Generative models for images III: Variational Auto-Encoders and Normalizing Flows

Bruno Galerne
bruno.galerne@univ-orleans.fr

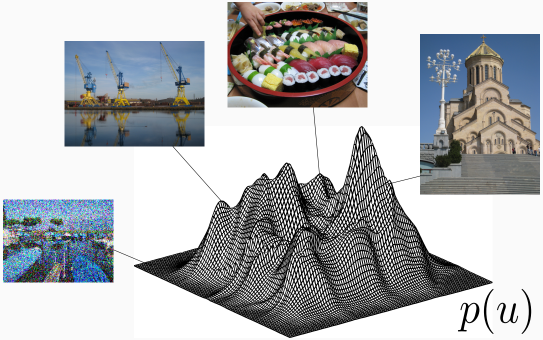
Master MVA 2024-25
Tuesday January 28, 2025

Institut Denis Poisson
Université d'Orléans, Université de Tours, CNRS
Institut universitaire de France (IUF)

Introduction on generative models

Generative models

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



(source: Charles Deledalle)

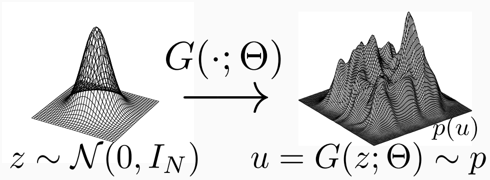
The images may represent:

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

2. Generate samples from this distribution.

Generative models

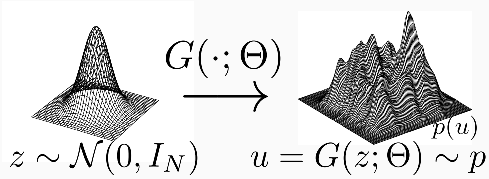
1. Model and/or learn a distribution $p(u)$ on the space of images.
2. Generate samples from this distribution.



- 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,. . .).

Variational autoencoders (VAE)

Variational autoencoders (VAE)

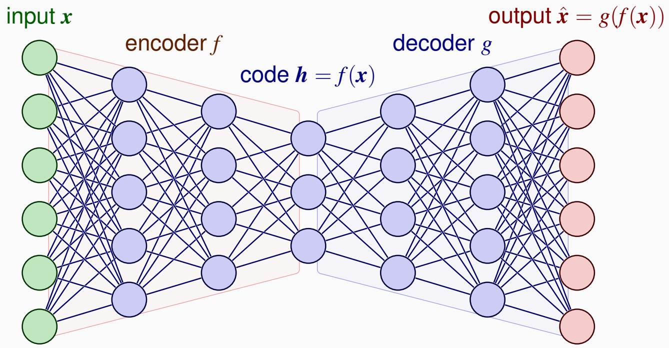
Main references:

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)

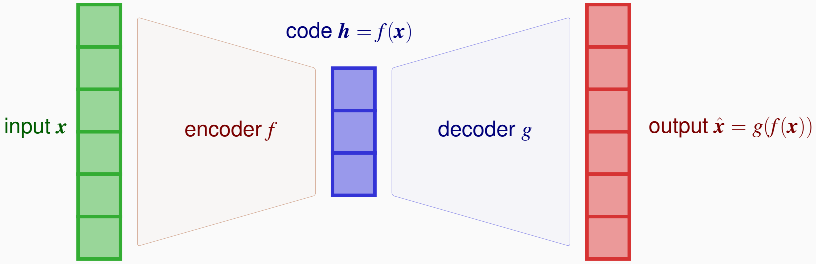
The network has a bottleneck hidden layer of lower dimension than the data.


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)

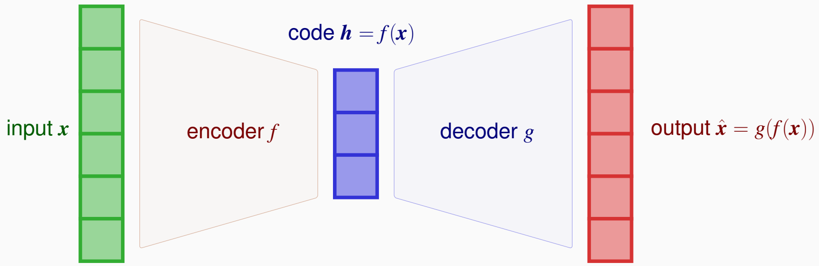
The network has a bottleneck hidden layer of lower dimension than the data.


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.

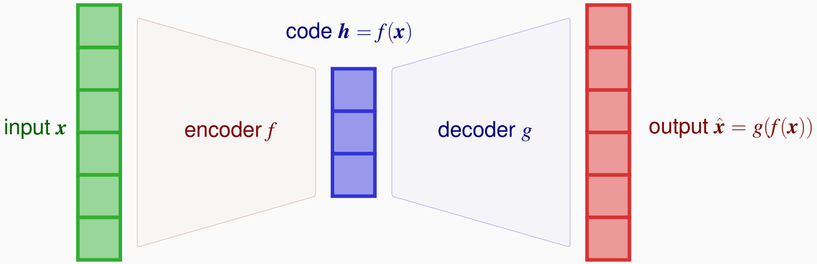


code $\boldsymbol{h} = f(\boldsymbol{x})$
input $\boldsymbol{x}$
encoder f
decoder g
output $\hat{\boldsymbol{x}} = g(f(\boldsymbol{x}))$

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 $\boldsymbol{x}$ within the dataset $\mathcal{D}$.
- If both f and g are linear, the best solution is the PCA projection using the first k principal 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 $\mathcal{D} = \{\boldsymbol{x}^{(i)},\ i = 1, \dots, N\}$:

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

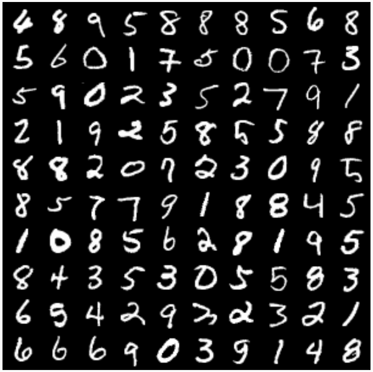
- Trade-off between the parameters of f and g, dimensions $d > k$ etc.

Autoencoders

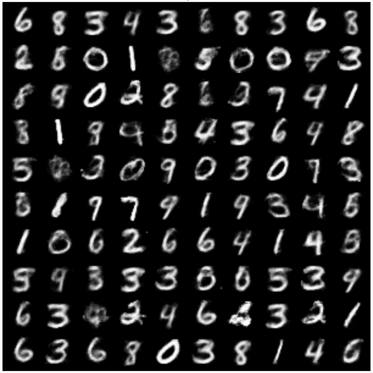
Numerical illustration with a subset of MNIST (1000 images only):

- 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:

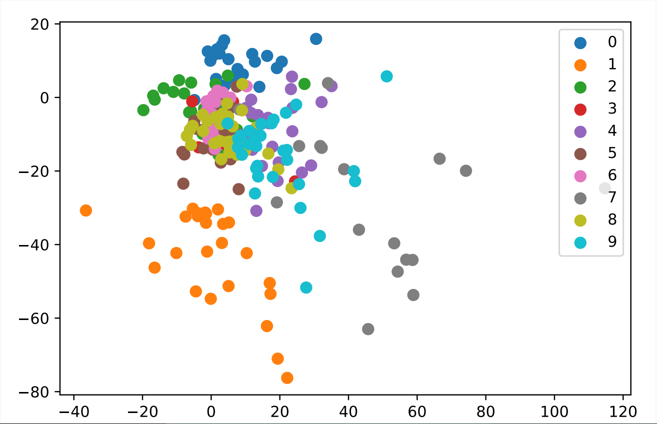


Autoencoders

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...

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.

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 $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is tractable: For a random minibatch $\mathcal{M} \subset \mathcal{D}$ having cardinal $M = |\mathcal{M}|$,

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

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 $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is tractable: For a random minibatch $\mathcal{M} \subset \mathcal{D}$ having cardinal $M = |\mathcal{M}|$,

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

- $\simeq$ 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 $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is tractable: For a random minibatch $\mathcal{M} \subset \mathcal{D}$ having cardinal $M = |\mathcal{M}|$,

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

- $\simeq$ 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 $\boldsymbol{x}$

Deep latent variable models

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 (GMM)**:

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

- 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 $p_{\boldsymbol{\theta}}(\boldsymbol{x}|z) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}(z), \boldsymbol{\Sigma}(z))$. Given that z has a density, this generalizes GMM with a mixture of 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 $p_{\boldsymbol{\theta}}(\boldsymbol{x}|z) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}(z), \boldsymbol{\Sigma}(z))$. Given that z has a density, this generalizes GMM with a mixture of 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 $p_{\boldsymbol{\theta}}(z|\boldsymbol{x}) = \dfrac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, z)}{p_{\boldsymbol{\theta}}(\boldsymbol{x})}$.

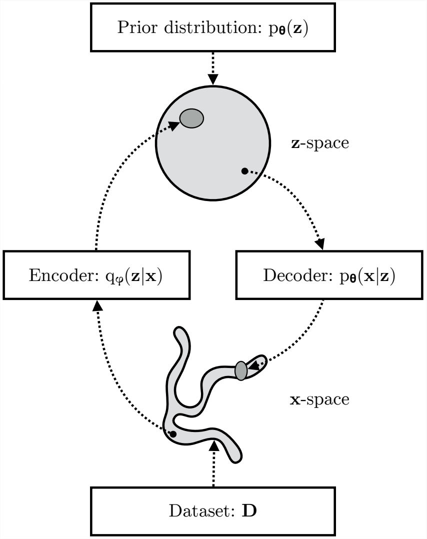
Variational autoencoders (VAE)

(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)

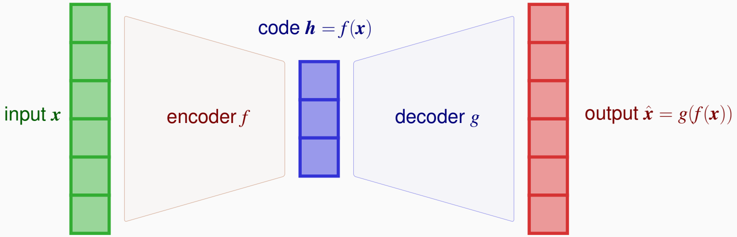
From (Kingma and Welling, 2019, p .20):

- 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 $p_\theta(x,z)$ factorized as $p_\theta(x,z) = p_\theta(z)p_\theta(x|z)$, with a prior distribution over latent space $p_\theta(z)$, and a stochastic decoder $p_\theta(x|z)$.
- The stochastic encoder $q_\varphi(z|x)$, also called inference model, approximates the true but intractable posterior $p_\theta(z|x)$ of the generative model.

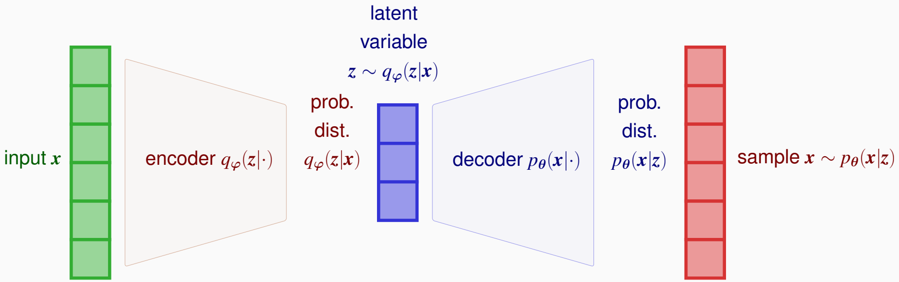

Prior distribution: $p_{\boldsymbol\theta}(\mathbf{z})$
$\mathbf{z}$-space
Encoder: $q_\varphi(\mathbf{z}|\mathbf{x})$
Decoder: $p_{\boldsymbol\theta}(\mathbf{x}|\mathbf{z})$
$\mathbf{x}$-space
Dataset: $\mathbf{D}$

Variational autoencoders (VAE)

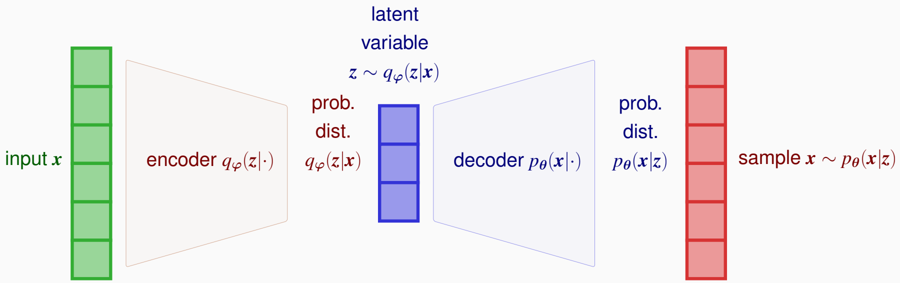
Autoencoders:


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)

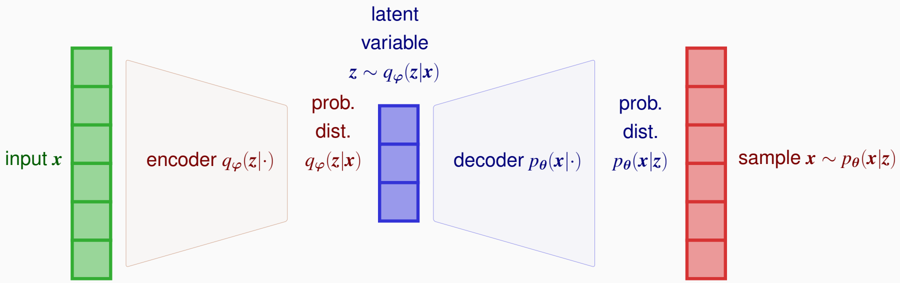

latent variable $z \sim q_\varphi(z|\boldsymbol{x})$
input $\boldsymbol{x}$
encoder $q_\varphi(z|\cdot)$
prob. dist. $q_\varphi(z|\boldsymbol{x})$
decoder $p_\theta(\boldsymbol{x}|\cdot)$
prob. dist. $p_\theta(\boldsymbol{x}|z)$
sample $\boldsymbol{x} \sim p_\theta(\boldsymbol{x}|z)$

Stochastic encoder:

- The encoder $q_\varphi(z|\boldsymbol{x})$ is understood as a parametric approximation of the true posterior $p_\theta(z|\boldsymbol{x})$.
- To achieve that the parameters φ 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
$z \sim q_\varphi(z|x)$
input x
encoder $q_\varphi(z|\cdot)$
prob. dist. $q_\varphi(z|x)$
decoder $p_\theta(x|\cdot)$
prob. dist. $p_\theta(x|z)$
sample $x \sim p_\theta(x|z)$

Next challenge: Learning!

- How can we learn the parameters θ (and φ) that maximize the log-likelihood

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

  is the (untractable) *marginal distribution* (or *model evidence*)?
- In fact we will only maximize a lower bound of each $\log p_\theta(x)$ 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 $\boldsymbol{x} \in \mathbb{R}^d$,

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

Kullback–Leibler divergence

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

$$D_{\mathrm{KL}}(P \parallel Q) = \begin{cases} \int_{\mathcal{X}} \log\left(\frac{P(dx)}{Q(dx)}\right) P(dx) & \text{if } P \text{ is absolutely continuous wrt } Q \\ +\infty & \text{otherwise} \end{cases}$$

where $\frac{P(dx)}{Q(dx)}$ is 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 distributions P and Q on some measurable space $\mathcal{X}$, one defines the **Kullback–Leibler divergence** of P wrt Q by, ,

$$D_{\mathrm{KL}}(P \parallel Q) = \begin{cases} \int_{\mathcal{X}} \log\left(\frac{P(dx)}{Q(dx)}\right) P(dx) & \text{if } P \text{ is absolutely continuous wrt } Q \\ +\infty & \text{otherwise} \end{cases}$$

where $\frac{P(dx)}{Q(dx)}$ is 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):

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

- 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)

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

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 $\boldsymbol{\theta}$.

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:

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

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

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:

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

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

VAE training

VAE Training algorithm:

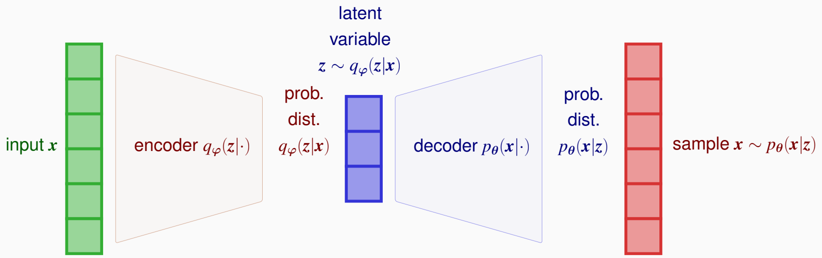
1. Draw a minibatch $\mathcal{M} = \{\boldsymbol{x}^{(i_1)}, \dots, \boldsymbol{x}^{(i_M)}\}$ of M samples from $\mathcal{D} = \{\boldsymbol{x}^{(i)},\ i = 1, \dots, N\}$
2. Draw M random $\varepsilon_m \sim p(\varepsilon)$, $m = 1, \dots, M$.
3. Compute $z^m = g(\varepsilon^{(m)}, \varphi, \boldsymbol{x}^{(i_m)}) \sim q_\varphi(z|\boldsymbol{x}^{(i_m)})$ 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 :** $p_\theta(z) = \mathcal{N}(z; \boldsymbol{0}, \boldsymbol{I})$ the prior is fixed without parameter to learn.
- **Gaussian stochastic decoder:** $p_\theta(\boldsymbol{x}|z) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_{\mathrm{dec}}(z), s^2\boldsymbol{I})$ with $\boldsymbol{\mu}_{\mathrm{dec}}(z) = \mathrm{NN}_\theta(z)$: 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 NN_φ and NN_θ 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 $z = g(\varepsilon, \boldsymbol{\varphi}, \boldsymbol{x}) = \boldsymbol{\mu}(\boldsymbol{x}) + \boldsymbol{\sigma}(\boldsymbol{x}) \odot \varepsilon$,

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

VAE: ELBO and Kullback–Leibler divergence

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

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

- 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 $q_{\varphi}(z|\boldsymbol{x})$ 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

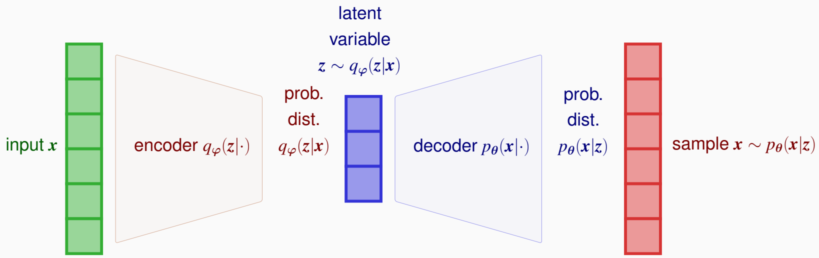
- **Gaussian stochastic encoder:** $q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}) = \mathcal{N}(z; \boldsymbol{\mu}(\boldsymbol{x}), \mathrm{diag}(\boldsymbol{\sigma}^2(\boldsymbol{x})))$ with $(\boldsymbol{\mu}(\boldsymbol{x}), \log\boldsymbol{\sigma}(\boldsymbol{x})) = \mathrm{NN}_{\boldsymbol{\varphi}}(\boldsymbol{x})$.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

1. Closed form formula for the KL-divergence:

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

2. Use this expression to propose another unbiased estimator $\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\boldsymbol{x})$ of the ELBO without MC estimate for the KL term.

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

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

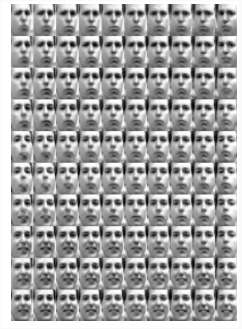
  Then, the likelihood is the binary cross-entropy:

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

- **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

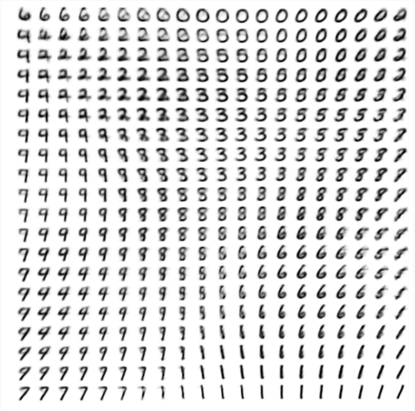

(b) Learned MNIST manifold

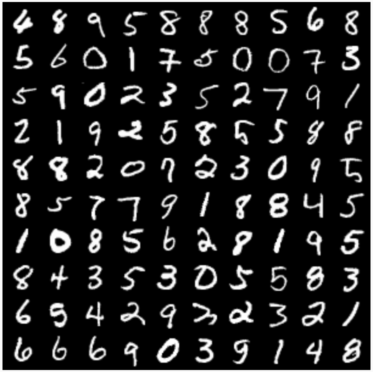
Figure 4: Visualisations of learned data manifold for generative models with two-dimensional latent space, learned with AEVB. Since the prior of the latent space is Gaussian, linearly spaced coordinates on the unit square were transformed through the inverse CDF of the Gaussian to produce values of the 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

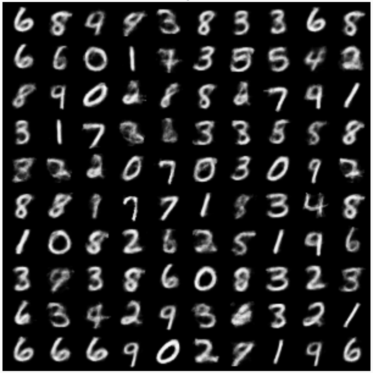
Numerical illustration with a subset of MNIST (1000 images only):

- 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:

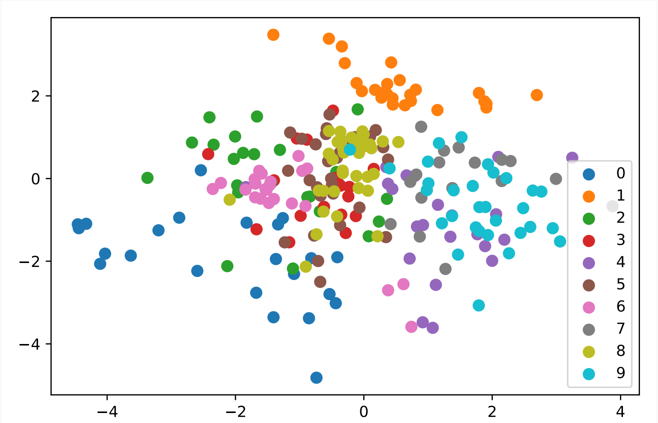


VAE: Results

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:



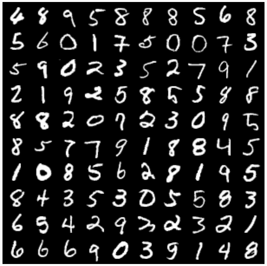
VAE: Results

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

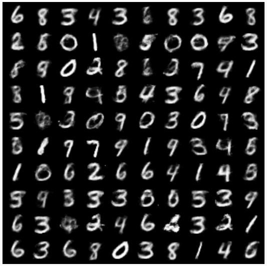
- Encoders and decoders are MLP trained for 1000 epochs.
- The **code dimension is $k = 2$ for visualization** (higher k values would give better results).

AE VS VAE

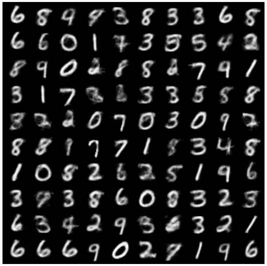
Input:



AE Output:



VAE Output:

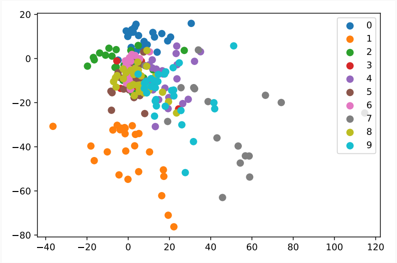


VAE: Results

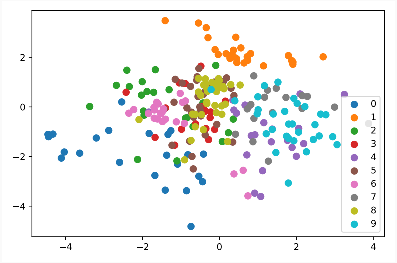
Numerical illustration with a subset of MNIST (1000 images only):

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

AE codes



VAE latent codes



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

Variational Autoencoders

VAE had a huge impact on the community (24 516 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 $q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})$, eg using normalizing flows (discussed later today).
- 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 $q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})$ 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.

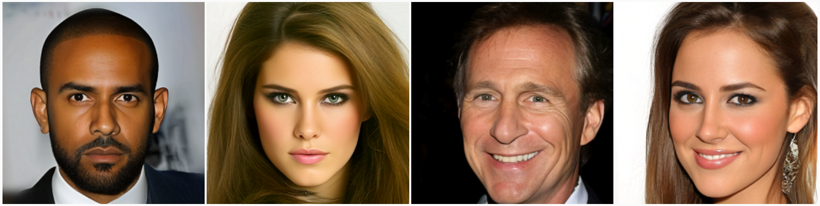


Figure 1: 256×256-pixel samples generated by NVAE, trained on CelebA HQ [28].

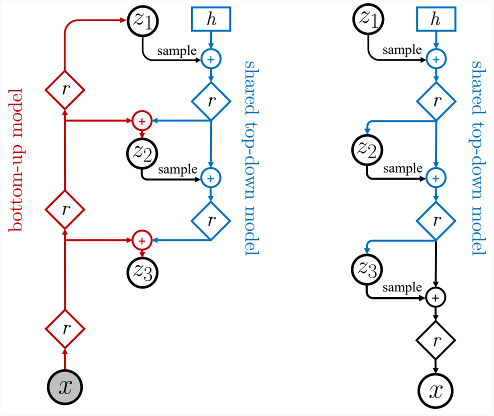
See also Very Deep VAE (Child, 2021).

NVAE: Architecture details

$$\mathcal{L}_{\text{VAE}}(\boldsymbol{x}) := \mathbb{E}_{q(z|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right] - D_{\text{KL}}(q(\boldsymbol{z}_1|\boldsymbol{x}) \parallel p(\boldsymbol{z}_1)) - \sum_{l=2}^{L} \mathbb{E}_{q(\boldsymbol{z}_{<l}|\boldsymbol{x})}\left[D_{\text{KL}}(q(\boldsymbol{z}_l|\boldsymbol{x}, \boldsymbol{z}_{<l}) \parallel p(\boldsymbol{z}_l|\boldsymbol{z}_{<l}))\right]$$

where $q(\boldsymbol{z}_{<l}|\boldsymbol{x}) = \prod_{i=1}^{l-1} q(\boldsymbol{z}_i|\boldsymbol{x}, \boldsymbol{z}_{<i})$ is the approx. posterior up to the $(l-1)^{th}$ group.

- Hierarchical architecture with shared encoder/decoder (Kingma et al., 2016).
- Complex cells using residual network (batch normalization, swish activation, ...).
- Conditioning based on shift in Gaussian distribution.



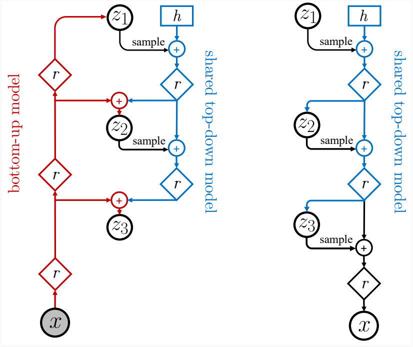
NVAE: Architecture details

- **Hierarchical prior:** $p(z_l|z_{<l}) = \mathcal{N}\left(\boldsymbol{\mu}(z_{<l}), \operatorname{diag} \boldsymbol{\sigma}^2(z_{<l})\right)$ is a normal distribution for the i^{th} variable in z_l in prior.
- **Residual distribution parameterization** of $q(z|\boldsymbol{x})$ relative to $p(z)$:

$$q(z_l|z_{<l}, \boldsymbol{x}) = \mathcal{N}\left(\boldsymbol{\mu}(z_{<l}) + \Delta\boldsymbol{\mu}(z_{<l}, \boldsymbol{x}), \operatorname{diag}\left(\boldsymbol{\sigma}^2(z_{<l}) \cdot \Delta\boldsymbol{\sigma}^2(z_{<l}, \boldsymbol{x})\right)\right)$$

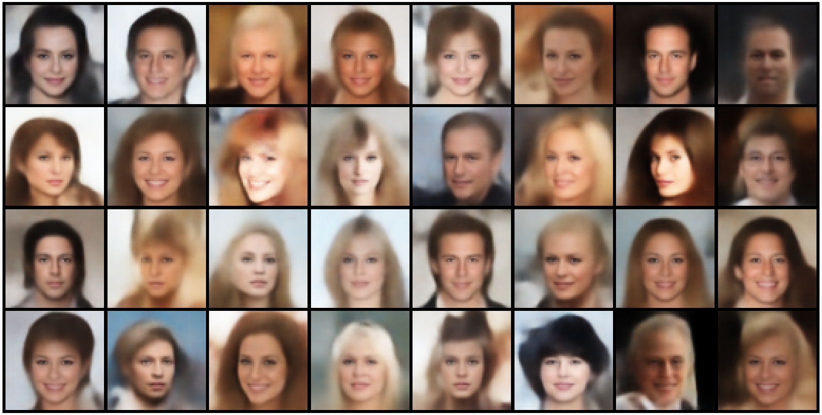
where $\Delta\mu(z_{<l}, \boldsymbol{x})$ and $\Delta\boldsymbol{\sigma}^2(z_{<l}, \boldsymbol{x})$ are the relative location and scale of the approximate posterior with respect to the prior.

- $\Delta\boldsymbol{\mu}(z_{<l}, \boldsymbol{x})$ and $\Delta\boldsymbol{\sigma}^2(z_{<l}, \boldsymbol{x})$ depends on features $\boldsymbol{x}_l$ with the same level
- Favors natural level of details hierarchy.



NVAE: Results of toy implementation

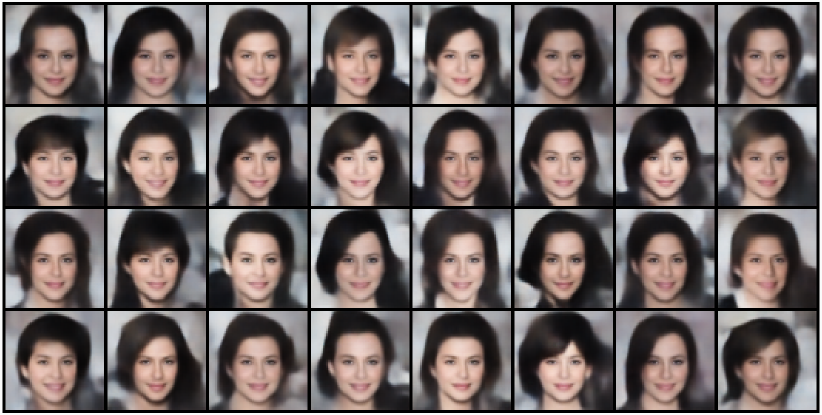
Samples of 64×64 portraits



Sample all levels

NVAE: Results of toy implementation

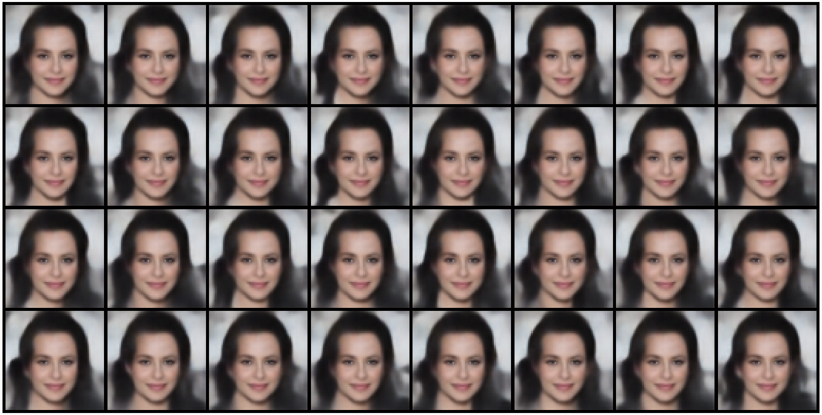
Samples of 64×64 portraits



Fixed z_1

NVAE: Results of toy implementation

Samples of 64×64 portraits



Fixed z_1 and z_2

VAE in practice

Today's practice session based on this hierarchical architecture.

Other ressources:

- Jakub Tomczak's implementation:
  `https://github.com/jmtomczak/intro_dgm/blob/main/vaes/vae_example.ipynb`
  ... but it does not use the closed-form formula

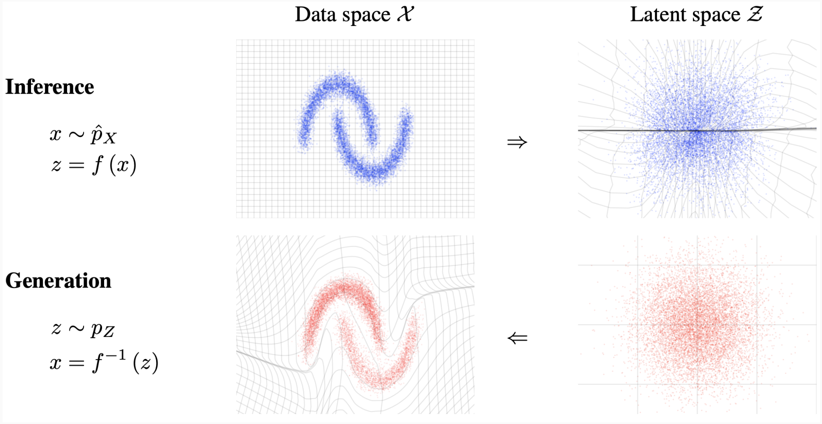
$$D_{\mathrm{KL}}(q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z})) = \frac{1}{2}\sum_{j=1}^{k}\left(\mu_j(\boldsymbol{x})^2 + \sigma_j(\boldsymbol{x})^2 - 1 - \log \sigma_j(\boldsymbol{x})^2\right).$$

- Simple MLP for MNIST (PyTorch examples):
  `https://github.com/pytorch/examples/tree/main/vae`

Normalizing flows

Normalizing flows

Motivation: Learn an invertible mapping from the data space to the latent space.



(source: From (Dinh et al., 2017))

- Latent space and data space have the same dimension.
- The latent distribution is generally assumed to be Gaussian.

Normalizing flows

Two main issues:

1. Parameterize a generic parametric invertible transform g_θ.
2. Learn the parameters θ to fit the dataset $\mathcal{D} = \{\boldsymbol{x}^{(i)},\ i = 1, \dots, N\} \subset \mathbb{R}^d$.

Learning is performed by simple loglikelihood maximization:

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

- Here $p_{\boldsymbol{\theta}} = (g_\theta)_{\#}\pi_0$ with $\pi_0 = \mathcal{N}(0, I_d)$.
- Since g_θ is assumed to a **diffeomorphism**, the expression is given thanks to the change of variable formula.
- In practice the dataset $\mathcal{D}$ is discrete and one adds noise to the data to deal with quantization and have a density (Kingma and Dhariwal, 2018).

Invertible transformations

The density of $p_\theta = (g_{\boldsymbol{\theta}})_\# \pi_0$ is given by a **change of variable**.

- We assume that g_θ is a **diffeomorphism**
- For any $f \in \mathcal{C}_c(\mathbb{R}^d, \mathbb{R})$

$$\mathbb{E}_{p_\theta}(f(X)) = \int_{\mathbb{R}^d} f(x) p_\theta(x) dx$$

$$\begin{aligned}\mathbb{E}_{p_\theta}(f(X)) &= \mathbb{E}_{\pi_0}(f(g_\theta(Z))) \\ &= \int_{\mathbb{R}^d} f(g_\theta(z)) p_0(z) dz \quad (z = g_\theta^{-1}(x)) \\ &= \int_{\mathbb{R}^d} f(x) p_0(g_\theta^{-1}(x)) |J(g_\theta^{-1})(x)| dx.\end{aligned}$$

where $|J(g_\theta^{-1})(x)| = \left| \det \left(\frac{\partial g_{\theta,m}^{-1}}{\partial x_n}(x) \right)_{1 \leq m,n \leq d} \right|$ is the determinant of the Jacobian.

Expression of the density:

$$\boxed{p_\theta(x) = p_0(g_\theta^{-1}(x)) |J(g_\theta^{-1})(x)|}$$

Remark: Generalized using the co-area/area formula (Caterini et al., 2021)).

Maximizing the log-likelihood

Expression of the density:

$$\boxed{p_\theta(x) = p_0(g_\theta^{-1}(x))|J(g_\theta^{-1})(x)|}$$

- Hence, maximizing the **log-likelihood** is equivalent to maximizing

$$\boxed{\mathcal{L}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x}_i \in \mathcal{D}} \log(p_0(g_\theta^{-1}(\boldsymbol{x}_i))) + \log(|J(g_\theta^{-1}(\boldsymbol{x}_i))|)}$$

- Short notation: $J_\theta(x) := J(g_\theta^{-1})(x)$.

Conditions on the transformations:

- g_θ and g_θ^{-1} are easy to compute and differentiate.
- The Jacobian J_θ is easy to compute and differentiate.
- But also g_θ should be as complex as required by the data...

Compositions of transformations

Composition of transformation: To obtain a complex flow one decomposes the flow as K “simple” diffeomorphisms:

$$g_\theta = g_\theta^0 \circ g_\theta^1 \circ \cdots \circ g_\theta^K$$

Then

$$\log(|J(g_\theta^{-1}(x))|) = \sum_{k=1}^{K} \log(|J((g_\theta^k)^{-1}(x^k))|)$$

with x^k the proper intermediate step in the sequence.

Different types of flows

- In (Rezende and Mohamed, 2015) planar and radial flows are presented.
- Two other very efficient flows (Dinh et al., 2017, 2015):
  - **Affine coupling layer**.
  - **Invertible 1x1 convolution**.

Different types of flows

- In (Rezende and Mohamed, 2015) planar and radial flows are presented.
- Two other very efficient flows (Dinh et al., 2017, 2015):
  - **Affine coupling layer**.
  - **Invertible 1x1 convolution**.

- How does the **affine coupling layer** work?
  - We split $x \in \mathbb{R}^d$ in $x = (x_0, x_1)$ with $x_0 \in \mathbb{R}^{d_0}$, $x_1 \in \mathbb{R}^{d_1}$.
  - **Forward** transform $g_\theta(x) = (x_0, \exp[s_\theta(x_0)] \odot x_1 + t_\theta(x_0))$ with s_θ and t_θ being any network.
  - **Reverse** transform $g_\theta^{-1}(x) = (x_0, (x_1 - t_\theta(x_0)) \oslash \exp[s_\theta(x_0)])$.
  - **Log-Jacobian**: $\log(|J_\theta(x)|) = \sum_{i=1}^{d_1} s_\theta(x_0)_i$.

Different types of flows

- In (Rezende and Mohamed, 2015) planar and radial flows are presented.
- Two other very efficient flows (Dinh et al., 2017, 2015):
  - **Affine coupling layer**.
  - **Invertible 1x1 convolution**.

- How does the **affine coupling layer** work?
  - We split $x \in \mathbb{R}^d$ in $x = (x_0, x_1)$ with $x_0 \in \mathbb{R}^{d_0}$, $x_1 \in \mathbb{R}^{d_1}$.
  - **Forward** transform $g_\theta(x) = (x_0, \exp[s_\theta(x_0)] \odot x_1 + t_\theta(x_0))$ with s_θ and t_θ being any network.
  - **Reverse** transform $g_\theta^{-1}(x) = (x_0, (x_1 - t_\theta(x_0)) \oslash \exp[s_\theta(x_0)])$.
  - **Log-Jacobian**: $\log(|J_\theta(x)|) = \sum_{i=1}^{d_1} s_\theta(x_0)_i$.

- How does the **invertible 1x1 convolution** work?
  - Matrix $\mathbf{W}_\theta \in \mathbb{R}^{C \times C}$ (number of channels), $x \in \mathbb{R}^{H \times W \times C}$.
  - **Forward** transform $g_\theta(x)_{i,j} = \mathbf{W}_\theta x_{i,j}$.
  - **Reverse** transform $g_\theta^{-1}(x)_{i,j} = \mathbf{W}_\theta^{-1} x_{i,j}$.
  - **Log-Jacobian** $\log(|J_\theta(x)|) = H \times W \times \log(|\mathbf{W}_\theta|)$.

Different types of flows

- There is no spatial convolution in these operations.
- However there a way to generate the image in a **multiscale** way (Dinh et al., 2017): Use a **squeeze** layer that change an image of size $H \times W \times C$ into an image of size $H/2 \times W/2 \times 4C$ by stacking spatial neighbors in the channel component.
- Then the next 1x1 convolution mixes the formerly spatial neighbors.

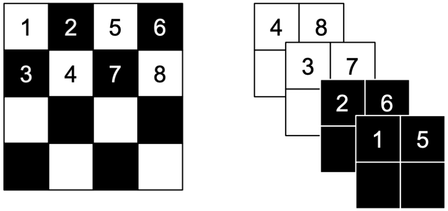
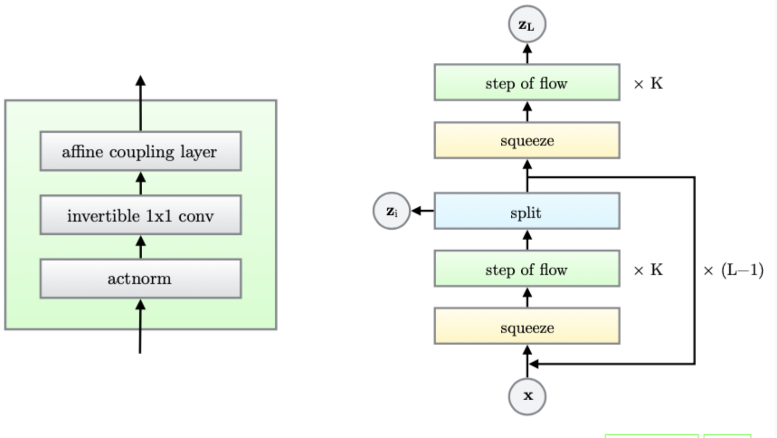


Figure 3: Masking schemes for affine coupling layers. On the left, a spatial checkerboard pattern mask. On the right, a channel-wise masking. The squeezing operation reduces the $4 \times 4 \times 1$ tensor (on the left) into a $2 \times 2 \times 4$ tensor (on the right). Before the squeezing operation, a checkerboard pattern is used for coupling layers while a channel-wise masking pattern is used afterward.

(source: From (Dinh et al., 2017))

Generative Flow (GLOW) (Kingma and Dhariwal, 2018)



(a) One step of our flow. (b) Multi-scale architecture (Dinh et al., 2016).

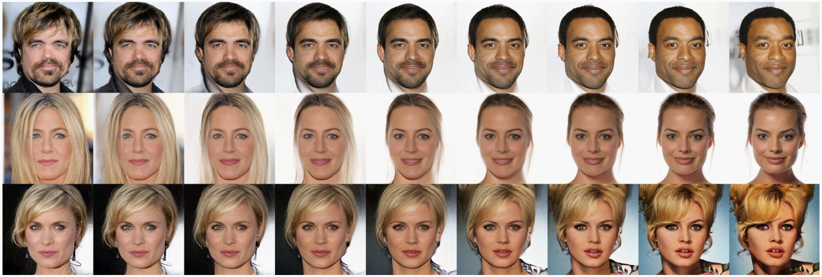
(source: From (Kingma and Dhariwal, 2018))

- Combining actnorm, invertible convolution and affine coupling layers (multiple times).
- The “actnorm” layer is simply an affine layer.

Generative Flow (GLOW) (Kingma and Dhariwal, 2018)

High quality results

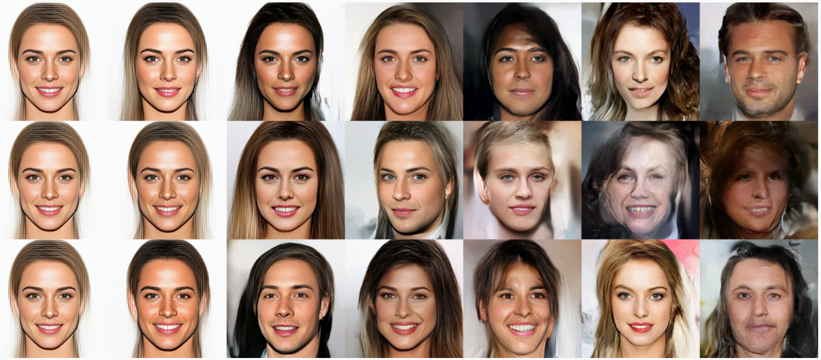
Generative Flow (GLOW) (Kingma and Dhariwal, 2018)



Linear interpolation in latent space between real images

- This experiments uses both the **inference** and **generation** of the flow.
- Not so easy to do with a GAN: Why?

Generative Flow (GLOW) (Kingma and Dhariwal, 2018)



Effect of change of temperature: Samples obtained at temperatures 0, 0.25, 0.6, 0.7, 0.8, 0.9, 1.0.

- **The temperature to be decreased** for high-quality image generation: latent codes z are sampled from $\mathcal{N}(0, \sigma I_d)$ with $\sigma < 1$.
- Temperature modulation also used for VAE.

References

References

Caterini, A. L., Loaiza-Ganem, G., Pleiss, G., and Cunningham, J. P. (2021). Rectangular flows for manifold learning. In *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*.

Child, R. (2021). Very deep {vae}s generalize autoregressive models and can outperform them on images. In *International Conference on Learning Representations*.

Dinh, L., Krueger, D., and Bengio, Y. (2015). NICE: non-linear independent components estimation. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2017). Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

References ii

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

Kingma, D. P. and Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improved variational inference with inverse autoregressive flow. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

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*.

References iii

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

Rezende, D. J. and Mohamed, S. (2015). Variational inference with normalizing flows. In Bach, F. R. and Blei, D. M., editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1530–1538. JMLR.org.

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.