mathbb{E}_{(u,z) \sim q_\phi} [\hat{p}_\theta(x)] , \\ &\geq \mathbb{E}_{(u,z) \sim q_\phi} \left[ \log \left( \frac{p_\theta(x, z, u)}{q_\phi(u, z|x)} \right) \right] = \mathcal{L}_{\text{aux}}(\theta, \phi, x). \end{aligned}$$

- E.g. Hierarchical VAEs [23], Hamiltonian VAE [6], Riemannian Hamiltonian VAE [7], MCMC VAE [28, 30]

# Auxiliary Latent Variables

- In a similar fashion as Eq. (1), one can build an unbiased estimator of the marginal likelihood  $p_\theta(x)$

$$\hat{p}_\theta(x) = \frac{p_\theta(x, z, u)}{q_\phi(u, z|x)} \text{ and } \mathbb{E}_{(u,z) \sim q_\phi} [\hat{p}_\theta] = p_\theta(x).$$

- This allows to derive an ELBO

$$\begin{aligned} \log p_\theta(x) &= \mathbb{E}_{(u,z) \sim q_\phi} [\hat{p}_\theta(x)] , \\ &\geq \mathbb{E}_{(u,z) \sim q_\phi} \left[ \log \left( \frac{p_\theta(x, z, u)}{q_\phi(u, z|x)} \right) \right] = \mathcal{L}_{\text{aux}}(\theta, \phi, x). \end{aligned}$$

- E.g.* Hierarchical VAEs [23], Hamiltonian VAE [6], Riemannian Hamiltonian VAE [7], MCMC VAE [28, 30]

# Building Better Estimators

- The estimator used in the vanilla VAE is given by

$$\hat{p}_\theta(x) = \frac{p_\theta(x, z)}{q_\phi(z|x)} \text{ and } \mathbb{E}_{z \sim q_\phi} [\hat{p}_\theta] = p_\theta(x). \quad (1)$$

- Several approaches proposed to build more complex estimators of the marginal likelihood  $p_\theta(x)$  [3, 20, 12, 30]
- E.g* Importance Weighted AutoEncoder (IWAE) that uses an ELBO derived from the  $K$ -sample importance weighted estimator.

$$\hat{p}_\theta(x) = \frac{1}{K} \sum_{i=1}^K \frac{p_\theta(x, z_i)}{q_\phi(z_i|x)} \text{ and } \mathbb{E}_{z_1, \dots, z_K \sim q_\phi(z|x)} [\hat{p}_\theta] = p_\theta(x).$$

$$\mathcal{L}_{\text{IWAE}}(\theta, \phi, x) = \mathbb{E}_{z_1, \dots, z_K \sim q_\phi(z|x)} \left[ \log \frac{1}{K} \sum_{i=1}^K \frac{p_\theta(x, z_i)}{q_\phi(z_i|x)} \right]$$

# Building Better Estimators

- The estimator used in the vanilla VAE is given by

$$\hat{p}_\theta(x) = \frac{p_\theta(x, z)}{q_\phi(z|x)} \text{ and } \mathbb{E}_{z \sim q_\phi} [\hat{p}_\theta] = p_\theta(x).$$

- Several approaches proposed to build more complex estimators of the marginal likelihood  $p_\theta(x)$  [3, 20, 12, 30]
- E.g* Importance Weighted AutoEncoder (IWAE) that uses an ELBO derived from the  $K$ -sample importance weighted estimator.

$$\hat{p}_\theta(x) = \frac{1}{K} \sum_{i=1}^K \frac{p_\theta(x, z_i)}{q_\phi(z_i|x)} \text{ and } \mathbb{E}_{z_1, \dots, z_K \sim q_\phi(z|x)} [\hat{p}_\theta] = p_\theta(x).$$

$$\mathcal{L}_{\text{IWAE}}(\theta, \phi, x) = \mathbb{E}_{z_1, \dots, z_K \sim q_\phi(z|x)} \left[ \log \frac{1}{K} \sum_{i=1}^K \frac{p_\theta(x, z_i)}{q_\phi(z_i|x)} \right]$$

# Building Better Estimators

- The estimator used in the vanilla VAE is given by

$$\hat{p}_\theta(x) = \frac{p_\theta(x, z)}{q_\phi(z|x)} \text{ and } \mathbb{E}_{z \sim q_\phi} [\hat{p}_\theta] = p_\theta(x).$$

- Several approaches proposed to build more complex estimators of the marginal likelihood  $p_\theta(x)$  [3, 20, 12, 30]
- E.g* Importance Weighted AutoEncoder (IWAE) that uses an ELBO derived from the  $K$ -sample importance weighted estimator.

$$\hat{p}_\theta(x) = \frac{1}{K} \sum_{i=1}^K \frac{p_\theta(x, z_i)}{q_\phi(z_i|x)} \text{ and } \mathbb{E}_{z_1, \dots, z_K \sim q_\phi(z|x)} [\hat{p}_\theta] = p_\theta(x).$$

$$\mathcal{L}_{\text{IWAE}}(\theta, \phi, x) = \mathbb{E}_{z_1, \dots, z_K \sim q_\phi(z|x)} \left[ \log \frac{1}{K} \sum_{i=1}^K \frac{p_\theta(x, z_i)}{q_\phi(z_i|x)} \right]$$

# Rethinking our Priors

- Recall the vanilla VAE ELBO

$$\mathcal{L}(\theta, \phi, x) = \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z)] - \text{KL}(q_\phi(z|x) || \mathbf{p}(z)).$$

- One may show that the prior maximising the ELBO is given by the aggregated posterior

$$q^{\text{avg}}(z) = \frac{1}{N} \sum_{i=1}^N q_\phi(z|x_i)$$

- However, it can lead to overfitting and is hard to use in practice

# Rethinking our Priors

- Recall the vanilla VAE ELBO

$$\mathcal{L}(\theta, \phi, x) = \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z)] - \text{KL}(q_\phi(z|x) || \mathbf{p}(z)) .$$

- One may show that the prior maximising the ELBO is given by the *aggregated* posterior

$$q^{\text{avg}}(z) = \frac{1}{N} \sum_{i=1}^N q_\phi(z|x_i)$$

- However, it can lead to overfitting and is hard to use in practice

# Rethinking our Priors

- Recall the vanilla VAE ELBO

$$\mathcal{L}(\theta, \phi, x) = \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z)] - \text{KL}(q_\phi(z|x) || \mathbf{p}(z)).$$

- One may show that the prior maximising the ELBO is given by the *aggregated* posterior

$$q^{\text{avg}}(z) = \frac{1}{N} \sum_{i=1}^N q_\phi(z|x_i)$$

- However, it can lead to overfitting and is hard to use in practice

# Rethinking our Priors

Several axis of development were proposed in the literature to improve the generative capability of the model and reduced the regularisation coming from the prior.

- Approximate the *aggregated posterior*:

$$p_{\lambda}^{\text{VAMP}}(z) = \frac{1}{K} \sum_{i=1}^K q_{\phi}(z|u_k),$$

where  $\lambda$  corresponds to the prior's parameters  $\lambda = \{\phi, u_1, \dots, u_K\}$ .

- Learn the prior during training [9, 24, 21, 1]
- post-training density estimation with Gaussian mixture or flows [32, 14]  
     $\implies$  Estimate density of the latent code

# Rethinking our Priors

Several axis of development were proposed in the literature to improve the generative capability of the model and reduced the regularisation coming from the prior.

- Approximate the *aggregated posterior*:

$$p_{\lambda}^{\text{VAMP}}(z) = \frac{1}{K} \sum_{i=1}^K q_{\phi}(z|u_k),$$

where  $\lambda$  corresponds to the prior's parameters  $\lambda = \{\phi, u_1, \dots, u_K\}$ .

- Learn the prior during training [9, 24, 21, 1]
  - post-training density estimation with Gaussian mixture or flows [32, 14]  
⇒ Estimate density of the latent code

# Rethinking our Priors

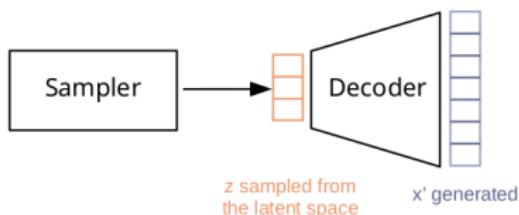
Several axis of development were proposed in the literature to improve the generative capability of the model and reduced the regularisation coming from the prior.

- Approximate the *aggregated posterior*:

$$p_{\lambda}^{\text{VAMP}}(z) = \frac{1}{K} \sum_{i=1}^K q_{\phi}(z|u_k),$$

where  $\lambda$  corresponds to the prior's parameters  $\lambda = \{\phi, u_1, \dots, u_K\}$ .

- Learn the prior during training [9, 24, 21, 1]
- post-training density estimation with Gaussian mixture or flows [32, 14]  
⇒ Estimate density of the latent code



# Training VAEs with Pythae

Let's Train VAEs

# What is Pythae?

- Pythae is a Python library that implements some of the most common VAEs models



[Documentation](#)

## pythae

---

This library implements some of the most common (Variational) Autoencoder models under a unified implementation. In particular, it provides the possibility to perform benchmark experiments and comparisons by training the models with the same autoencoding neural network architecture. The feature *make your own autoencoder* allows you to train any of these models with your own data and own Encoder and Decoder neural networks. It integrates experiment monitoring tools such [wandb](#), [mlflow](#) or [comet-ml](#) and allows model sharing and loading from the [HuggingFace Hub](#) in a few lines of code.

### News 🎉

As of v0.1.0, Pythae now supports distributed training using PyTorch's [DDP](#). You can now train your favorite VAE faster and on larger datasets, still with a few lines of code. See our speed-up [benchmark](#).

# Why Pythae ?

- **Unifying implementations**

- ✗ Existing implementations may be *difficult to adapt* to other use-cases, be in *different frameworks* or *no longer maintained*.
  - ✓ **Pythae's brick-like structure** allows for seamless but efficient interchange between models, sampling techniques, network architectures, model hyper-parameters and training schemes.

- A reproducible research environment

- ✗ Reproducibility is hard: implementations may *no longer maintained* or *unavailable*.
  - ✓ Pythae reproduced most of the most popular GAE methods (when code was available or enough information provided in the paper).

- Usable by all

- ✗ Existing codes may only allow reproduction of specific results available in the paper.
  - ✓ Pythae makes GAE models accessible to beginners and experts. The library has an [online documentation](#) and is also illustrated through tutorials available either on a local machine or on the Google Colab platform.

# Why Pythae ?

- **Unifying implementations**

- ✗ Existing implementations may be *difficult to adapt* to other use-cases, be in *different frameworks* or *no longer maintained*.
  - ✓ **Pythae's brick-like structure** allows for seamless but efficient interchange between models, sampling techniques, network architectures, model hyper-parameters and training schemes.

- **A reproducible research environment**

- ✗ Reproducibility is hard: implementations may *no longer maintained* or *unavailable*.
  - ✓ **Pythae** reproduced most of the most popular GAE methods (when code was available or enough information provided in the paper).

- **Usable by all**

- ✗ Existing codes may only allow reproduction of specific results available in the paper.
  - ✓ Pythae makes GAE models accessible to beginners and experts. The library has an online documentation and is also illustrated through tutorials available either on a local machine or on the Google Colab platform.

# Why Pythae ?

- **Unifying implementations**

- ✗ Existing implementations may be *difficult to adapt* to other use-cases, be in *different frameworks* or *no longer maintained*.
  - ✓ **Pythae's brick-like structure** allows for seamless but efficient interchange between models, sampling techniques, network architectures, model hyper-parameters and training schemes.

- **A reproducible research environment**

- ✗ Reproducibility is hard: implementations may *no longer maintained* or *unavailable*.
  - ✓ **Pythae** reproduced most of the most popular GAE methods (when code was available or enough information provided in the paper).

- **Usable by all**

- ✗ Existing codes may only allow reproduction of specific results available in the paper.
  - ✓ **Pythae** makes GAE models accessible to beginners and experts. The library has an **online documentation** and is also **illustrated through tutorials** available either on a local machine or on the Google Colab platform.

# Pythae - Code structure

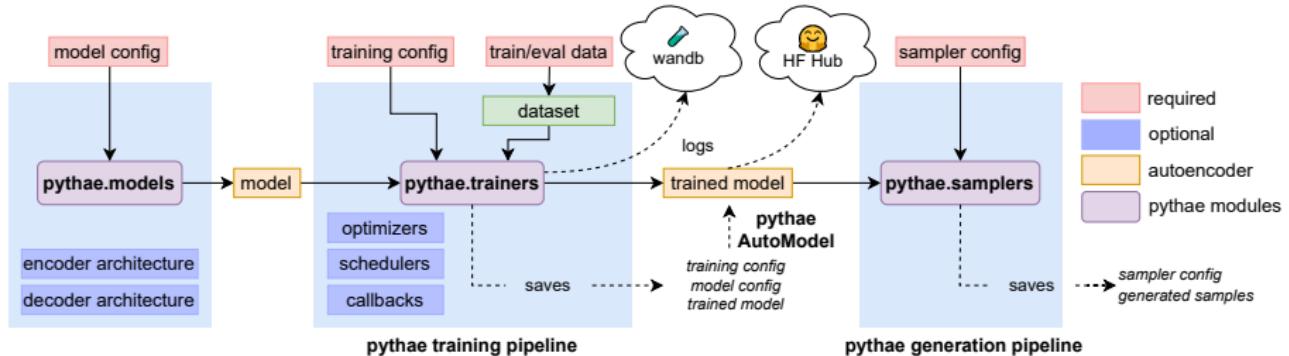


Figure: Code structure

# Pythae - API

## Architecture Definition

```
● ● ●

from pythae.models.nn import BaseEncoder, BaseDecoder
from pythae.models.base.base_utils import ModelOutput

# Define encoder architecture
class My_Encoder(BaseEncoder):
    def __init__(self):
        BaseEncoder.__init__(self)
        self.layers = my_nn.layers()

    def forward(self, x: torch.Tensor) -> ModelOutput:
        out = self.layers(x)
        output = ModelOutput(embedding=out)
        return output

# Define decoder architecture
class My_Decoder(BaseDecoder):
    def __init__(self):
        BaseDecoder.__init__(self)
        self.layers = my_nn.layers()

    def forward(self, x: torch.Tensor) -> ModelOutput:
        out = self.layers(x)
        output = ModelOutput(reconstruction=out)
        return output

# Instantiate your encoder and decoder
my_encoder = My_Encoder()
my_decoder = My_Decoder()
```

# Pythae - API

## Architecture Definition

```
● ● ●  
from pythae.models.nn import BaseEncoder, BaseDecoder  
from pythae.models.base.base_utils import ModelOutput  
  
# Define encoder architecture  
class My_Encoder(BaseEncoder):  
    def __init__(self):  
        BaseEncoder.__init__(self)  
        self.layers = my_nn.layers()  
  
    def forward(self, x: torch.Tensor) -> ModelOutput:  
        out = self.layers(x)  
        output = ModelOutput(embedding=out)  
        return output  
  
# Define decoder architecture  
class My_Decoder(BaseDecoder):  
    def __init__(self):  
        BaseDecoder.__init__(self)  
        self.layers = my_nn.layers()  
  
    def forward(self, x: torch.Tensor) -> ModelOutput:  
        out = self.layers(x)  
        output = ModelOutput(reconstruction=out)  
        return output  
  
# Instantiate your encoder and decoder  
my_encoder = My_Encoder()  
my_decoder = My_Decoder()
```

## Training

```
● ● ●  
from pythae.pipelines import TrainingPipeline  
from pythae.models import VAE, VAEConfig  
from pythae.trainers import BaseTrainerConfig  
  
# Set up the training configuration  
my_training_config = BaseTrainerConfig(...)  
  
# Set up the model configuration  
model_config = VAEConfig(...)  
  
# Build the model  
my_vae_model = VAE(  
    model_config=my_vae_config,  
    encoder=my_encoder,  
    decoder=my_decoder  
)  
  
# Build the pipeline  
pipeline = TrainingPipeline(  
    training_config=my_training_config,  
    model=my_vae_model  
)  
  
# Launch the pipeline  
pipeline(  
    train_data=your_train_data,  
    eval_data=your_eval_data  
)
```

# Pythae - API

## Architecture Definition

```

from pythae.models.nn import BaseEncoder, BaseDecoder
from pythae.models.base.base_utils import ModelOutput

# Define encoder architecture
class My_Encoder(BaseEncoder):
    def __init__(self):
        BaseEncoder.__init__(self)
        self.layers = my_nn.layers()

    def forward(self, x: torch.Tensor) -> ModelOutput:
        out = self.layers(x)
        output = ModelOutput(embedding=out)
        return output

# Define decoder architecture
class My_Decoder(BaseDecoder):
    def __init__(self):
        BaseDecoder.__init__(self)
        self.layers = my_nn.layers()

    def forward(self, x: torch.Tensor) -> ModelOutput:
        out = self.layers(x)
        output = ModelOutput(reconstruction=out)
        return output

# Instantiate your encoder and decoder
my_encoder = My_Encoder()
my_decoder = My_Decoder()

```

## Training

```

from pythae.pipelines import TrainingPipeline
from pythae.models import VAE, VAEConfig
from pythae.trainers import BaseTrainerConfig

# Set up the training configuration
my_training_config = BaseTrainerConfig(...)

# Set up the model configuration
model_config = VAEConfig(...)

# Build the model
my_vae_model = VAE(
    model_config=my_vae_config,
    encoder=my_encoder,
    decoder=my_decoder
)

# Build the pipeline
pipeline = TrainingPipeline(
    training_config=my_training_config,
    model=my_vae_model
)

# Launch the pipeline
pipeline(
    train_data=your_train_data,
    eval_data=your_eval_data
)

```

## Data Generation

```

from pythae.models import AutoModel
from pythae.samplers import GaussianMixtureSamplerConfig
from pythae.pipelines import GenerationPipeline

# Retrieve the trained model
my_trained_vae = AutoModel.load_from_folder(
    'path/to/your/trained/model'
)

# Set up the sampler configuration
my_sampler_config = GaussianMixtureSamplerConfig(
    n_components=10
)

# Build the pipeline
pipeline = GenerationPipeline(
    model=my_trained_vae,
    sampler_config=my_sampler_config
)

# Launch data generation
generated_samples = pipeline(
    num_samples=100,
    return_gen=True,
    train_data=train_data,
    eval_data=None,
)

```

# Zoom on Configurations

- How to define my training configuration?

```
● ● ●  
from pytha.trainers import BaseTrainerConfig  
  
# Set up the training configuration  
my_training_config = BaseTrainerConfig(  
    output_dir='my_model',  
    num_epochs=50,  
    learning_rate=1e-3,  
    per_device_train_batch_size=200,  
    per_device_eval_batch_size=200,  
    train_dataloader_num_workers=2,  
    eval_dataloader_num_workers=2,  
    steps_saving=20,  
    optimizer_cls="AdamW",  
    optimizer_params={"weight_decay": 0.05, "betas": (0.91, 0.995)},  
    scheduler_cls="ReduceLROnPlateau",  
    scheduler_params={"patience": 5, "factor": 0.5}  
)
```

Figure: Example of training configuration

# Zoom on Configurations

- How to define my model configuration?

```
● ● ●  
from pythaе.models import WAE_MMD, WAE_MMD_Config  
  
# Set up the model configuration  
my_wae_config = WAE_MMD_Config(  
    input_dim=(1, 28, 28),  
    latent_dim=10,  
    kernel_choice="imq",  
    reg_weight=0.01  
)  
  
# Build the model  
my_wae_model = WAE_MMD(  
    model_config=my_wae_config,  
    encoder=my_encoder, # pass your encoder as argument when building the model  
    decoder=my_decoder # pass your decoder as argument when building the model  
)
```

Figure: Example of model configuration

# Distributed Training with Pythae

- Pythae also support Distributed training using PyTorch DDP

```
● ● ●  
training_config = BaseTrainerConfig(  
    num_epochs=10,  
    learning_rate=1e-3,  
    per_device_train_batch_size=64,  
    per_device_eval_batch_size=64,  
    train_dataloader_num_workers=8,  
    eval_dataloader_num_workers=8,  
    dist_backend="nccl", # distributed backend  
    world_size=8 # number of gpus to use (n_nodes x n_gpus_per_node),  
    rank=5 # process/gpu id,  
    local_rank=1 # node id,  
    master_addr="localhost" # master address,  
    master_port="12345" # master port,  
)
```

Figure: Training configuration in a distributed setting

# Pythae - Implemented Models

GAE Model	Pythae model
Autoencoder	AE
Variational Autoencoder	VAE
Beta Variational Autoencoder	BetaVAE
VAE with Linear Normalizing Flows	VAE_LinNF
VAE with Inverse Autoregressive Flows	VAE_IAF
Disentangled $\beta$ -VAE	DisentangledBetaVAE
Disentangling by Factorising	FactorVAE
Beta-TC-VAE	BetaTCVAE
Importance Weighted Autoencoder	IWAE
Multiply Importance Weighted Autoencoder	MIWAE
Partially Importance Weighted Autoencoder	PIWAE
Combination Importance Weighted Autoencoder	CIWAE
VAE with perceptual metric similarity	MSSIM_VAE
Wasserstein Autoencoder	WAE
Info Variational Autoencoder	INFOVAE_MMD
VAMP Autoencoder	VAMP
Hyperspherical VAE	SVAE
Poincaré Disk VAE	PoincaréVAE
Adversarial Autoencoder	Adversarial_AE
Variational Autoencoder GAN	VAEGAN
Vector Quantized VAE	VQVAE
Hamiltonian VAE	HVAE
Regularized AE with L2 decoder param	RAE_L2
Regularized AE with gradient penalty	RAE_GP
Riemannian Hamiltonian VAE	RHVAE

Figure: 25 implemented models

# Pythae - Experiments monitoring

Pythae integrates experiment monitoring tools



```
from pythae.trainers.training_callbacks import WandbCallback
callbacks = []

# Build the callback
wandb_cb = WandbCallback()

# Set up the callback
wandb_cb.setup(
    training_config=your_training_config,
    model_config=your_model_config,
    project_name='your_wandb_project',
    entity_name='your_wandb_entity',
)

# Add it to the callbacks list
callbacks.append(wandb_cb)
```

```
from pythae.trainers.training_callbacks import MLFlowCallback
callbacks = []

# Build the callback
mlflow_cb = MLFlowCallback() # Build the callback

# Set up the callback
mlflow_cb.setup(
    training_config=your_training_config,
    model_config=your_model_config,
    run_name='mlflow_cb_example',
)

# Add it to the callbacks list
callbacks.append(mlflow_cb)
```

```
from pythae.trainers.training_callbacks import CometCallback
callbacks = []

# Build the callback
comet_cb = CometCallback() # Build the callback

# Set up the callback
comet_cb.setup(
    training_config=training_config,
    model_config=model_config,
    api_key='your_comet_api_key',
    project_name='your_comet_project',
)

# Add it to the callbacks list
callbacks.append(wandb_cb)
```

## Callback set-up

# Pythae - Experiments monitoring

Pythae integrates experiment monitoring tools



```
from pythae.trainers.training_callbacks import WandbCallback
callbacks = []
# Build the callback
wandb_cb = WandbCallback()
# Set up the callback
wandb_cb.setup(
    training_config=your_training_config,
    model_config=your_model_config,
    project_name='your_wandb_project',
    entity_name='your_wandb_entity',
)
# Add it to the callbacks list
callbacks.append(wandb_cb)
```



```
from pythae.trainers.training_callbacks import MlflowCallback
callbacks = []
# Build the callback
mlflow_cb = MlflowCallback() # Build the callback
# Set up the callback
mlflow_cb.setup(
    training_config=your_training_config,
    model_config=your_model_config,
    run_name='mlflow_cb_example',
)
# Add it to the callbacks list
callbacks.append(mlflow_cb)
```



```
from pythae.trainers.training_callbacks import CometCallback
callbacks = []
# Build the callback
comet_cb = CometCallback() # Build the callback
# Set up the callback
comet_cb.setup(
    training_config=training_config,
    model_config=model_config,
    api_key='your_comet_api_key',
    project_name='your_comet_project',
)
# Add it to the callbacks list
callbacks.append(wandb_cb)
```

## Callback set-up

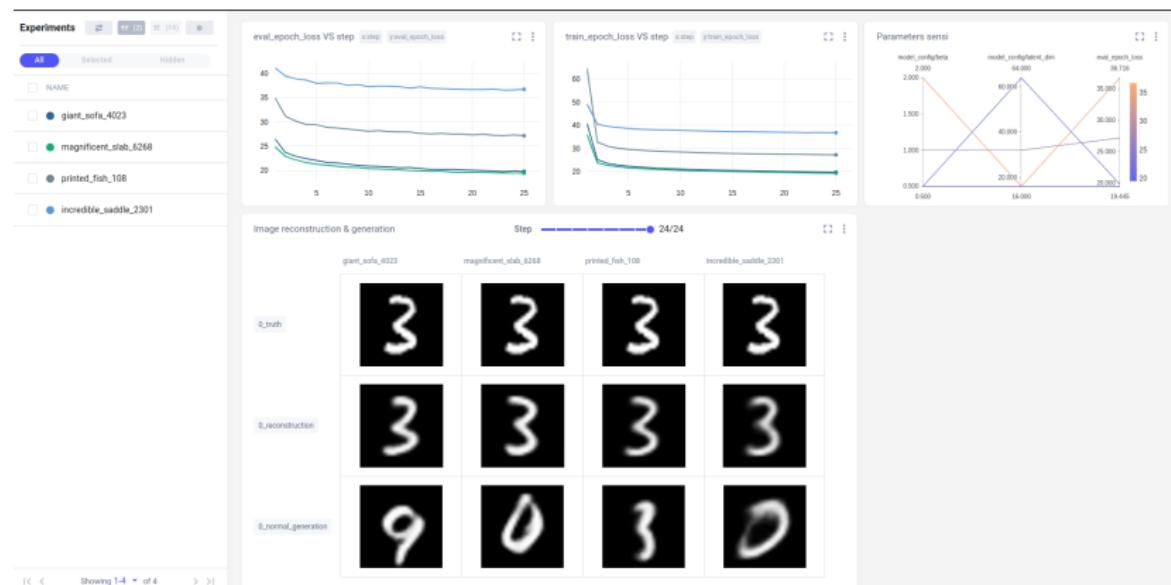
```
pipeline = TrainingPipeline(
    training_config=config,
    model=model
)
pipeline(
    train_data=train_dataset,
    eval_data=eval_dataset,
    callbacks=callbacks
)
```

```
pipeline = TrainingPipeline(
    training_config=config,
    model=model
)
pipeline(
    train_data=train_dataset,
    eval_data=eval_dataset,
    callbacks=callbacks
)
```

```
pipeline = TrainingPipeline(
    training_config=config,
    model=model
)
pipeline(
    train_data=train_dataset,
    eval_data=eval_dataset,
    callbacks=callbacks
)
```

## Callback usage

# Pythae - Experiments monitoring



# Pythae - Model sharing

Pythae allows model sharing through the HuggingFace Hub



```
my_vae_model.push_to_hf_hub(hf_hub_path="your_hf_username/your_hf_hub_repo")
```

Model saving

# Pythae - Model sharing

Pythae allows model sharing through the HuggingFace Hub



```
my_vae_model.push_to_hf_hub(hf_hub_path="your_hf_username/your_hf_hub_repo")
```

Model saving

```
from pythae.models import AutoModel  
my_downloaded_vae = AutoModel.load_from_hf_hub(hf_hub_path="path_to_hf_repo")
```

Model loading

# Pythae - Resources

# Thank you!

- ✓ Github: [https://github.com/clementchadebec/benchmark\\_VAE](https://github.com/clementchadebec/benchmark_VAE)
- ✓ Online documentation: <https://pythae.readthedocs.io/en/latest/>
- ✓ Pypi project page: <https://pypi.org/project/pythae/>
- ✓ Open to contributors!



```
pip install pythae
```

# Bibliography I

- [1] Jyoti Aneja, Alexander Schwing, Jan Kautz, and Arash Vahdat. NCP-VAE: Variational autoencoders with noise contrastive priors. *arXiv:2010.02917 [cs, stat]*, 2020.
- [2] Samuel Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 10–21, 2016.
- [3] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv:1509.00519 [cs, stat]*, 2016.
- [4] Alexandre Bône, Maxime Louis, Olivier Colliot, Stanley Durrleman, Alzheimer's Disease Neuroimaging Initiative, and others. Learning low-dimensional representations of shape data sets with diffeomorphic autoencoders. In *International Conference on Information Processing in Medical Imaging*, pages 195–207. Springer, 2019.
- [5] Francesco Paolo Casale, Adrian Dalca, Luca Saglietti, Jennifer Listgarten, and Nicolo Fusi. Gaussian process prior variational autoencoders. *Advances in neural information processing systems*, 31, 2018.

## Bibliography II

- [6] Anthony L Caterini, Arnaud Doucet, and Dino Sejdinovic. Hamiltonian variational auto-encoder. In *Advances in Neural Information Processing Systems*, pages 8167–8177, 2018.
- [7] Clément Chadebec, Elina Thibeau-Sutre, Ninon Burgos, and Stéphanie Allassonnière. Data augmentation in high dimensional low sample size setting using a geometry-based variational autoencoder. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.
- [8] Clément Chadebec, Louis J Vincent, and Stéphanie Allassonnière. Pythae: Unifying generative autoencoders in python—a benchmarking use case. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2022.
- [9] Xi Chen, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731*, 2016.
- [10] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.
- [11] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

## Bibliography III

- [12] Justin Domke and Daniel R Sheldon. Importance weighting and variational inference. *Advances in neural information processing systems*, 31, 2018.
- [13] Vincent Fortuin, Dmitry Baranchuk, Gunnar Rätsch, and Stephan Mandt. Gp-vae: Deep probabilistic time series imputation. In *International conference on artificial intelligence and statistics*, pages 1651–1661. PMLR, 2020.
- [14] Partha Ghosh, Mehdi SM Sajjadi, Antonio Vergari, Michael Black, and Bernhard Schölkopf. From variational to deterministic autoencoders. In *8th International Conference on Learning Representations, ICLR 2020*, 2020.
- [15] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.
- [16] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *arXiv:1312.6114 [cs, stat]*, 2014.
- [17] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

## Bibliography IV

- [18] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29, 2016.
- [19] Lars Maaløe, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. Auxiliary deep generative models. In *International conference on machine learning*, pages 1445–1453. PMLR, 2016.
- [20] Chris J Maddison, John Lawson, George Tucker, Nicolas Heess, Mohammad Norouzi, Andriy Mnih, Arnaud Doucet, and Yee Teh. Filtering variational objectives. *Advances in Neural Information Processing Systems*, 30, 2017.
- [21] Bo Pang, Tian Han, Erik Nijkamp, Song-Chun Zhu, and Ying Nian Wu. Learning latent space energy-based prior model. *Advances in Neural Information Processing Systems*, 33, 2020.
- [22] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *Advances in neural information processing systems*, 30, 2017.
- [23] Rajesh Ranganath, Dustin Tran, and David Blei. Hierarchical variational models. In *International conference on machine learning*, pages 324–333. PMLR, 2016.

# Bibliography V

- [24] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in Neural Information Processing Systems*, 2020.
- [25] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR, 2015.
- [26] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.
- [27] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.
- [28] Tim Salimans, Diederik Kingma, and Max Welling. Markov chain monte carlo and variational inference: Bridging the gap. In *International Conference on Machine Learning*, pages 1218–1226, 2015.
- [29] Daniel Schwalbe-Koda and Rafael Gómez-Bombarelli. Generative models for automatic chemical design. *Machine Learning Meets Quantum Physics*, pages 445–467, 2020.

# Bibliography VI

- [30] Achille Thin, Nikita Kotelevskii, Arnaud Doucet, Alain Durmus, Eric Moulines, and Maxim Panov. Monte carlo variational auto-encoders. In *International Conference on Machine Learning*, pages 10247–10257. PMLR, 2021.
- [31] Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. *Advances in neural information processing systems*, 33:19667–19679, 2020.
- [32] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.