# Generative models for images

Bruno Galerne
**bruno.galerne@univ-orleans.fr**

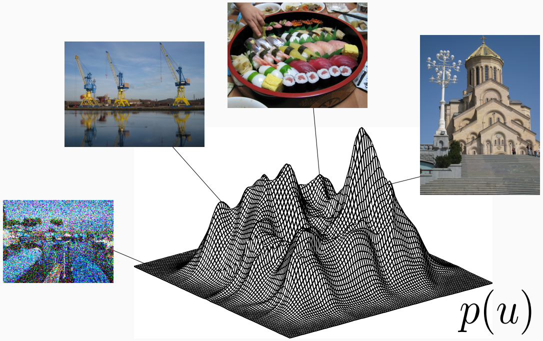
Summer school **Deep learning and applications**
IRMA, Université de Strasbourg
Monday August 29, 2022

Institut Denis Poisson
Université d'Orléans, Université de Tours, CNRS

## Introduction on generative models

## Generative models

1. Model and/or learn a distribution $p(u)$ on the space of images.



(source: Charles Deledalle)

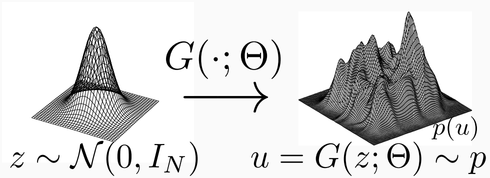
The images may represent:

- different instances of the same texture image.
- all images naturally described by a dataset of images.

2. Generate samples from this distribution.

## Generative models

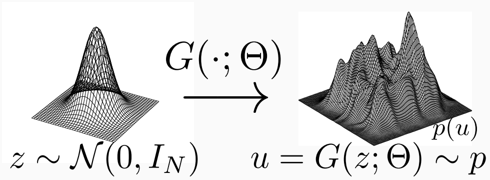
1. Model and/or learn a distribution $p(u)$ on the space of images.
2. Generate samples from this distribution.



- $z$ is a generic source of randomness, often called the latent variable.
- If $G(\cdot;\Theta)$ is known, then $p = G(\cdot;\Theta)_{\#}\mathcal{N}(0, I_n)$ is the push-forward of the latent distribution.

## Generative models

1. Model and/or learn a distribution $p(u)$ on the space of images.
2. Generate samples from this distribution.



- $z$ is a generic source of randomness, often called the latent variable.
- If $G(\cdot;\Theta)$ is known, then $p = G(\cdot;\Theta)_{\#}\mathcal{N}(0, I_n)$ is the push-forward of the latent distribution.

The generator $G(\cdot;\Theta)$ can be:

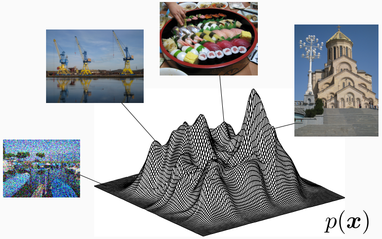
- A deterministic function (e.g. convolution operator),
- A neural network with learned parameter,
- An iterative optimization algorithm (gradient descent,...),
- A stochastic sampling algorithm (e.g. MCMC, Langevin diffusion,...).

## Image generation: Gaussian model

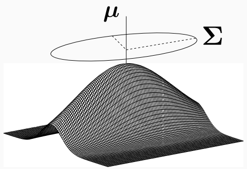
- Consider a **Gaussian model** for the distribution of images $\boldsymbol{x}$ with $d$ pixels:

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}|}} \exp\left[-(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}-\boldsymbol{\mu})\right]$$

  - $\boldsymbol{\mu}$: mean image,
  - $\boldsymbol{\Sigma}$: covariance matrix of images.



$\approx$



Gaussian prior $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$

(source: Charles Deledalle)

## Image generation: Gaussian model

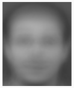
- Take a training dataset $\mathcal{D}$ of images:

$$\mathcal{D} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$$

$$= \Big\{ \quad , \quad , \quad , \quad , \quad , \quad , \quad \dots \Big\}_{\times N}$$

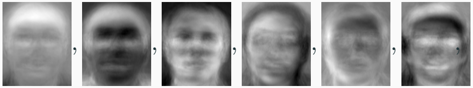
- Estimate the mean

$$\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_i \boldsymbol{x}_i =$$



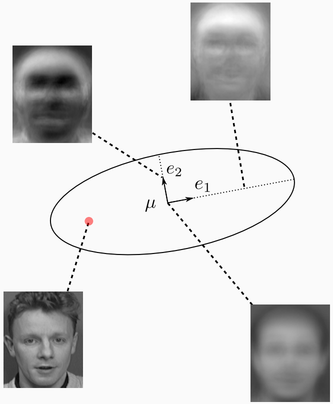
- Estimate the covariance matrix: $\hat{\boldsymbol{\Sigma}} = \frac{1}{N}\sum_i (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}})^T = \hat{\boldsymbol{E}}\hat{\boldsymbol{\Lambda}}\hat{\boldsymbol{E}}^T$

$$\hat{\boldsymbol{E}} = \underbrace{\Big\{ \quad , \quad , \quad , \quad , \quad , \quad , \quad \dots \Big\}_{\times N}}_{\text{eigenvectors of } \hat{\boldsymbol{\Sigma}}\text{, \textit{i.e.}, main variation axis}}$$



## Image generation: Gaussian model

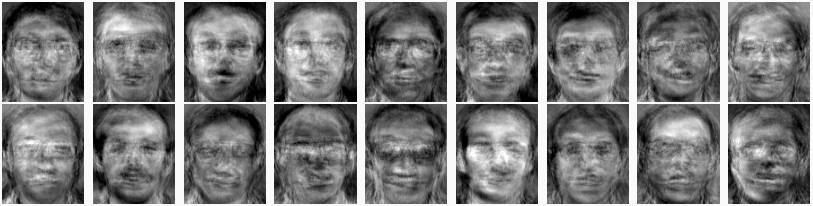
You now have learned a **generative model**:



## Image generation: Gaussian model

**How to generate samples from $\mathcal{N}(\hat{\mu}, \hat{\Sigma})$?**

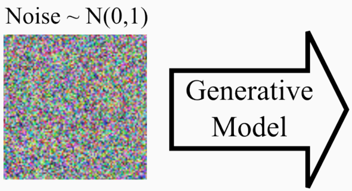
$$\begin{cases} z & \sim \mathcal{N}(0, \boldsymbol{I}_n) \quad \leftarrow \text{Generate random latent variable} \\ x & = \hat{\mu} + \hat{\boldsymbol{E}}\hat{\boldsymbol{\Lambda}}^{1/2} z \end{cases}$$



**The model does not generate realistic faces.**

- The Gaussian distribution assumption is too simplistic.
- Each generated image is just a linear random combination of the eigenvectors.
- The generator corresponds to a linear neural network (without non-linearities).

## Image generation: Gaussian model



- Deep generative modeling consists of learning non-linear generative models to reproduce complex data such as realistic images.
- It relies on deep neural networks and several solutions have been proposed since the “Deep learning revolution" (2012).

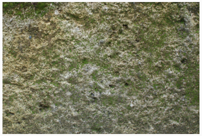
## Generative models: Examples

**Texture synthesis with a stationary Gaussian model:** (Galerne et al., 2011)

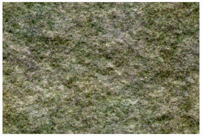
- Data: A single texture image $\boldsymbol{h}$.
- Inferred distribution: $p$ is the Gaussian distribution with similar mean and covariance statistics.
- $z$ is a Gaussian white noise image (each pixel is iid with standard normal distribution).
- $G$ is a convolution operator with know parameters $\Theta$.

Data

Generated images



Spot $\boldsymbol{h}$



$G(z_1; \Theta)$

$G(z_2; \Theta)$

$G(z_3; \Theta)$

## Generative models: Examples

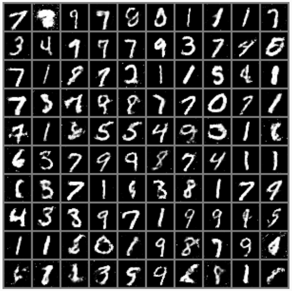
**Generative Aversarial Networks:** (Goodfellow et al., 2014)

- Data: A database of images.
- Inferred distribution: Not explicit, push-forward measure given by generator.
- $z$ is a Gaussian array in a latent space.
- $G(\cdot;\Theta)$ is a (convolutional) neural network with parameters $\Theta$ learned using an adversarial discriminator network $D(\cdot;\Theta_D)$.

Data

MNIST: handwritten digits

Generated images



Fake images (100 epochs)

## Generative models: Examples

**Generative Aversarial Networks:** Style GAN (Karras et al., 2019)

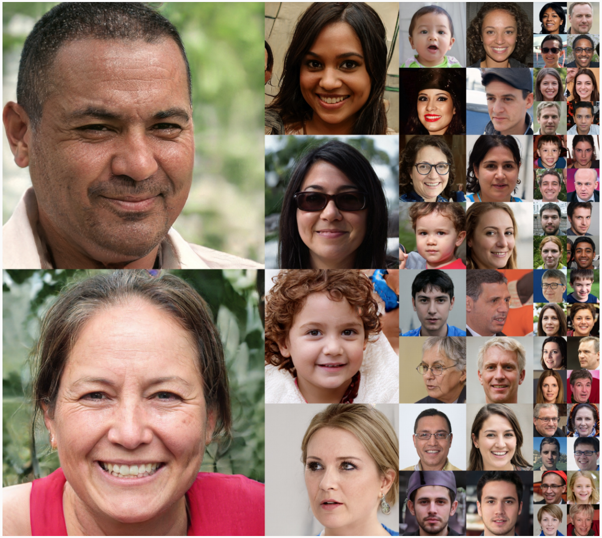
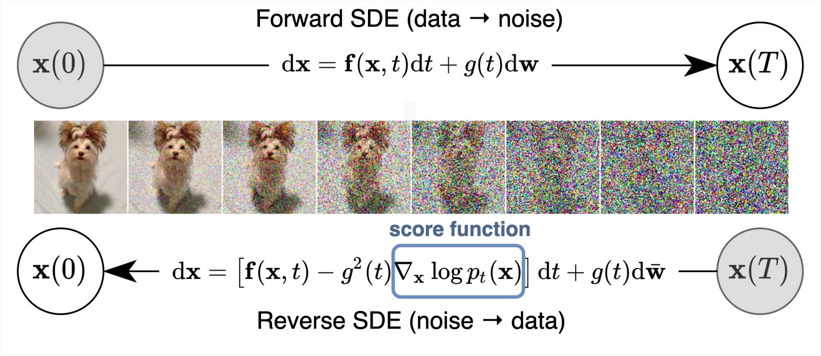


Image size:
1024$\times$ 1024 px
(source: Karras et al.)

## Denoising diffusion probabilistic models

- Learn to revert a degradation process: Add more and more noise to an image.
- First similar model (Sohl-Dickstein et al., 2015)



Forward SDE (data → noise)
$\mathbf{x}(0)$
$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$
$\mathbf{x}(T)$
score function
$\mathbf{x}(0)$
$\mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g^2(t)\boxed{\nabla_{\mathbf{x}} \log p_t(\mathbf{x})}\right] \mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}$
$\mathbf{x}(T)$
Reverse SDE (noise → data)

(source: Yang Song)

- Probably the most promising framework these days... but things change very quickly in this field!

## Diffusion models

(Ho et al., 2020): “Denoising Diffusion Probabilistic Models": One of the first paper producing images with reasonable resolution.

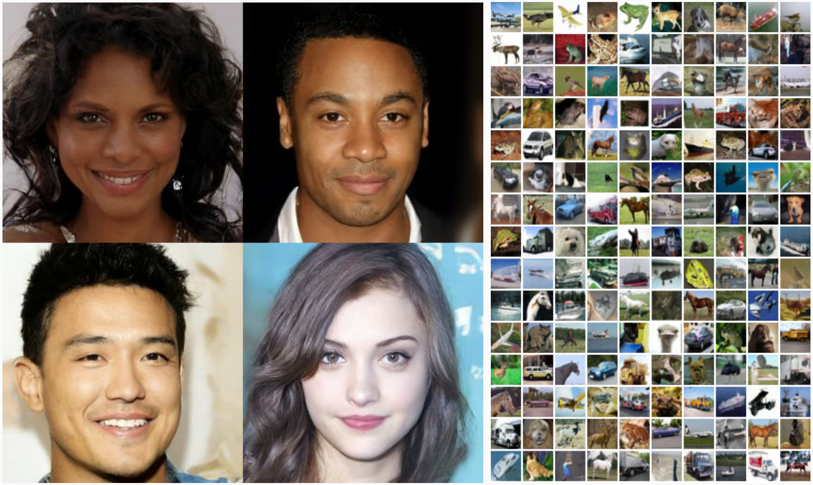


Figure 1: Generated samples on CelebA-HQ 256 $\times$ 256 (left) and unconditional CIFAR10 (right)

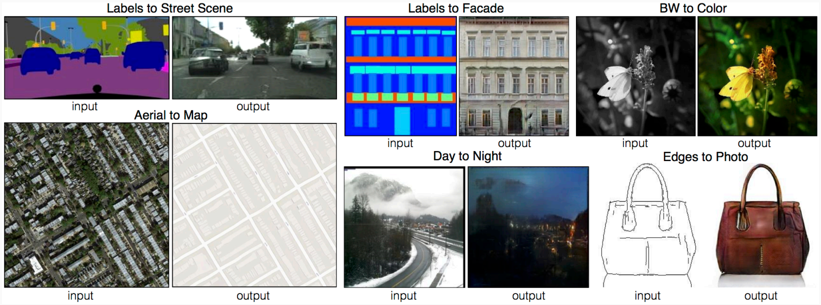
## Generative models: Motivations

Why generative models are interesting ?

- Generating realistic images is important by itself for entertainment industry (visual effects, video games, augmented reality...)
- Good image model leads to good image processing: Generative models can be used as a parametric space for solving inverse problems. Example: Inpainting of a portrait image.
- Also generative models leads to non trivial image manipulation using **conditional generative models**.

## Conditional generative models: Example

**Pix2pix: Image-to-Image Translation with Conditional Adversarial Nets** (Isola et al., 2017)



- GAN conditioned on input image.
- Generator: U-net architecture
- Discriminator: Patch discriminator applied to each patch
- Opens the way for new creative tools

(source: Isola et al.)

## Conditional generative models: Example

Latest trends using **diffusion models**: Text to image generation

- DALL·E 1 & 2: CreatingImages from Text (Open AI, January 2021 and April 2022)
- Imagen, Google research (May 2022)

DALL·E 2 (Open AI)
Input: An astronaut riding a horse in a photorealistic style

Imagen (Google)
Input: A dog looking curiously in the mirror, seeing a cat.

## Generative models for images: Plan of the course

**Part I: Introduction**

**Part II: Variational AutoEncoders (VAE)**

1. Autoencoders
2. Deep latent variable models
3. Evidence lowerbound (ELBO) for training VAEs

**Part III: Generative Adversarial Networks (GAN)**

1. Introduction to GAN
2. Adversarial training and its issues
3. Wasserstein GAN (with recap on optimal transport)
4. Conditional GAN

**Part VI: Examples of applications of generative models in imaging science**

(depending on time)

## Variational autoencoders (VAE)

## Variational autoencoders (VAE)

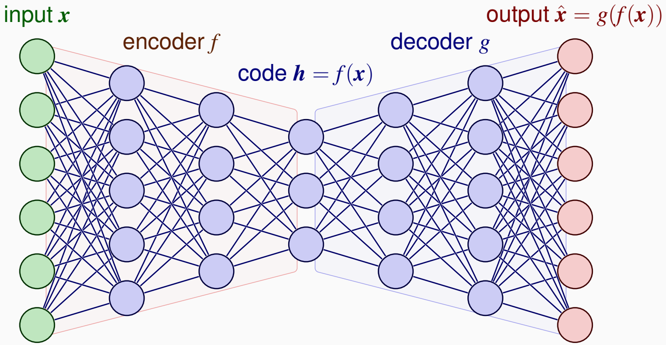
**Main references:**

1. Original paper: (Kingma and Welling, 2014): “Auto-Encoding Variational Bayes"
2. Short book by the same authors: (Kingma and Welling, 2019): “An Introduction to Variational Autoencoders". Freely available on ArXiv.
3. Recent book: (Tomczak, 2022): “Deep Generative Modeling" with practice sessions in the official GitHub repository.

## Autoencoders

But first what is an **autoencoder**? “An **autoencoder** is a neural network that is trained to attempt to copy its input to its output." (Goodfellow et al., 2016)

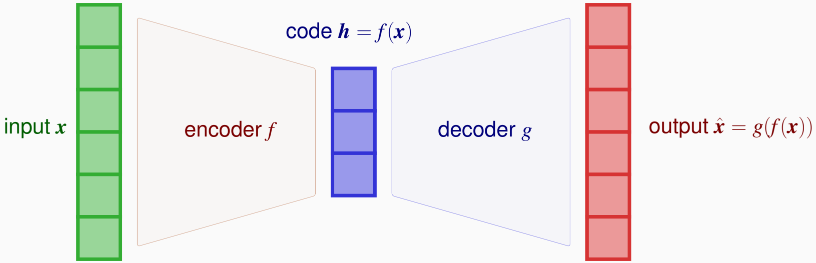
The network has a bottleneck hidden layer of lower dimension than the data.


input $\boldsymbol{x}$
encoder $f$
code $\boldsymbol{h} = f(\boldsymbol{x})$
decoder $g$
output $\hat{\boldsymbol{x}} = g(f(\boldsymbol{x}))$

## Autoencoders

But first what is an **autoencoder**? “An **autoencoder** is a neural network that is trained to attempt to copy its input to its output." (Goodfellow et al., 2016)

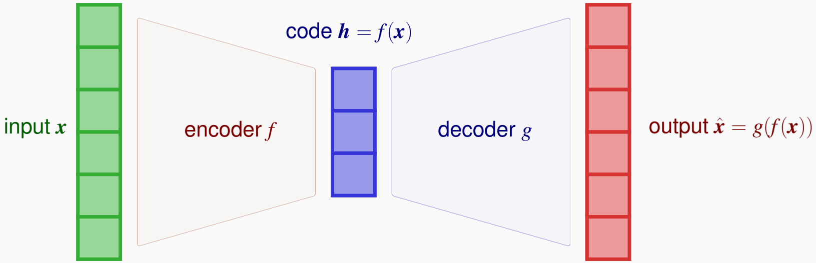
The network has a bottleneck hidden layer of lower dimension than the data.


code $\boldsymbol{h} = f(\boldsymbol{x})$
input $\boldsymbol{x}$
encoder $f$
decoder $g$
output $\hat{\boldsymbol{x}} = g(f(\boldsymbol{x}))$

## Autoencoders

But first what is an **autoencoder**? “An **autoencoder** is a neural network that is trained to attempt to copy its input to its output." (Goodfellow et al., 2016)

The network has a bottleneck hidden layer of lower dimension than the data.

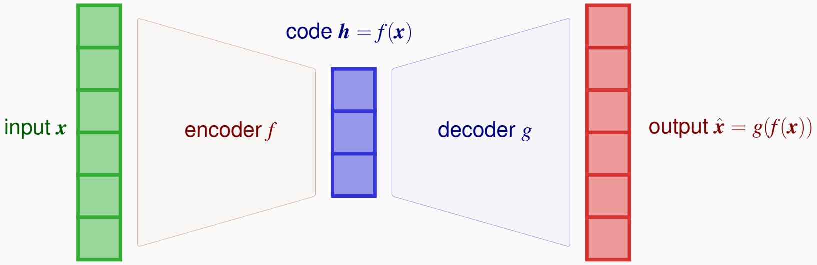


code $\boldsymbol{h} = f(\boldsymbol{x})$
input $\boldsymbol{x}$
encoder $f$
decoder $g$
output $\hat{\boldsymbol{x}} = g(f(\boldsymbol{x}))$

The network is trained by minizing a **reconstruction error** over the dataset $\mathcal{D} = \{\boldsymbol{x}^{(i)},\ i = 1, \dots, N\} \subset \mathbb{R}^d$

$$\text{MSE} = \frac{1}{N} \sum_{x \in \mathcal{D}} \|g(f(\boldsymbol{x})) - \boldsymbol{x}\|^2.$$

## Autoencoders


code $\boldsymbol{h} = f(\boldsymbol{x})$
input $\boldsymbol{x}$
encoder $f$
decoder $g$
output $\hat{\boldsymbol{x}} = g(f(\boldsymbol{x}))$

- Motivation: The encoder output $\boldsymbol{h} = f(\boldsymbol{x}) \in \mathbb{R}^k$ should produce an adapted compact representation of the sample $\boldsymbol{x}$ among the dataset $\mathcal{D}$.
- If both $f$ and $g$ are linear, the best solution will output the PCA projection using the first $k$ components.
- One hopes to learn the most salient features of the distribution.
- If $f$ and $g$ have a lot of capacity, then trivial code can be learnt by storing the dataset $\mathcal{D} = \{\boldsymbol{x}^{(i)},\ i = 1, \dots, N\}$:

$$f(\boldsymbol{x}^{(i)}) = i \quad \text{and} \quad g(i) = \boldsymbol{x}^{(i)}$$

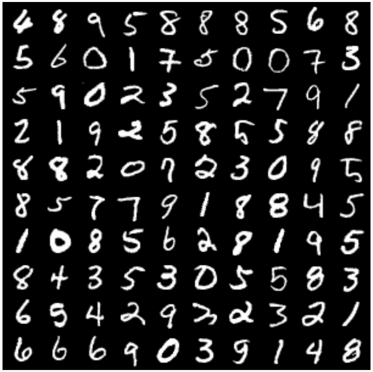
- Trade-off between the parameters of $f$ and $g$, dimensions $d > k$ etc.

## Autoencoders

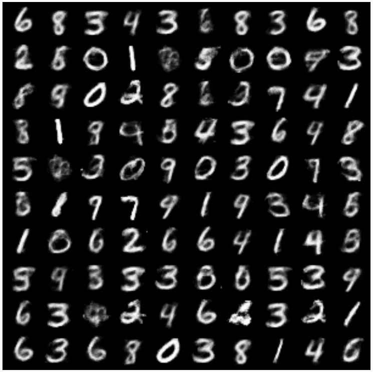
**Numerical illustration** with a subset of MNIST (1000 images only):

- Encoders and decoders are MLP trained for 1000 epochs.
- The **code dimension is $k = 2$ for visualization** (higher $k$ values would give better results).

Input test images:



Output:

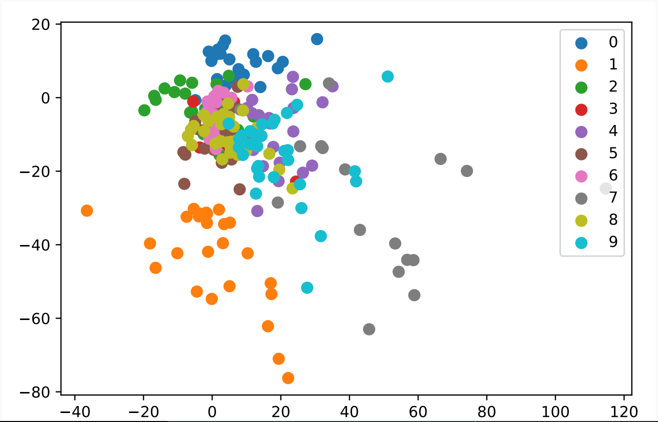


## Autoencoders

**Numerical illustration** with a subset of MNIST (1000 images only):

- Encoders and decoders are MLP trained for 1000 epochs.
- The **code dimension is $k = 2$ for visualization** (higher $k$ values would give better results).

2D latent code of 256 test images:



## Deep latent variable models

- In terms of architecture, **variational autoencoders (VAE)** are similar to autoencoders.
- The difference lies in the modeling and training of the network: VAEs learn (non linear) **deep latent variable models**.

What is a **latent variable model?** Back to probabilistic modeling...

- We are given an input dataset
$$\mathcal{D} = \{\boldsymbol{x}^{(i)},\ i = 1, \dots, N\} \subset \mathbb{R}^d$$
- We assume that the dataset $\mathcal{D}$ consists of distinct, independent measurements from the same unknown underlying process, whose true (probability) distribution $P^*$ is unknown.

**Remark: Identification of distribution and density**

- $p^* : \mathbb{R}^d \to \mathbb{R}_+$ will refer to the density with respect to (wrt) the Lebesgue measure of the unknown distribution $P^*$.
- Depending on context it can also be a discrete distribution (e.g. binarized images in $\{0, 1\}^d$)... Be careful!
- That said $P^*$ will be identified with $p^*(\boldsymbol{x})$ from now on.

## Deep latent variable models

**Framework:**

- We are given an input dataset $\mathcal{D} = \{\boldsymbol{x}^{(i)},\ i = 1, \dots, N\} \subset \mathbb{R}^d$ with iid samples from an unknown distribution $p^*(\boldsymbol{x})$.

**Probabilistic modeling:**

- Propose a **parametric model** $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ with parameters $\boldsymbol{\theta}$
- Learn good parameters $\boldsymbol{\theta}$ so that $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is close to $p^*(\boldsymbol{x})$: This is generally done by maximizing the dataset log-likelihood:

$$\max_{\theta} \log p_{\boldsymbol{\theta}}(\mathcal{D}) \quad \text{where} \quad \log p_{\boldsymbol{\theta}}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}).$$

- Maximazing the likelihood can be achieved by **minibatch stochastic gradient descent (SGD)** (on $-\log p_{\boldsymbol{\theta}}(\mathcal{D})$) providing $\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is tractable: For a random minibatch $\mathcal{M} \subset \mathcal{D}$,

$$\frac{1}{|\mathcal{M}|} \sum_{\boldsymbol{x} \in \mathcal{M}} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \quad \text{is an unbiased estimator of} \quad \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathcal{D}).$$

## Deep latent variable models

**Framework:**

- We are given an input dataset $\mathcal{D} = \{\boldsymbol{x}^{(i)},\ i = 1, \dots, N\} \subset \mathbb{R}^d$ with iid samples from an unknown distribution $p^*(\boldsymbol{x})$.

**Probabilistic modeling:**

- Propose a **parametric model** $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ with parameters $\boldsymbol{\theta}$
- Learn good parameters $\boldsymbol{\theta}$ so that $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is close to $p^*(\boldsymbol{x})$: This is generally done by maximizing the dataset log-likelihood:

$$\max_{\theta} \log p_{\boldsymbol{\theta}}(\mathcal{D}) \quad \text{where} \quad \log p_{\boldsymbol{\theta}}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}).$$

- Maximazing the likelihood can be achieved by **minibatch stochastic gradient descent (SGD)** (on $-\log p_{\boldsymbol{\theta}}(\mathcal{D})$) providing $\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is tractable: For a random minibatch $\mathcal{M} \subset \mathcal{D}$,

$$\frac{1}{|\mathcal{M}|} \sum_{\boldsymbol{x} \in \mathcal{M}} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \quad \text{is an unbiased estimator of} \quad \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathcal{D}).$$

- $\simeq$ means “is an unbiased estimator of"

## Deep latent variable models

**Framework:**

- We are given an input dataset $\mathcal{D} = \{\boldsymbol{x}^{(i)},\ i = 1, \dots, N\} \subset \mathbb{R}^d$ with iid samples from an unknown distribution $p^*(\boldsymbol{x})$.

**Probabilistic modeling:**

- Propose a **parametric model** $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ with parameters $\boldsymbol{\theta}$
- Learn good parameters $\boldsymbol{\theta}$ so that $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is close to $p^*(\boldsymbol{x})$: This is generally done by maximizing the dataset log-likelihood:

$$\max_{\theta} \log p_{\boldsymbol{\theta}}(\mathcal{D}) \quad \text{where} \quad \log p_{\boldsymbol{\theta}}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}).$$

- Maximazing the likelihood can be achieved by **minibatch stochastic gradient descent (SGD)** (on $-\log p_{\boldsymbol{\theta}}(\mathcal{D})$) providing $\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is tractable: For a random minibatch $\mathcal{M} \subset \mathcal{D}$,

$$\frac{1}{|\mathcal{M}|} \sum_{\boldsymbol{x} \in \mathcal{M}} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \simeq \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathcal{D}).$$

- $\simeq$ means “is an unbiased estimator of"

## Deep latent variable models

**Latent variables:**

- Latent variables are variables that are part of the model, but which we don't observe, and are therefore not part of the dataset. They are hidden factors.

  **Examples for portraits:** Age of the person, hair color,...
- One generally has a factorized joint distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}, z) = p_{\boldsymbol{\theta}}(z) p_{\boldsymbol{\theta}}(\boldsymbol{x}|z)$ that corresponds to a natural hierarchical generative process:
  1. Sample $z \sim p_{\boldsymbol{\theta}}(z)$ (generate latent variables = hidden factors)
  2. Sample $\boldsymbol{x} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|z)$ (conditional generator given latent variables)

**Vocabulary for latent variable models:**

- $p_{\boldsymbol{\theta}}(\boldsymbol{x}, z)$: latent variable model
- $p_{\boldsymbol{\theta}}(z) = \int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\boldsymbol{x}, z) d\boldsymbol{x}$: *prior distribution* over $z$.
- $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int_{\mathbb{R}^k} p_{\boldsymbol{\theta}}(\boldsymbol{x}, z) dz$: *marginal distribution* or *model evidence*
- $p_{\boldsymbol{\theta}}(\boldsymbol{x}|z)$: *conditional distribution* of $\boldsymbol{x}$ given $z$
- $p_{\boldsymbol{\theta}}(z|\boldsymbol{x})$: *posterior distribution* of $z$ given $\boldsymbol{x}$

## Latent variable models: Example of Gaussian mixture models

- $z \sim p_{\boldsymbol{\theta}}(z)$ is some discrete variable with $K$ values with distribution $\boldsymbol{\pi} = (\pi_1, \dots, \pi_K) \in \mathbb{R}^K$:

$$p_{\boldsymbol{\theta}}(z = j) = \pi_j, \quad j = 1, \dots, K.$$

- For each $j \in \{1, \dots, K\}$ the conditional distributions $p_{\boldsymbol{\theta}}(\boldsymbol{x}|z = j) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ is Gaussian with mean $\boldsymbol{\mu}_j$ and covariance matrix $\boldsymbol{\Sigma}_j$.
- The model parameters are $\boldsymbol{\theta} = \{\boldsymbol{\pi}, (\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1), \dots, (\boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K)\}$.
- The marginal distribution is a **Gaussian mixture model**:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

- The parameters can be learned from data using an EM (Expectation-Maximization) algorithm.
- Interest of latent models: Rich and flexible marginal distribution with only simple intermediate distributions.

## Deep latent variable models

**Deep latent variable model:** A latent variable model $p_{\boldsymbol{\theta}}(\boldsymbol{x}, z)$ is called **deep** when the parameters of the distribution are encoded with a (deep) neural network.

- **Example:** Given $z \sim p_{\boldsymbol{\theta}}(z)$, some neural network $f$ outputs $f(z) = (\boldsymbol{\mu}(z), \boldsymbol{\Sigma}(z))$ and one sets $p_{\boldsymbol{\theta}}(\boldsymbol{x}|z) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}(z), \boldsymbol{\Sigma}(z))$. This generalizes Gaussian mixture models with an infinite number of Gaussians, but also imposes regularity between the parameters since the neural network $f$ is continuous.

## Deep latent variable models

**Deep latent variable model:** A latent variable model $p_{\boldsymbol{\theta}}(\boldsymbol{x}, z)$ is called **deep** when the parameters of the distribution are encoded with a (deep) neural network.

- **Example:** Given $z \sim p_{\boldsymbol{\theta}}(z)$, some neural network $f$ outputs $f(z) = (\boldsymbol{\mu}(z), \boldsymbol{\Sigma}(z))$ and one sets $p_{\boldsymbol{\theta}}(\boldsymbol{x}|z) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}(z), \boldsymbol{\Sigma}(z))$. This generalizes Gaussian mixture models with an infinite number of Gaussians, but also imposes regularity between the parameters since the neural network $f$ is continuous.

**Intractability of marginal distribution:** In such a setting, computing the marginal

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int_{\mathbb{R}^k} p_{\boldsymbol{\theta}}(\boldsymbol{x}, z) dz = \int_{\mathbb{R}^k} p_{\boldsymbol{\theta}}(\boldsymbol{x}|z) p_{\boldsymbol{\theta}}(z) dz$$

is intractable and thus we cannot compute its value nor its gradient wrt $\boldsymbol{\theta}$ for maximum log-likelihood estimation.

**Intractability of inference:** Inference refers to sampling/recovering the latent variable $z$ of a given sample $\boldsymbol{x}$, that is sampling the posterior $p_{\boldsymbol{\theta}}(z|\boldsymbol{x})$. This is also generally intractable since $p_{\boldsymbol{\theta}}(z|\boldsymbol{x}) = \dfrac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, z)}{p_{\boldsymbol{\theta}}(\boldsymbol{x})}$.

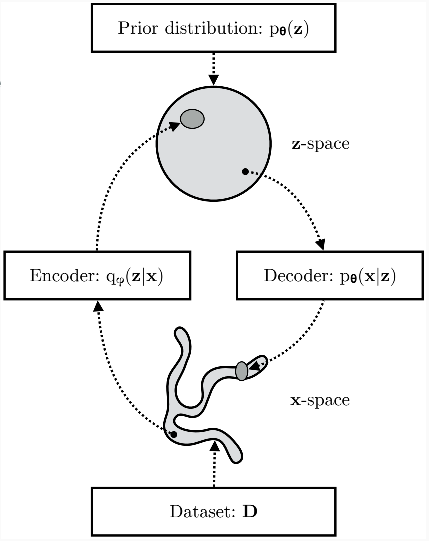
## Variational autoencoders (VAE)

(Kingma and Welling, 2019): “The framework of **variational autoencoders (VAEs)** provides a computationally efficient way for optimizing deep latent variable models jointly with a corresponding inference model using SGD."

## Variational autoencoders (VAE)

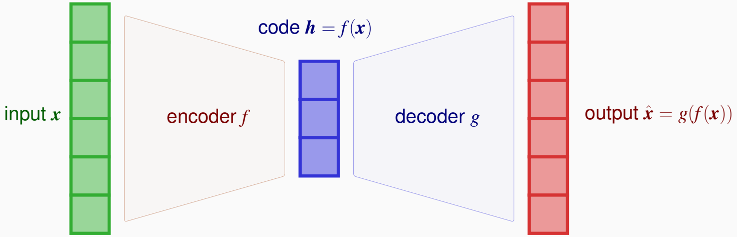
From (Kingma and Welling, 2019, p .20):

- A VAE learns stochastic mappings between an observed $x$-space, whose empirical distribution is typically complicated, and a latent $z$-space, whose distribution can be relatively simple.
- The generative model learns a joint distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x},z)$ factorized as $p_{\boldsymbol{\theta}}(\boldsymbol{x},z) = p_{\boldsymbol{\theta}}(z)p_{\boldsymbol{\theta}}(\boldsymbol{x}|z)$, with a prior distribution over latent space $p_{\boldsymbol{\theta}}(z)$, and a stochastic decoder $p_{\boldsymbol{\theta}}(\boldsymbol{x}|z)$.
- The stochastic encoder $q_{\varphi}(z|\boldsymbol{x})$, also called inference model, approximates the true but intractable posterior $p_{\boldsymbol{\theta}}(z|\boldsymbol{x})$ of the generative model.

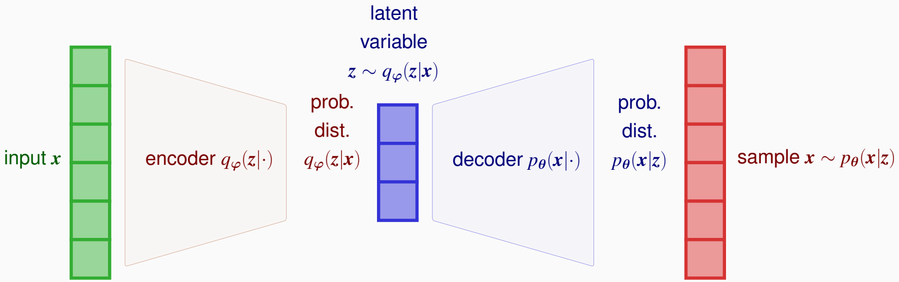

Prior distribution: $p_{\boldsymbol{\theta}}(\mathbf{z})$
$\mathbf{z}$-space
Encoder: $q_{\varphi}(\mathbf{z}|\mathbf{x})$
Decoder: $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$
$\mathbf{x}$-space
Dataset: $\mathbf{D}$

## Variational autoencoders (VAE)

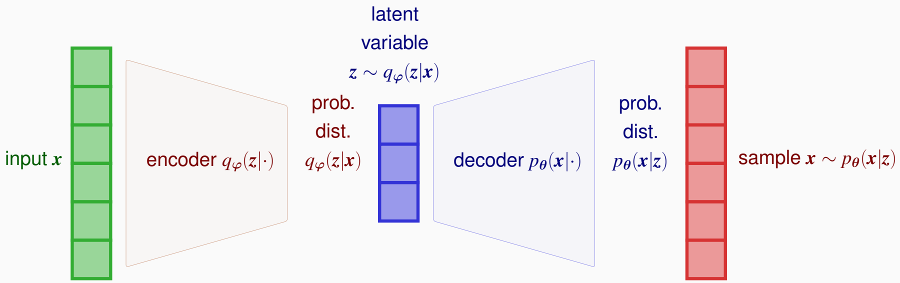
**Autoencoders:**


code $\boldsymbol{h} = f(\boldsymbol{x})$
input $\boldsymbol{x}$
encoder $f$
decoder $g$
output $\hat{\boldsymbol{x}} = g(f(\boldsymbol{x}))$

**Variational autoencoders:** NN outputs encode probability distributions


latent variable $\boldsymbol{z} \sim q_\varphi(\boldsymbol{z}|\boldsymbol{x})$
input $\boldsymbol{x}$
encoder $q_\varphi(\boldsymbol{z}|\cdot)$
prob. dist. $q_\varphi(\boldsymbol{z}|\boldsymbol{x})$
decoder $p_\theta(\boldsymbol{x}|\cdot)$
prob. dist. $p_\theta(\boldsymbol{x}|\boldsymbol{z})$
sample $\boldsymbol{x} \sim p_\theta(\boldsymbol{x}|\boldsymbol{z})$

## Variational autoencoders (VAE)

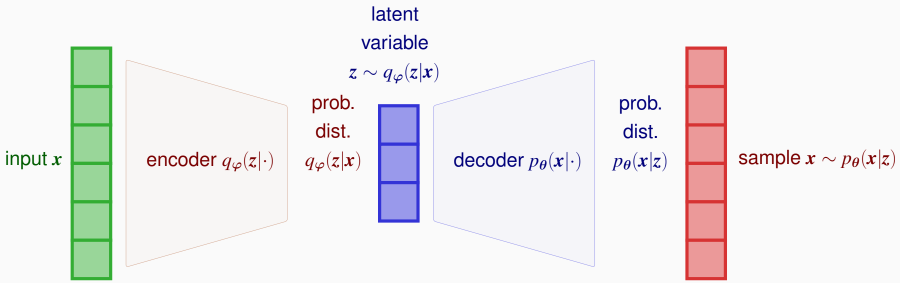

latent variable $z \sim q_\varphi(z|\boldsymbol{x})$
input $\boldsymbol{x}$
encoder $q_\varphi(z|\cdot)$
prob. dist. $q_\varphi(z|\boldsymbol{x})$
decoder $p_\theta(\boldsymbol{x}|\cdot)$
prob. dist. $p_\theta(\boldsymbol{x}|z)$
sample $\boldsymbol{x} \sim p_\theta(\boldsymbol{x}|z)$

**Stochastic encoder:**

- The encoder $q_\varphi(z|\boldsymbol{x})$ is understood as a parametric approximation of the true posterior $p_{\boldsymbol{\theta}}(z|\boldsymbol{x})$.
- To achieve that the parameters $\varphi$ must be trained along with the parameters $\boldsymbol{\theta}$ of the generative model.
- **Example of stochastic encoder:** A neural network outputs two vectors $(\boldsymbol{\mu}(\boldsymbol{x}), \log \boldsymbol{\sigma}(\boldsymbol{x})) = \mathrm{NN}_\varphi(\boldsymbol{x})$ and one sets:

$$q_\varphi(z|\boldsymbol{x}) = \mathcal{N}(z; \boldsymbol{\mu}(\boldsymbol{x}), \mathrm{diag}(\boldsymbol{\sigma}^2(\boldsymbol{x}))).$$

## Variational autoencoders (VAE)


latent variable
$z \sim q_\varphi(z|\boldsymbol{x})$
input $\boldsymbol{x}$
encoder $q_\varphi(z|\cdot)$
prob. dist. $q_\varphi(z|\boldsymbol{x})$
decoder $p_\theta(\boldsymbol{x}|\cdot)$
prob. dist. $p_\theta(\boldsymbol{x}|z)$
sample $\boldsymbol{x} \sim p_\theta(\boldsymbol{x}|z)$

**Next challenge: Learning!**

- How can we learn the parameters $\boldsymbol{\theta}$ (and $\varphi$) such that maximizes the log-likelihood of

$$\log p_{\boldsymbol{\theta}}(\mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum_{x\in\mathcal{D}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \quad \text{where} \quad p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int_{\mathbb{R}^k} p_{\boldsymbol{\theta}}(\boldsymbol{x}, z)dz$$

is the (untractable) *marginal distribution* (or *model evidence*)?
- In fact we will only maximize a lower bound of each $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ called the **evidence lower bound** (ELBO).

## Evidence lower bound (ELBO)

**Evidence lower bound (ELBO):**

Let $q_\varphi(z|x)$ be any parametric family of distributions that are positive (i.e. charge every non negligible sets like non degenerate Gaussian distributions).

For all $x \in \mathbb{R}^d$,

$$\begin{aligned}
\log p_\theta(x) &= \mathbb{E}_{z\sim q_\varphi(z|x)}\left[\log p_\theta(x)\right] \\
&= \mathbb{E}_{z\sim q_\varphi(z|x)}\left[\log\left[\frac{p_\theta(x,z)}{p_\theta(z|x)}\right]\right] \\
&= \mathbb{E}_{z\sim q_\varphi(z|x)}\left[\log\left[\frac{p_\theta(x,z)}{p_\theta(z|x)}\,\frac{q_\varphi(z|x)}{q_\varphi(z|x)}\right]\right] \\
&= \mathbb{E}_{z\sim q_\varphi(z|x)}\left[\log\left[\frac{p_\theta(x,z)}{q_\varphi(z|x)}\right]\right] + \underbrace{\mathbb{E}_{z\sim q_\varphi(z|x)}\left[\log\left[\frac{q_\varphi(z|x)}{p_\theta(z|x)}\right]\right]}_{D_{\mathrm{KL}}(q_\varphi(z|x)\|p_\theta(z|x))\geq 0} \\
&\geq \mathbb{E}_{z\sim q_\varphi(z|x)}\left[\log\left[\frac{p_\theta(x,z)}{q_\varphi(z|x)}\right]\right] := \mathcal{L}_{\theta,\varphi}(x) \quad \text{(ELBO)}
\end{aligned}$$

## Kullback–Leibler divergence

**General case:** Given two distribution $P$ and $Q$ on some measurable space $\mathcal{X}$, one defines the **Kullback–Leibler divergence** of $P$ wrt $Q$ by

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{\mathcal{X}} \log\left(\frac{P(dx)}{Q(dx)}\right) P(dx)$$

if $P$ is absolutely continuous wrt $Q$, and $D_{\mathrm{KL}}(P \parallel Q) = +\infty$ otherwise, where $\frac{P(dx)}{Q(dx)}$ the Radon–Nikodym derivative of $P$ wrt $Q$.

**Case with density wrt the Lebesgue measure:** If $\mathcal{X} = \mathbb{R}^d$ and $P$ and $Q$ have densities $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$ then

$$D_{\mathrm{KL}}(p(\boldsymbol{x}) \parallel q(\boldsymbol{x})) = \int_{\mathbb{R}^d} \log\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right) p(\boldsymbol{x}) d\boldsymbol{x} = \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}\left[\log\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right)\right].$$

## Kullback–Leibler divergence

**General case:** Given two distribution $P$ and $Q$ on some measurable space $\mathcal{X}$, one defines the **Kullback–Leibler divergence** of $P$ wrt $Q$ by

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{\mathcal{X}} \log\left(\frac{P(dx)}{Q(dx)}\right) P(dx)$$

if $P$ is absolutely continuous wrt $Q$, and $D_{\mathrm{KL}}(P \parallel Q) = +\infty$ otherwise, where $\frac{P(dx)}{Q(dx)}$ the Radon–Nikodym derivative of $P$ wrt $Q$.

**Case with density wrt the Lebesgue measure:** If $\mathcal{X} = \mathbb{R}^d$ and $P$ and $Q$ have densities $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$ then

$$D_{\mathrm{KL}}(p(\boldsymbol{x}) \parallel q(\boldsymbol{x})) = \int_{\mathbb{R}^d} \log\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right) p(\boldsymbol{x}) d\boldsymbol{x} = \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}\left[\log\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right)\right].$$

**Main properties:**

- $D_{\mathrm{KL}}(P \parallel Q) \geq 0$ and $D_{\mathrm{KL}}(P \parallel Q) = 0 \iff P = Q$
- $D_{\mathrm{KL}}(P \parallel Q) \neq D_{\mathrm{KL}}(Q \parallel P)$
- $\lim_{n\to+\infty} D_{\mathrm{KL}}(P_n \parallel Q) = 0$ implies convergence in distribution (and even in total variation).
- $D_{\mathrm{KL}}(P \parallel Q)$ is convex in $(P, Q)$.

## Evidence lower bound (ELBO)

**Evidence lower bound (ELBO):**

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\boldsymbol{x}) = \mathbb{E}_{z\sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},z)}{q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})}\right]\right] = p_{\boldsymbol{\theta}}(\boldsymbol{x}) - D_{\mathrm{KL}}(q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(z|\boldsymbol{x})) \leq \log p_{\boldsymbol{\theta}}(\boldsymbol{x}).$$

- The KL-divergence $D_{\mathrm{KL}}(q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(z|\boldsymbol{x}))$ gives the tightness of the lower bound: the better the approximation of the true posterior is the tighter is the lower bound.
- Main contribution of VAE (Kingma and Welling, 2014):
  **Use the ELBO $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\boldsymbol{x})$ as a training loss** for improving the log-likelihood.
- To use $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\boldsymbol{x})$ as a training loss using SGD we need to compute unbiased estimators of both

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\boldsymbol{x}) \quad \text{and} \quad \nabla_{\boldsymbol{\varphi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\boldsymbol{x}).$$

## Evidence lower bound (ELBO)

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

## Evidence lower bound (ELBO)

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\boldsymbol{x}) = \mathbb{E}_{z\sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},z)}{q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})}\right]\right]$$
$$= \mathbb{E}_{z\sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x},z)\right] - \mathbb{E}_{z\sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})}\left[\log q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})\right]$$

**Unbiased estimator for $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\boldsymbol{x})$:**

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\theta},\varphi}(\boldsymbol{x}) = \mathbb{E}_{z\sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})}\left[\nabla_{\boldsymbol{\theta}}\log p_{\boldsymbol{\theta}}(\boldsymbol{x},z)\right] \simeq \nabla_{\boldsymbol{\theta}}\log p_{\boldsymbol{\theta}}(\boldsymbol{x},z^{(1)}) \quad \text{where} \quad z^{(1)}\sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}).$$

Recall that $p_{\boldsymbol{\theta}}(\boldsymbol{x},z) = p_{\boldsymbol{\theta}}(z)p_{\boldsymbol{\theta}}(\boldsymbol{x}|z)$ is a known parametric function (involving the stochastic decoder) that can be (automatically) differentiated wrt $\boldsymbol{\theta}$.

## Evidence lower bound (ELBO)

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

**Unbiased estimator for $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\boldsymbol{x})$:**

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\theta},\varphi}(\boldsymbol{x}) = \mathbb{E}_{z\sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})}\left[\nabla_{\boldsymbol{\theta}}\log p_{\boldsymbol{\theta}}(\boldsymbol{x},z)\right] \simeq \nabla_{\boldsymbol{\theta}}\log p_{\boldsymbol{\theta}}(\boldsymbol{x},z^{(1)}) \quad \text{where} \quad z^{(1)} \sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}).$$

Recall that $p_{\boldsymbol{\theta}}(\boldsymbol{x},z) = p_{\boldsymbol{\theta}}(z)p_{\boldsymbol{\theta}}(\boldsymbol{x}|z)$ is a known parametric function (involving the stochastic decoder) that can be (automatically) differentiated wrt $\boldsymbol{\theta}$.

**Unbiased estimator for $\nabla_{\boldsymbol{\varphi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\boldsymbol{x})$:**

- Not as straightforward since the ELBO expectation is taken with respect to $q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})$ that depends on $\boldsymbol{\varphi}$!

## Evidence lower bound (ELBO)

**Reparameterization trick:**

- Hypothesis: There is a fixed distribution $p(\varepsilon)$ and a deterministic function $g$ such that for any given $\boldsymbol{x}$ and $\boldsymbol{\varphi}$

$$\varepsilon \sim p(\varepsilon) \quad \Longrightarrow \quad z = g(\varepsilon, \boldsymbol{\varphi}, \boldsymbol{x}) \sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}).$$

- The function $g$ decouples the randomness source and the parameters for simulating the approximate posterior $q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})$.

**Example of Gaussian stochastic encoder:**

- $q_{\varphi}(z|\boldsymbol{x}) = \mathcal{N}(z; \boldsymbol{\mu}(\boldsymbol{x}), \operatorname{diag}(\boldsymbol{\sigma}^2(\boldsymbol{x})))$ with $(\boldsymbol{\mu}(\boldsymbol{x}), \log \boldsymbol{\sigma}(\boldsymbol{x})) = \mathrm{NN}_{\varphi}(\boldsymbol{x})$.
- With $p(\varepsilon) = \mathcal{N}(\varepsilon; \mathbf{0}, \boldsymbol{I})$ the standard Gaussian distribution:

$$z = \boldsymbol{\mu}(\boldsymbol{x}) + \boldsymbol{\sigma}(\boldsymbol{x}) \odot \varepsilon \sim \mathcal{N}(z; \boldsymbol{\mu}(\boldsymbol{x}), \operatorname{diag}(\boldsymbol{\sigma}^2(\boldsymbol{x}))).$$

## Evidence lower bound (ELBO)

**Reparameterization trick:**

- Hypothesis: There is a fixed distribution $p(\varepsilon)$ and a deterministic function $g$ such that for any given $\boldsymbol{x}$ and $\boldsymbol{\varphi}$

$$\varepsilon \sim p(\varepsilon) \quad \Longrightarrow \quad z = g(\varepsilon, \boldsymbol{\varphi}, \boldsymbol{x}) \sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}).$$

## Evidence lower bound (ELBO)

**Reparameterization trick:**

- Hypothesis: There is a fixed distribution $p(\varepsilon)$ and a deterministic function $g$ such that for any given $\boldsymbol{x}$ and $\boldsymbol{\varphi}$

$$\varepsilon \sim p(\varepsilon) \quad \Longrightarrow \quad z = g(\varepsilon, \boldsymbol{\varphi}, \boldsymbol{x}) \sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}).$$

**Change of variable in the ELBO:**

$$\begin{aligned}
\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\boldsymbol{x}) &= \mathbb{E}_{z\sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, z)\right] - \mathbb{E}_{z\sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})}\left[\log q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})\right] \\
&= \mathbb{E}_{\varepsilon\sim p(\varepsilon)}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, g(\varepsilon, \boldsymbol{\varphi}, \boldsymbol{x}))\right] - \mathbb{E}_{\varepsilon\sim p(\varepsilon)}\left[\log q_{\boldsymbol{\varphi}}(g(\varepsilon, \boldsymbol{\varphi}, \boldsymbol{x})|\boldsymbol{x})\right]
\end{aligned}$$

## Evidence lower bound (ELBO)

**Reparameterization trick:**

- Hypothesis: There is a fixed distribution $p(\varepsilon)$ and a deterministic function $g$ such that for any given $\boldsymbol{x}$ and $\boldsymbol{\varphi}$

$$\varepsilon \sim p(\varepsilon) \quad \Longrightarrow \quad z = g(\varepsilon, \boldsymbol{\varphi}, \boldsymbol{x}) \sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}).$$

**Change of variable in the ELBO:**

$$\begin{aligned}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\boldsymbol{x}) &= \mathbb{E}_{z\sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, z)\right] - \mathbb{E}_{z\sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})}\left[\log q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})\right] \\ &= \mathbb{E}_{\boldsymbol{\varepsilon}\sim p(\boldsymbol{\varepsilon})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, g(\varepsilon, \boldsymbol{\varphi}, \boldsymbol{x}))\right] - \mathbb{E}_{\boldsymbol{\varepsilon}\sim p(\boldsymbol{\varepsilon})}\left[\log q_{\boldsymbol{\varphi}}(g(\varepsilon, \boldsymbol{\varphi}, \boldsymbol{x})|\boldsymbol{x})\right]\end{aligned}$$

**Unbiased estimator for $\nabla_{\boldsymbol{\varphi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\boldsymbol{x})$:**

- Draw $\varepsilon^{(1)} \sim p(\varepsilon)$ and (automatically) differentiate wrt $\boldsymbol{\varphi}$ the expression

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, g(\varepsilon^{(1)}, \boldsymbol{\varphi}, \boldsymbol{x})) - \log q_{\boldsymbol{\varphi}}(g(\varepsilon^{(1)}, \boldsymbol{\varphi}, \boldsymbol{x})|\boldsymbol{x})$$

- Same for differentiating wrt $\boldsymbol{\theta}$.

## VAE training

**VAE Training algorithm:**

1. Draw a minibatch $\mathcal{M} = \{\boldsymbol{x}^{(i_1)}, \dots, \boldsymbol{x}^{(i_M)}\}$ of $M$ samples from $\mathcal{D} = \{\boldsymbol{x}^{(i)},\ i = 1, \dots, N\}$
2. Draw $M$ random $\varepsilon_m \sim p(\varepsilon)$, $m = 1, \dots, M$.
3. Compute $z^m = g(\varepsilon^{(m)}, \varphi, \boldsymbol{x}^{(i_m)}) \sim q_\varphi(z|\boldsymbol{x}^{(i_m)})$ using the encoder network parameters.
4. Apply the decoder network to each latent variable $z^m$ and return

$$\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\mathcal{M}) = \frac{1}{M}\sum_{m=1}^{M} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i_m)}, g(\varepsilon^{(m)}, \boldsymbol{\varphi}, \boldsymbol{x}^{(i_m)})) - \log q_{\boldsymbol{\varphi}}(g(\varepsilon^{(m)}, \boldsymbol{\varphi}, \boldsymbol{x}^{(i_m)})|\boldsymbol{x}^{(i_m)})$$

5. Compute $\nabla_{\boldsymbol{\theta}} \tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\mathcal{M})$ and $\nabla_{\boldsymbol{\varphi}} \tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\mathcal{M})$ by automatic differentiation and update the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\varphi}$ by an SGD step.

## VAE training

**VAE Training algorithm:**

1. Draw a minibatch $\mathcal{M} = \{\boldsymbol{x}^{(i_1)}, \dots, \boldsymbol{x}^{(i_M)}\}$ of $M$ samples from $\mathcal{D} = \{\boldsymbol{x}^{(i)},\ i = 1, \dots, N\}$
2. Draw $M$ random $\varepsilon_m \sim p(\varepsilon)$, $m = 1, \dots, M$.
3. Compute $z^m = g(\varepsilon^{(m)}, \varphi, \boldsymbol{x}^{(i_m)}) \sim q_\varphi(z|\boldsymbol{x}^{(i_m)})$ using the encoder network parameters.
4. Apply the decoder network to each latent variable $z^m$ and return

$$\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\mathcal{M}) = \frac{1}{M}\sum_{m=1}^{M} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i_m)}, g(\boldsymbol{\varepsilon}^{(m)}, \boldsymbol{\varphi}, \boldsymbol{x}^{(i_m)})) - \log q_{\boldsymbol{\varphi}}(g(\boldsymbol{\varepsilon}^{(m)}, \boldsymbol{\varphi}, \boldsymbol{x}^{(i_m)})|\boldsymbol{x}^{(i_m)})$$

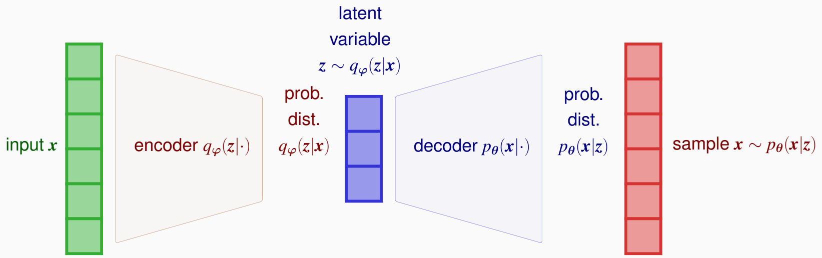
5. Compute $\nabla_{\boldsymbol{\theta}} \tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\mathcal{M})$ and $\nabla_{\boldsymbol{\varphi}} \tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\mathcal{M})$ by automatic differentiation and update the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\varphi}$ by an SGD step.

**Remark:** $\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\mathcal{M})$ is an unbiased estimator of the training loss

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

where we have double stochasticity from sampling the batch $\mathcal{M}$ and approximating each expectation with a single realization.

## VAE: Gaussian encoder and decoder


latent variable $z \sim q_\varphi(z|\boldsymbol{x})$
input $\boldsymbol{x}$
encoder $q_\varphi(z|\cdot)$
prob. dist. $q_\varphi(z|\boldsymbol{x})$
decoder $p_\theta(\boldsymbol{x}|\cdot)$
prob. dist. $p_\theta(\boldsymbol{x}|z)$
sample $\boldsymbol{x} \sim p_\theta(\boldsymbol{x}|z)$

**Example of Gaussian stochastic encoder and decoder:**

- **Gaussian stochastic encoder:** $q_\varphi(z|\boldsymbol{x}) = \mathcal{N}(z; \boldsymbol{\mu}(\boldsymbol{x}), \mathrm{diag}(\boldsymbol{\sigma}^2(\boldsymbol{x})))$ with $(\boldsymbol{\mu}(\boldsymbol{x}), \log \boldsymbol{\sigma}(\boldsymbol{x})) = \mathrm{NN}_\varphi(\boldsymbol{x})$.
- **Gaussian prior :** $p_{\boldsymbol{\theta}}(z) = \mathcal{N}(z; \mathbf{0}, \boldsymbol{I})$ the prior is fixed without parameter to learn.
- **Gaussian stochastic decoder:** $p_{\boldsymbol{\theta}}(\boldsymbol{x}|z) = \mathcal{N}(z; \boldsymbol{\mu}_{\mathrm{dec}}(z), s^2\boldsymbol{I})$ with $\boldsymbol{\mu}_{\mathrm{dec}}(z) = \mathrm{NN}_{\boldsymbol{\theta}}(z)$: Fixed isotropic Gaussian around a decoded mean $\boldsymbol{\mu}(z)$. The noise level $s > 0$ should be fixed according to the dataset range value.
- The architectures for $\mathrm{NN}_\varphi$ and $\mathrm{NN}_{\boldsymbol{\theta}}$ are generally chosen symmetric.

## VAE: Gaussian encoder and decoder

**Density of a Gaussian distribution:** For $\boldsymbol{x} \in \mathbb{R}^d$,

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}|}} \exp\left( -\frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}) \right)$$

$$\log \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{d}{2} \log(2\pi) - \frac{1}{2} \log(|\boldsymbol{\Sigma}|) - \frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})$$

**Expression of the ELBO loss:** With $z = g(\varepsilon, \boldsymbol{\varphi}, \boldsymbol{x}) = \boldsymbol{\mu}(\boldsymbol{x}) + \boldsymbol{\sigma}(\boldsymbol{x}) \odot \varepsilon$,

$$\begin{aligned}
\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\boldsymbol{x}) &= \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, z) - \log q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}) \\
&= \log p_{\boldsymbol{\theta}}(z) + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|z) - \log q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}) \\
&= -\frac{k}{2} \log(2\pi) - \frac{1}{2} \|z\|^2 \\
&\quad - \frac{d}{2} \log(2\pi s) - \frac{1}{2s^2} \|\boldsymbol{x} - \boldsymbol{\mu}_{\mathrm{dec}}(z)\|^2 \\
&\quad + \frac{k}{2} \log(2\pi) + \frac{1}{2} \sum_{j=1}^{k} \log \sigma_j(\boldsymbol{x}) + \frac{1}{2} (z - \boldsymbol{\mu}(\boldsymbol{x}))^2 \oslash \boldsymbol{\sigma}^2(\boldsymbol{x}) \\
&= \underbrace{-\frac{1}{2s^2} \|\boldsymbol{x} - \boldsymbol{\mu}_{\mathrm{dec}}(z)\|^2}_{\text{reconstruction error}} \underbrace{- \frac{1}{2} \|z\|^2 + \frac{1}{2} \sum_{j=1}^{k} \log \sigma_j(\boldsymbol{x}) + \frac{1}{2} (z - \boldsymbol{\mu}(\boldsymbol{x}))^2 \oslash \boldsymbol{\sigma}^2(\boldsymbol{x})}_{\text{latent code regularization}} + C
\end{aligned}$$

## VAE: ELBO and Kullback–Leibler divergence

The “latent code regularization" is better seen by **refactorizing the ELBO**:

$$\begin{aligned}
\mathcal{L}_{\theta,\varphi}(x) &= \mathbb{E}_{z\sim q_\varphi(z|x)}\left[\log\left[\frac{p_\theta(x,z)}{q_\varphi(z|x)}\right]\right]\\
&= \mathbb{E}_{z\sim q_\varphi(z|x)}\left[\log\left[\frac{p_\theta(z)p_\theta(x|z)}{q_\varphi(z|x)}\right]\right]\\
&= \mathbb{E}_{z\sim q_\varphi(z|x)}\left[\log p_\theta(x|z)\right] + \mathbb{E}_{z\sim q_\varphi(z|x)}\left[\log\left[\frac{p_\theta(z)}{q_\varphi(z|x)}\right]\right]\\
&= \underbrace{\mathbb{E}_{z\sim q_\varphi(z|x)}\left[\log p_\theta(x|z)\right]}_{\text{reconstruction error}} - \underbrace{D_{\mathrm{KL}}(q_\varphi(z|x)\parallel p_\theta(z))}_{\text{latent code regularization}}
\end{aligned}$$

- The latent code regularization enforces all the approximate posterior to be close to the prior.
- But to have a small reconstruction error, the support of the distributions $q_\varphi(z|x)$ have to be well-separated.
- This results in an encoder-decoder with well-spread latent code distribution.

## VAE: ELBO and Kullback–Leibler divergence

**Refactorizing the ELBO**:

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\boldsymbol{x}) = \underbrace{\mathbb{E}_{z\sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|z)\right]}_{\text{reconstruction error}} - \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(z))}_{\text{latent code regularization}}$$

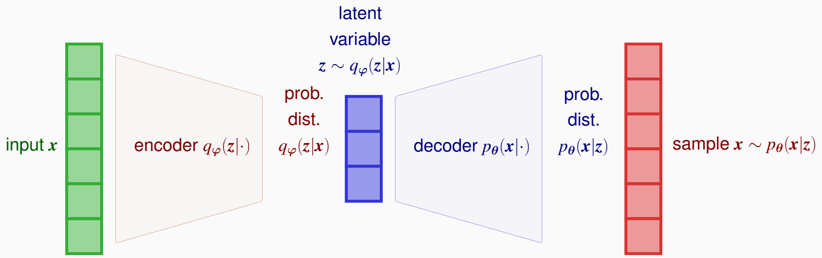
**Example of Gaussian stochastic encoder:**

- **Gaussian stochastic encoder:** $q_{\varphi}(z|\boldsymbol{x}) = \mathcal{N}(z; \boldsymbol{\mu}(\boldsymbol{x}), \mathrm{diag}(\boldsymbol{\sigma}^2(\boldsymbol{x})))$ with $(\boldsymbol{\mu}(\boldsymbol{x}), \log\boldsymbol{\sigma}(\boldsymbol{x})) = \mathrm{NN}_{\varphi}(\boldsymbol{x})$.
- **Gaussian prior :** $p_{\boldsymbol{\theta}}(z) = \mathcal{N}(z; \mathbf{0}, \boldsymbol{I})$ the prior is fixed without parameter to learn.
- Then one has a closed form formula for the KL-divergence:

$$D_{\mathrm{KL}}(q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(z)) = \frac{1}{2}\sum_{j=1}^{k}\left(\mu_j(\boldsymbol{x})^2 + \sigma_j(\boldsymbol{x})^2 - 1 - \log\sigma_j(\boldsymbol{x})\right)$$

- Can be used to avoid Monte Carlo estimate with one sample (still needed for the reconstruction error though).

## VAE: Other examples of stochastic decoders



latent variable $z \sim q_\varphi(z|\boldsymbol{x})$
input $\boldsymbol{x}$
encoder $q_\varphi(z|\cdot)$
prob. dist. $q_\varphi(z|\boldsymbol{x})$
decoder $p_\theta(\boldsymbol{x}|\cdot)$
prob. dist. $p_\theta(\boldsymbol{x}|z)$
sample $\boldsymbol{x} \sim p_\theta(\boldsymbol{x}|z)$

- **Stochastic decoder for binary data:** With $\boldsymbol{x} \in \{0,1\}^d$, one sets

  $$p_{\boldsymbol{\theta}}(\boldsymbol{x}|z) = \mathrm{BernoulliVector}(\boldsymbol{x}; \boldsymbol{p}(z)) \quad \text{where} \quad \boldsymbol{p}(z) = \mathrm{NN}_{\boldsymbol{\theta}}(z).$$

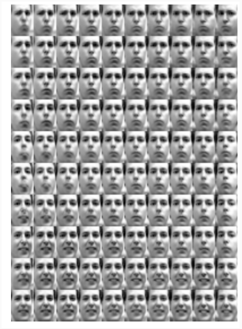
  Then, the likelihood is the binary cross-entropy:

  $$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|z) = \sum_{\ell=1}^{d} x_\ell \log p_\ell + (1 - x_\ell)\log(1 - p_\ell)$$

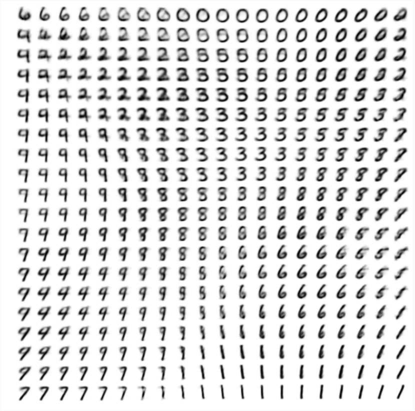
- **Stochastic decoder for discrete data:** Same approach with a NN that outputs a softmax array with the number of classes and cross-entropy...
- Here pixels are supposed independent resulting in noisy samples from $p_{\boldsymbol{\theta}}(\boldsymbol{x}|z)$... But one often outputs the expectation for visualization!

## VAE: Results

From the original paper: (Kingma and Welling, 2014): “Auto-Encoding Variational Bayes" (AEVB)



(a) Learned Frey Face manifold



(b) Learned MNIST manifold

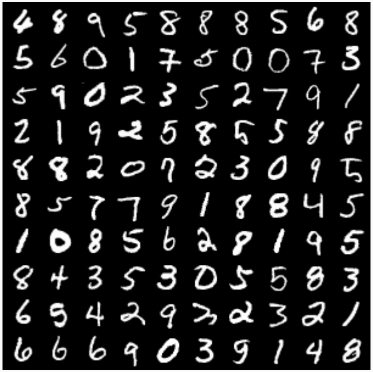
Figure 4: Visualisations of learned data manifold for generative models with two-dimensional latent space, learned with AEVB. Since the prior of the latent space is Gaussian, linearly spaced coordinates on the unit square were transformed through the inverse CDF of the Gaussian to produce values of the latent variables $\mathbf{z}$. For each of these values $\mathbf{z}$, we plotted the corresponding generative $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ with the learned parameters $\boldsymbol{\theta}$.

## VAE: Results

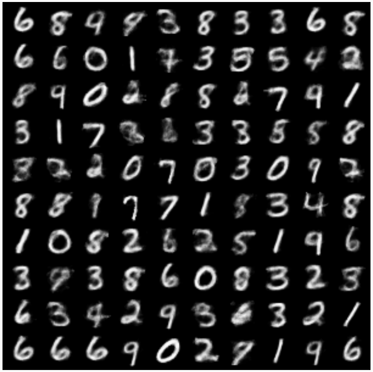
**Numerical illustration** with a subset of MNIST (1000 images only):

- Encoders and decoders are MLP trained for 1000 epochs.
- The **code dimension is $k = 2$ for visualization** (higher $k$ values would give better results).

Input test images:



Output:

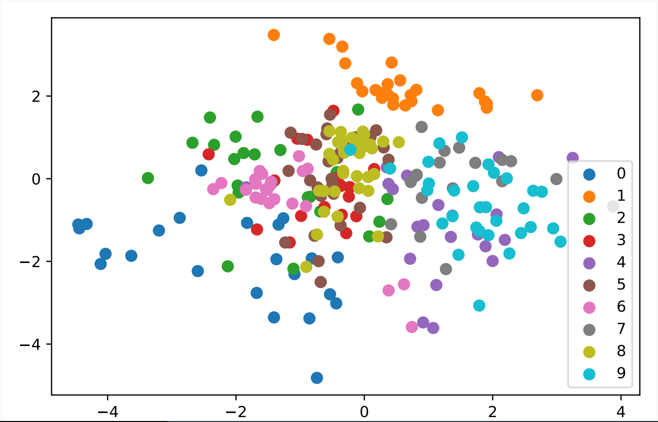


## VAE: Results

**Numerical illustration** with a subset of MNIST (1000 images only):

- Encoders and decoders are MLP trained for 1000 epochs.
- The **code dimension is $k = 2$ for visualization** (higher $k$ values would give better results).

2D latent code of 256 test images:



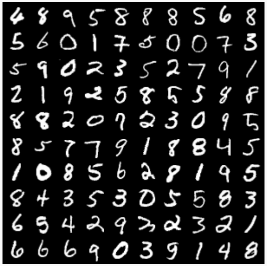
## VAE: Results

**Numerical illustration** with a subset of MNIST (1000 images only):

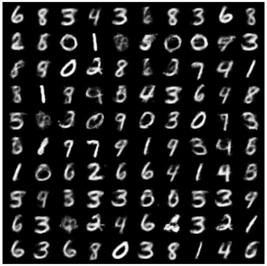
- Encoders and decoders are MLP trained for 1000 epochs.
- The **code dimension is $k = 2$ for visualization** (higher $k$ values would give better results).

AE VS VAE

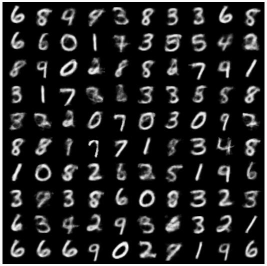
Input:



AE Output:



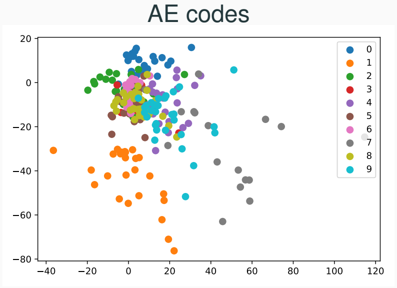
VAE Output:

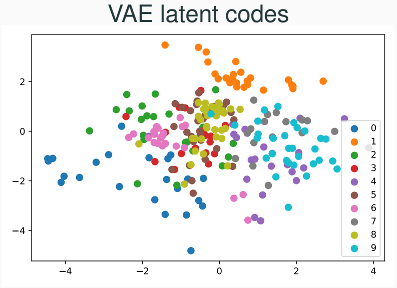


## VAE: Results

**Numerical illustration** with a subset of MNIST (1000 images only):

- Encoders and decoders are MLP trained for 1000 epochs.
- The **code dimension is $k = 2$ for visualization** (higher $k$ values would give better results).





The prior distribution enforces regularity/tightness of the VAE latent code distribution.

## Variational Autoencoders

VAE had a huge impact on the community (22 458 citations on Google Scholar!).

Lot of things can be improved (Kingma and Welling, 2019; Tomczak, 2022):

- Use more complex priors $p_{\boldsymbol{\theta}}(z)$ and decoder models $q_{\varphi}(z|\boldsymbol{x})$, eg using normalizing flows.
- Use a hierarchy of latent variables $z_1$, $z_2$, etc.

Issues regarding VAE (Kingma and Welling, 2019; Tomczak, 2022):

- *Posterior collapse*: All approximate posteriors $q_{\varphi}(z|\boldsymbol{x})$ are stucked to the prior to minimize the KL term of the ELBO.
- *Hole problem*: Some subset of the latent space is not populated by encoded data.
- *Blurriness of generative model*: produced images tend to be blurry as for standard autoencoders...

Pros of VAE:

- Very quick to sample once trained.

## VAE: SOTA resutls

(Vahdat and Kautz, 2020): “NVAE: A Deep Hierarchical Variational Autoencoder"

- VAE can be made competitive using well-designed architectures.

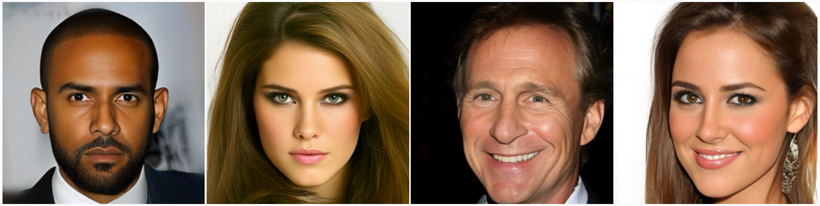


Figure 1: 256×256-pixel samples generated by NVAE, trained on CelebA HQ [28].

## VAE in practice

See Jakub Tomczak's implementation:

`https://github.com/jmtomczak/intro_dgm/blob/main/vaes/vae_example.ipynb`

## References

## References i

## References

Galerne, B., Gousseau, Y., and Morel, J.-M. (2011). Random phase textures: Theory and synthesis. *IEEE Trans. Image Process.*, 20(1):257 – 267.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc.

## References ii

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Karras, T., Laine, S., and Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational Bayes. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Kingma, D. P. and Welling, M. (2019). An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392.

## References iii

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2256–2265, Lille, France. PMLR.

Tomczak, J. M. (2022). *Deep Generative Modeling*. Springer International Publishing, Cham.

Vahdat, A. and Kautz, J. (2020). Nvae: A deep hierarchical variational autoencoder. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 19667–19679. Curran Associates, Inc.