Generative models for images I

Bruno Galerne
bruno.galerne@univ-orleans.fr

BIP Artificial Intelligence for Science
University of Caen Normandy, Wednesday August 28, 2024

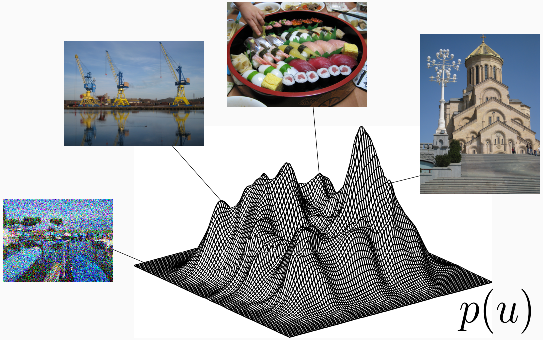
Institut Denis Poisson
Université d'Orléans, Université de Tours, CNRS
Institut universitaire de France (IUF)

Material for the course is here:
https://www.idpoisson.fr/galerne/caen2024/index.html

Introduction on generative models

Generative models

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



(source: Charles Deledalle)

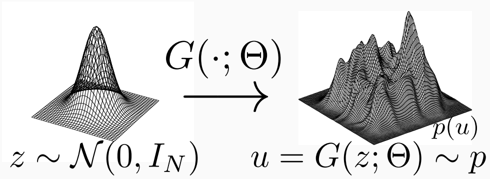
The images may represent:

- 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

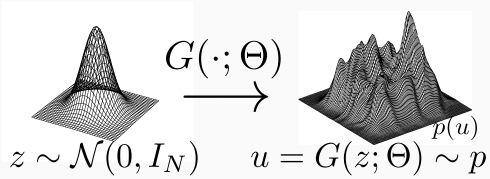
1. Model and/or learn a distribution $p(u)$ on the space of images.
2. Generate samples from this distribution.



- 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:

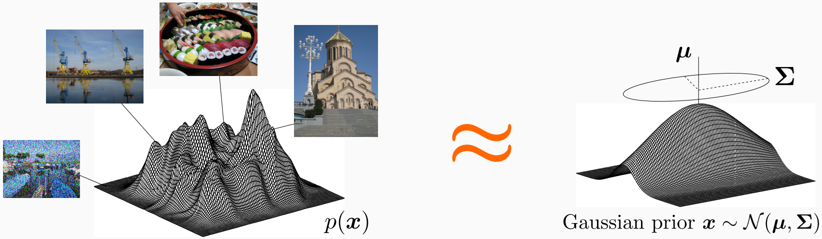
- A deterministic function (e.g. convolution operator),
- A neural network with learned parameter,
- An iterative optimization algorithm (gradient descent,...),
- A stochastic sampling algorithm (e.g. MCMC, Langevin diffusion,...).

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.



$p(\boldsymbol{x})$
$\approx$
$\boldsymbol{\mu}$
$\boldsymbol{\Sigma}$
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:

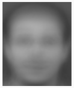
$$\mathcal{D} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$$

$= \Bigg\{$

$, \dots \Bigg\}_{\times N}$

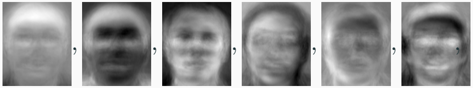
- Estimate the mean

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



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

$\hat{\boldsymbol{E}} = \Bigg\{$



$, \dots \Bigg\}_{\times N}$

eigenvectors of $\hat{\boldsymbol{\Sigma}}$, *i.e.*, main variation axis

Image generation: Gaussian model

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

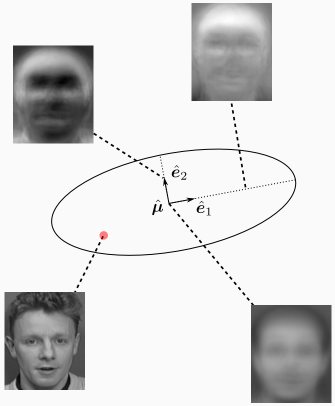
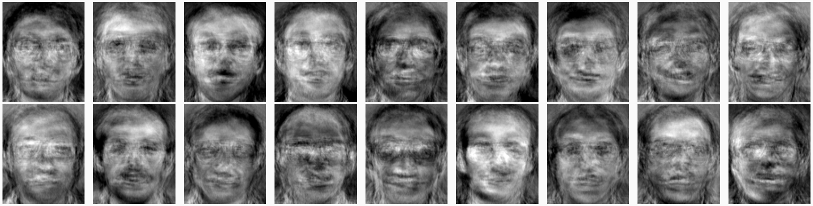


Image generation: Gaussian model

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

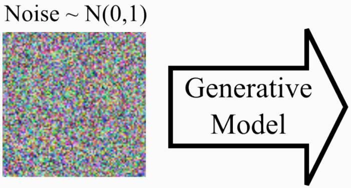
$$\begin{cases} z & \sim \mathcal{N}(0, \boldsymbol{I}_d) \quad \leftarrow \text{Generate random latent variable} \\ \boldsymbol{x} & = \hat{\boldsymbol{\mu}} + \hat{\boldsymbol{E}}\hat{\boldsymbol{\Lambda}}^{1/2} z \end{cases}$$



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 (with independence !).
- The generator corresponds to a one layer liner neural network (without non-linearities).

Image generation: Gaussian model



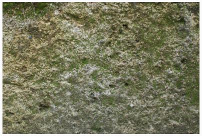
- Deep generative modeling consists in learning non-linear generative models to reproduce complex data such as realistic images.
- It relies on deep neural networks and several solutions have been proposed since the “Deep learning revolution” (2012).

Generative models: Examples

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

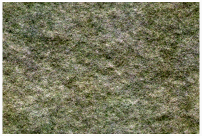
- Data: A single texture image $\boldsymbol{h}$.
- Inferred distribution: p is the stationary Gaussian distribution with similar mean and covariance statistics.
- z is a Gaussian white noise image (each pixel is iid with standard normal distribution).
- G is a convolution operator with know parameters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

Data

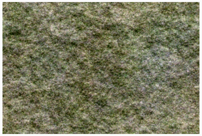


Spot $\boldsymbol{h}$

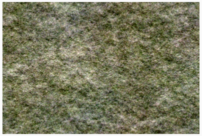
Generated images



$G(z_1;\Theta)$



$G(z_2;\Theta)$



$G(z_3;\Theta)$

Generative models: Examples

Generative Adversarial 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 Θ learned using an adversarial discriminator network $D(\cdot;\Theta_D)$.

Data

MNIST: handwritten digits

Generated images

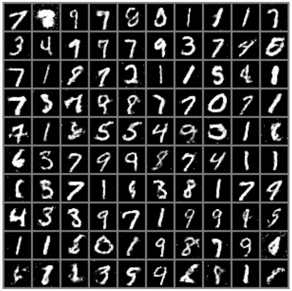

Fake images (100 epochs)

Image size: 28×28 px

Generative models: Examples

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

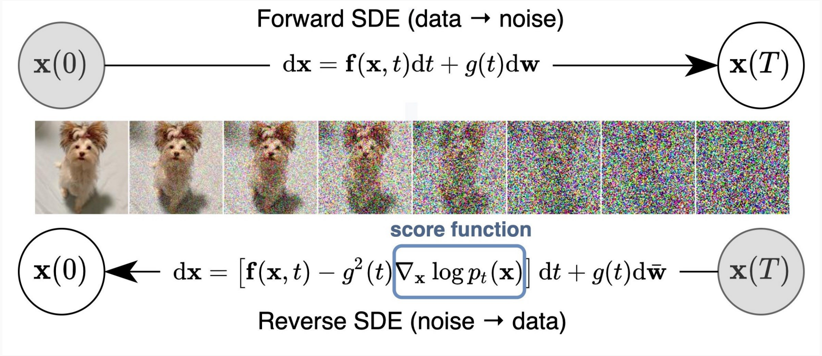
Image size:

1024×1024 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 (DDPM): One of the first paper producing images with reasonable resolution.

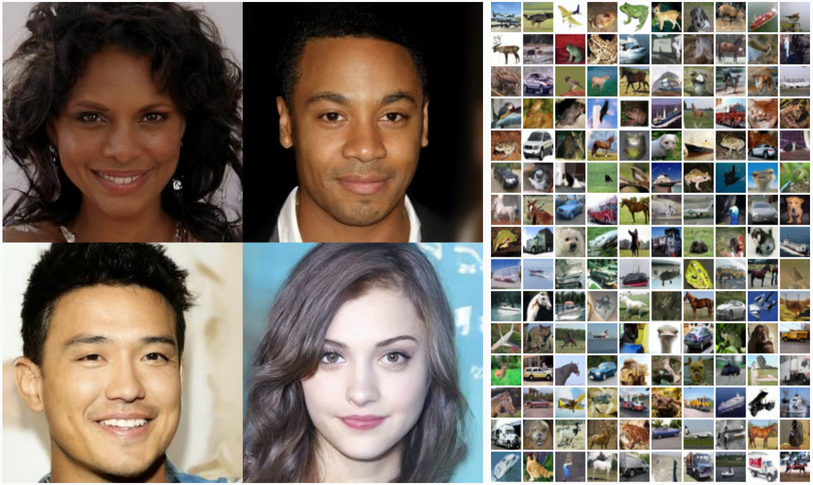


Figure 1: Generated samples on CelebA-HQ 256×256 (left) and unconditional CIFAR10 (right)

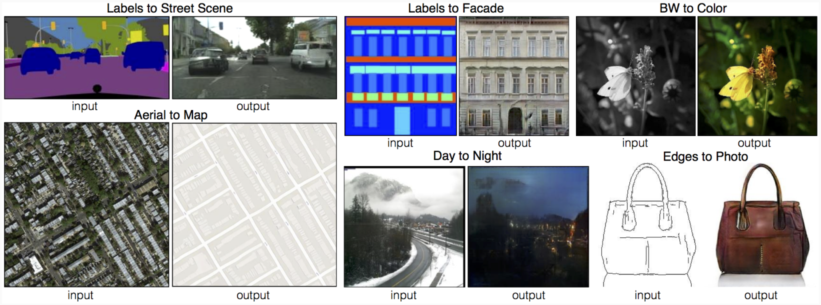
Generative models: Motivations

Why generative models are interesting ?

- **Generating realistic images is important by itself** for entertainment industry (visual effects, video games, augmented reality...), design, advertising industry,...
- **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 opens the way to **non trivial image manipulation** using **conditional generative models**.

Conditional generative models: Examples

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: Examples

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

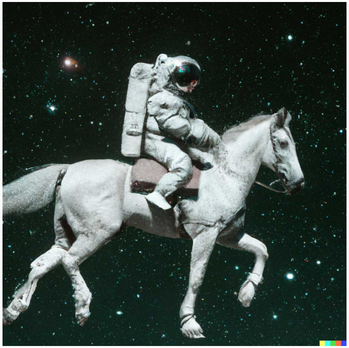
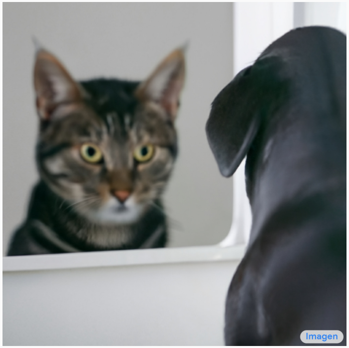
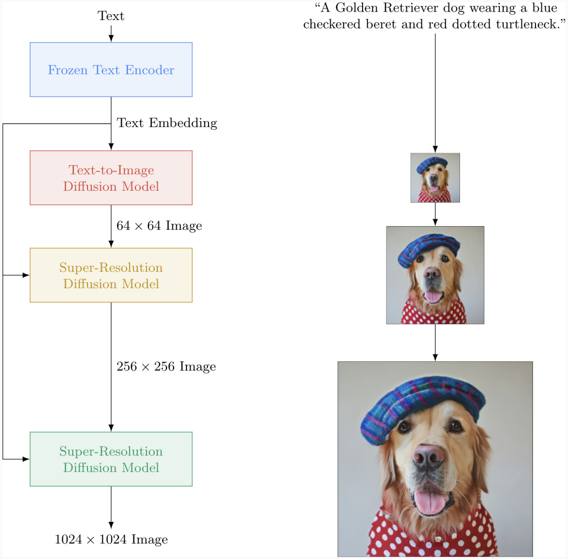


Imagen (Google)
Input: A dog looking curiously in the mirror, seeing a cat.



Conditional generative models: Examples

Imagen pipeline:



Text
Frozen Text Encoder
Text Embedding
Text-to-Image Diffusion Model
64 × 64 Image
Super-Resolution Diffusion Model
256 × 256 Image
Super-Resolution Diffusion Model
1024 × 1024 Image
“A Golden Retriever dog wearing a blue checkered beret and red dotted turtleneck.”

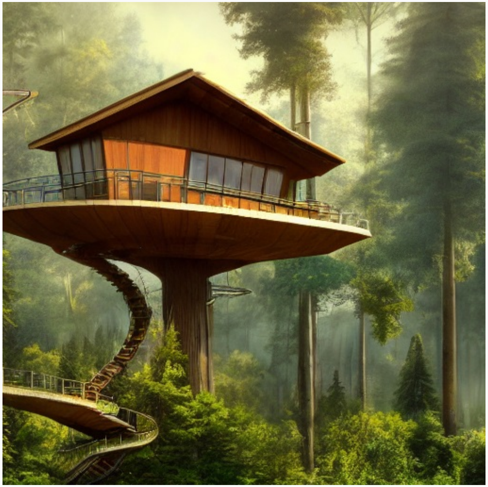
(source: (Saharia et al., 2022))

Conditional generative models: Examples

In August 2022, StableDiffusion was released:

- Based on the paper (Rombach et al., 2022)
- **Open source!**

futuristic tree house, hyper realistic, epic composition, cinematic, landscape vista photography by Carr Clifton & Galen Rowell, Landscape veduta photo by Dustin Lefevre & tdraw, detailed landscape painting by Ivan Shishkin, rendered in Enscape, Miyazaki, Nausicaa Ghibli, 4k detailed post processing, unreal engine
Steps: 50, Sampler: PLMS, CFG scale: 9, Seed: 2937258437



Diffusion models in 2023

Diffusion models are considered mature models and have been used in a large variety of frameworks.

- Diffusion models **beyond image generation**: Text to video, motion generation, proteins, soft robots,...
- **Control of (latent) diffusion models**((Ruiz et al., 2023), (Zhang et al., 2023),...)
- **Diffusion models as priors for imaging inverse problems** ((Chung et al., 2023), (Song et al., 2023), lot of applications in medical imaging, etc.)

Diffusion models in 2023

DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation

Nataniel Ruiz[*,1,2] Yuanzhen Li[1] Varun Jampani[1]
Yael Pritch[1] Michael Rubinstein[1] Kfir Aberman[1]
[1] Google Research [2] Boston University

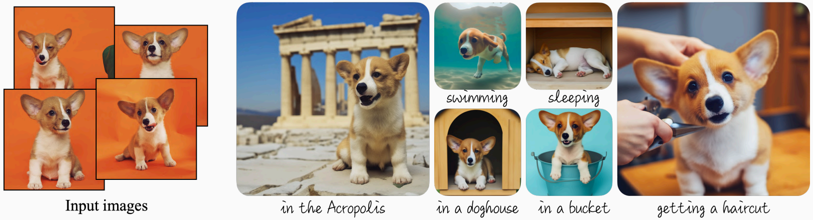


Figure 1. With just a few images (typically 3-5) of a subject (left), *DreamBooth*—our AI-powered photo booth—can generate a myriad of images of the subject in different contexts (right), using the guidance of a text prompt. The results exhibit natural interactions with the environment, as well as novel articulations and variation in lighting conditions, all while maintaining high fidelity to the key visual features of the subject.

(source: (Ruiz et al., 2023))

Diffusion models in 2023

Adding Conditional Control to Text-to-Image Diffusion Models

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala
Stanford University
{lvmin, anyirao, maneesh}@cs.stanford.edu

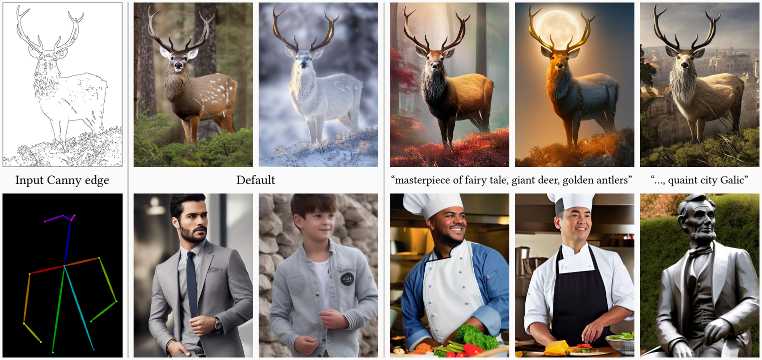


Figure 1: Controlling Stable Diffusion with learned conditions. ControlNet allows users to add conditions like Canny edges (top), human pose (bottom), *etc*., to control the image generation of large pretrained diffusion models. The default results use the prompt "a high-quality, detailed, and professional image". Users can optionally give prompts like the "chef in kitchen".

(source: ControlNet (Zhang et al., 2023))

Diffusion models in 2023

Diffusion posterior sampling for general noisy inverse problems (Chung et al., 2023)

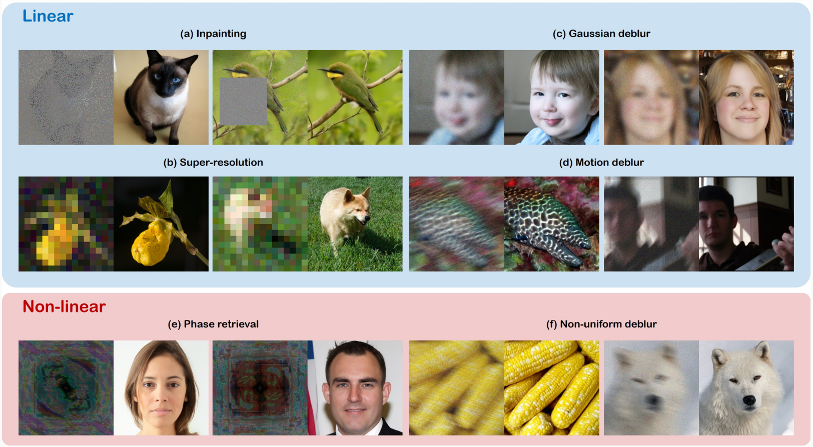


Figure 1: Solving noisy linear, and nonlinear inverse problems with diffusion models. Our reconstruction results (right) from the measurements (left) are shown.

(source: (Chung et al., 2023))

Generative models for images: Plan of the course

1. Introduction to generative models for images (done)
2. Variational AutoEncoders (VAEs)
3. Generative Adversarial Networks (GANs)
4. Diffusion models
5. Application of generative models for imaging inverse problems

Variational autoencoders (VAE)

Variational autoencoders (VAE)

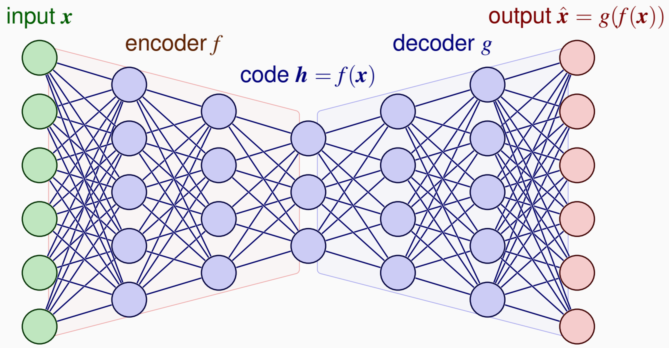
Main references:

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)

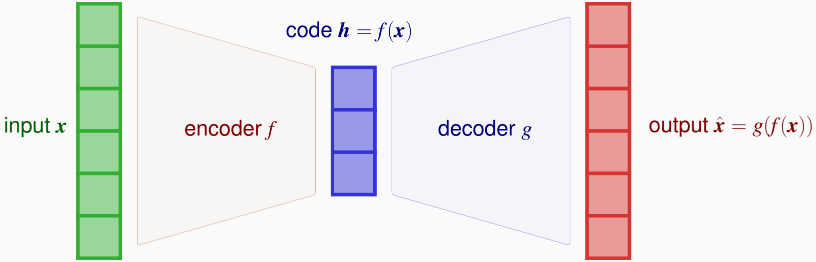
The network has a bottleneck hidden layer of lower dimension than the data.


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.

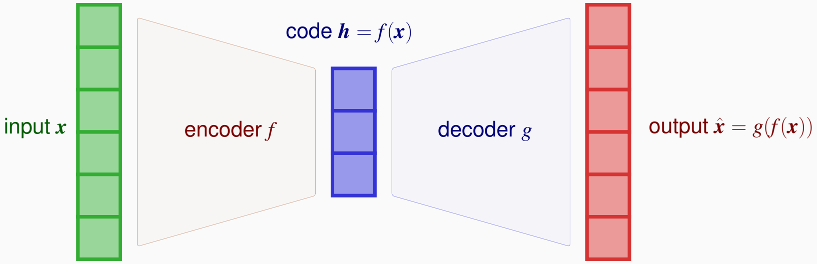


input $\boldsymbol{x}$
encoder f
code $\boldsymbol{h} = f(\boldsymbol{x})$
decoder g
output $\hat{\boldsymbol{x}} = g(f(\boldsymbol{x}))$

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)

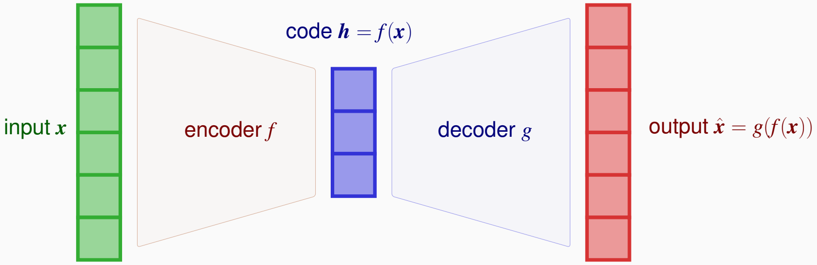
The network has a bottleneck hidden layer of lower dimension than the data.


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

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

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

Autoencoders


input $\boldsymbol{x}$
encoder f
code $\boldsymbol{h} = f(\boldsymbol{x})$
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.

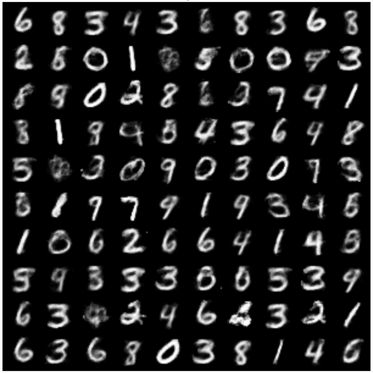
Autoencoders

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:

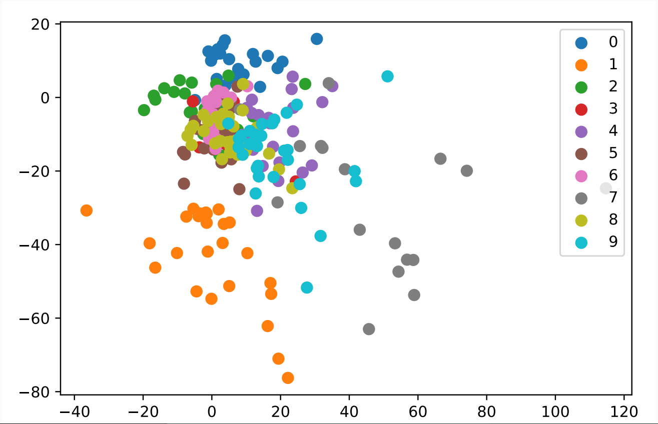


Autoencoders

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 $p_{\boldsymbol{\theta}}(z|\boldsymbol{x}) = \dfrac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, z)}{p_{\boldsymbol{\theta}}(\boldsymbol{x})}$.

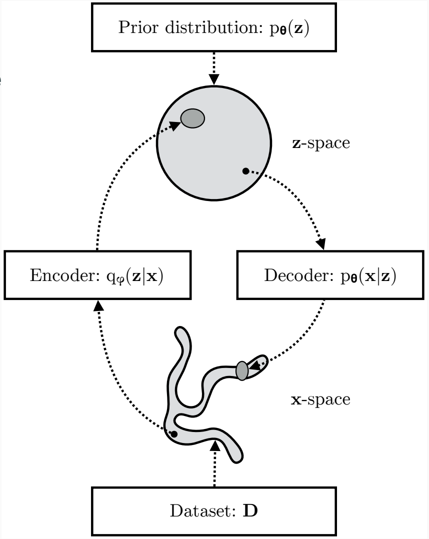
Variational autoencoders (VAE)

(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)

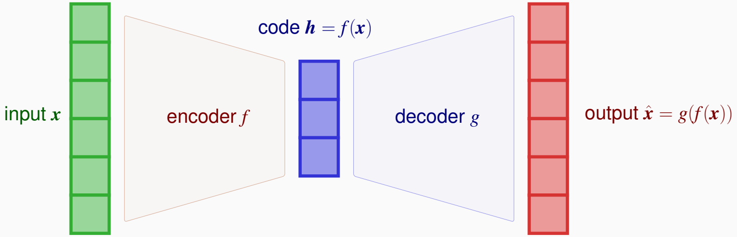
From (Kingma and Welling, 2019, p .20):

- A VAE learns stochastic mappings between an observed $\boldsymbol{x}$-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 $q_{\boldsymbol{\varphi}}(z|\boldsymbol{x})$, also called inference model, approximates the true but intractable posterior $p_{\boldsymbol{\theta}}(z|\boldsymbol{x})$ of the generative model.

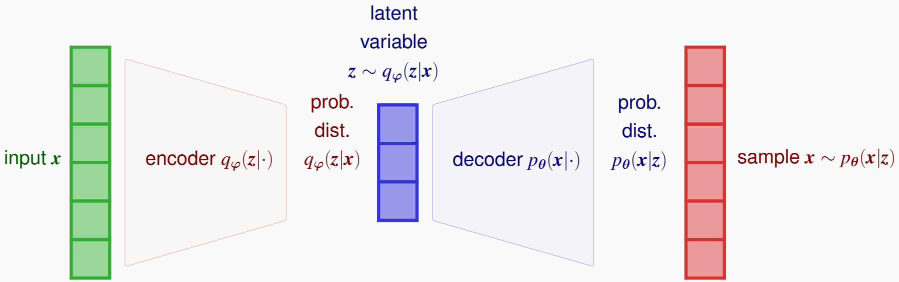

Prior distribution: $p_{\boldsymbol{\theta}}(\mathbf{z})$
$\mathbf{z}$-space
Encoder: $q_{\varphi}(\mathbf{z}|\mathbf{x})$
Decoder: $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$
$\mathbf{x}$-space
Dataset: $\mathbf{D}$

Variational autoencoders (VAE)

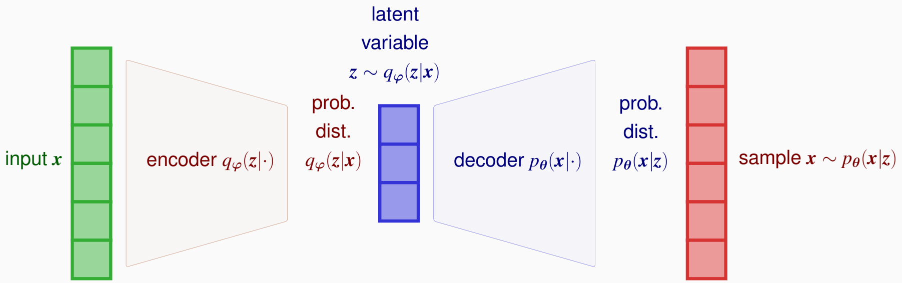
Autoencoders:


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)



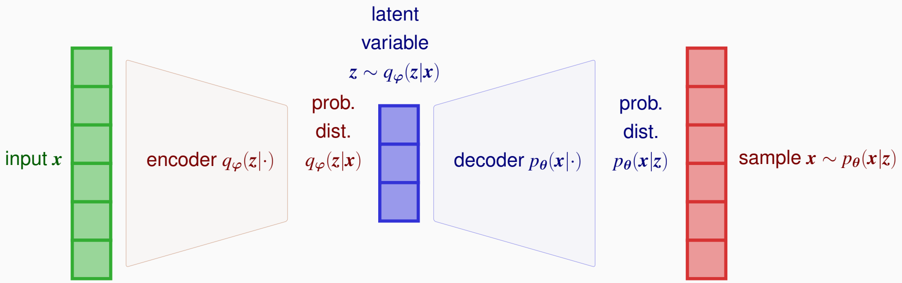
latent variable $z \sim q_\varphi(z|\boldsymbol{x})$
input $\boldsymbol{x}$
encoder $q_\varphi(z|\cdot)$
prob. dist. $q_\varphi(z|\boldsymbol{x})$
decoder $p_\theta(\boldsymbol{x}|\cdot)$
prob. dist. $p_\theta(\boldsymbol{x}|z)$
sample $\boldsymbol{x} \sim p_\theta(\boldsymbol{x}|z)$

Stochastic encoder:

- The encoder $q_\varphi(z|\boldsymbol{x})$ is understood as a parametric approximation of the true posterior $p_\theta(z|\boldsymbol{x})$.
- 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 $z \sim q_\varphi(z|\boldsymbol{x})$
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 φ) that maximize the log-likelihood

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

  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 $q_{\varphi}(z|\boldsymbol{x})$ 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 $D_{\mathrm{KL}}(P \parallel Q) = 0 \iff P = Q$
- $D_{\mathrm{KL}}(P \parallel Q) \neq D_{\mathrm{KL}}(Q \parallel P)$
- $\lim\limits_{n\to+\infty} D_{\mathrm{KL}}(P_n \parallel Q) = 0$ 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 $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(x)$ 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)

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

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

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

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

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.

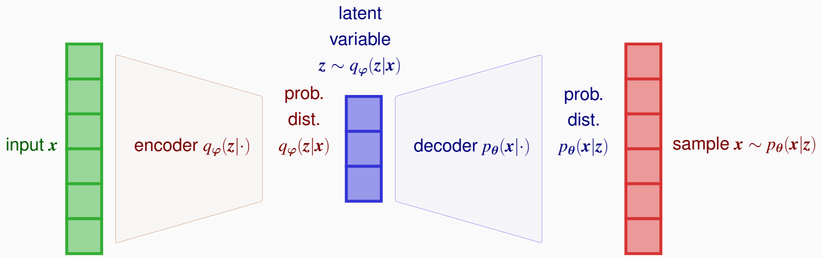
VAE training

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
$$\frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\varphi}}(\boldsymbol{x}^{(i)}) = \frac{1}{N}\sum_{i=1}^{N}\left(\mathbb{E}_{z\sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}, \boldsymbol{z})\right] - \mathbb{E}_{z\sim q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}^{(i)})}\left[\log q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})\right]\right)$$
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:** $q_\varphi(z|\boldsymbol{x}) = \mathcal{N}(z; \boldsymbol{\mu}(\boldsymbol{x}), \operatorname{diag}(\boldsymbol{\sigma}^2(\boldsymbol{x})))$ 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:** $p_\theta(\boldsymbol{x}|z) = \mathcal{N}(z; \boldsymbol{\mu}_{\text{dec}}(z), s^2\boldsymbol{I})$ with $\boldsymbol{\mu}_{\text{dec}}(z) = \mathrm{NN}_\theta(z)$: 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 $z = g(\varepsilon, \boldsymbol{\varphi}, \boldsymbol{x}) = \boldsymbol{\mu}(\boldsymbol{x}) + \boldsymbol{\sigma}(\boldsymbol{x}) \odot \varepsilon$,

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

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:

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

Exercise: Example of Gaussian stochastic encoder

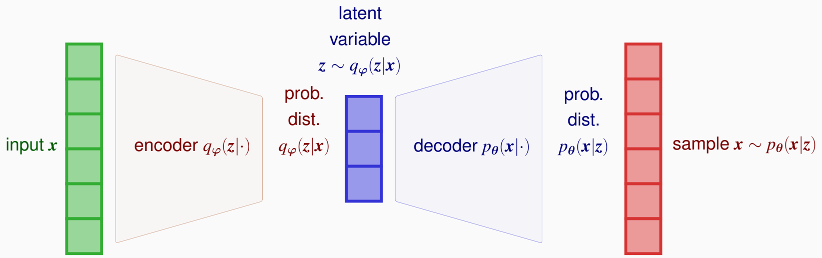
- **Gaussian stochastic encoder:** $q_{\varphi}(z|\boldsymbol{x}) = \mathcal{N}(z; \boldsymbol{\mu}(\boldsymbol{x}), \mathrm{diag}(\boldsymbol{\sigma}^2(\boldsymbol{x})))$ with $(\boldsymbol{\mu}(\boldsymbol{x}), \log \boldsymbol{\sigma}(\boldsymbol{x})) = \mathrm{NN}_{\varphi}(\boldsymbol{x})$.
- **Gaussian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

1. Compute the closed form formula for the KL-divergence:

$$D_{\mathrm{KL}}(q_{\boldsymbol{\varphi}}(z|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(z))$$

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 $z \sim q_\varphi(z|x)$
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 $x \sim p_\theta(x|z)$

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

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

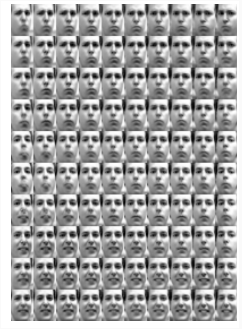
  Then, the likelihood is the binary cross-entropy:

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

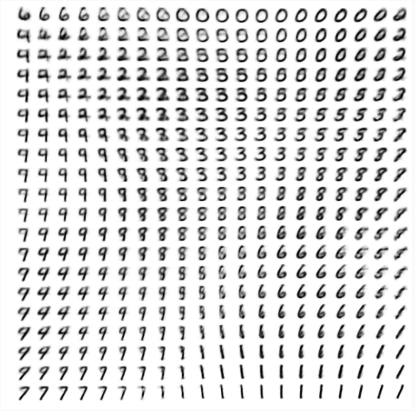
- **Stochastic decoder for discrete data:** Same approach with a NN that outputs a softmax array with the number of classes and cross-entropy...
- Here pixels are supposed independent resulting in noisy samples from $p_\theta(x|z)$... **But one often outputs the expectation for visualization!**

VAE: Results

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



(a) Learned Frey Face manifold



(b) Learned MNIST manifold

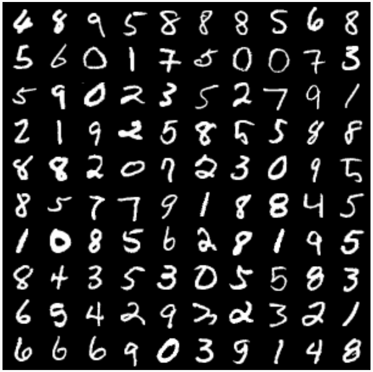
Figure 4: Visualisations of learned data manifold for generative models with two-dimensional latent space, learned with AEVB. Since the prior of the latent space is Gaussian, linearly spaced coordinates on the unit square were transformed through the inverse CDF of the Gaussian to produce values of the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

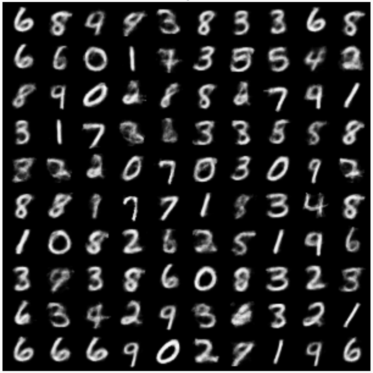
Numerical illustration with a subset of MNIST (1000 images only):

- 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:

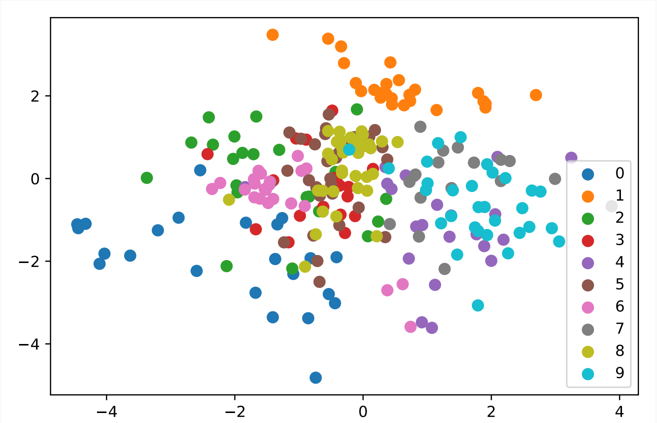


VAE: Results

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:



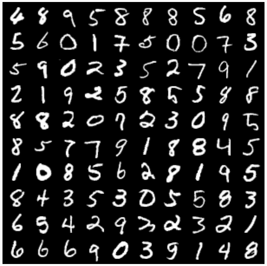
VAE: Results

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

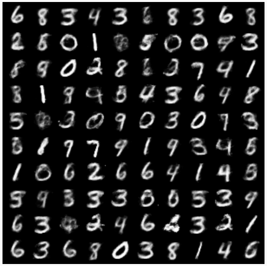
- Encoders and decoders are MLP trained for 1000 epochs.
- The **code dimension is $k = 2$ for visualization** (higher k values would give better results).

AE VS VAE

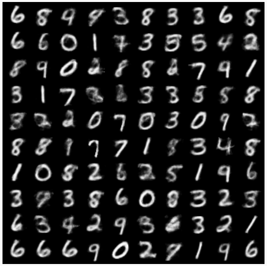
Input:



AE Output:



VAE Output:

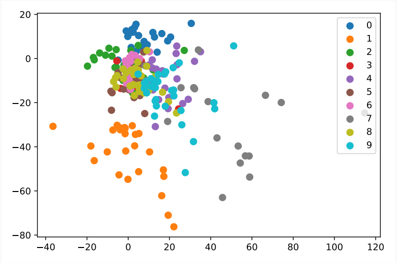


VAE: Results

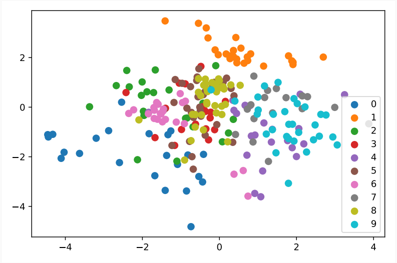
Numerical illustration with a subset of MNIST (1000 images only):

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

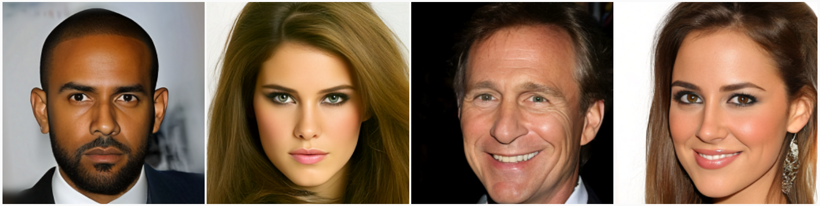


Figure 1: 256×256-pixel samples generated by NVAE, trained on CelebA HQ [28].

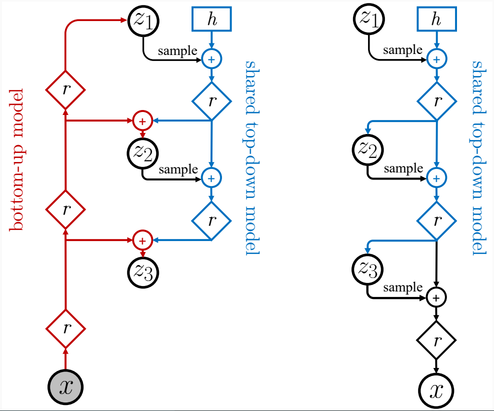
See also Very Deep VAE (Child, 2021).

NVAE: Architecture details

$$\mathcal{L}_{\text{VAE}}(\boldsymbol{x}) := \mathbb{E}_{q(z|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|z)\right] - D_{\text{KL}}(q(z_1|\boldsymbol{x})|p(z_1)) - \sum_{l=2}^{L} \mathbb{E}_{q(z_{<l}|\boldsymbol{x})}\left[D_{\text{KL}}(q(z_l|\boldsymbol{x}, z_{<l})|p(z_l|z_{<l})\right.$$

where $q(z_{<l}|\boldsymbol{x}) = \prod_{i=1}^{l-1} q(z_i|\boldsymbol{x}, z_{<i})$ is the approximate 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.



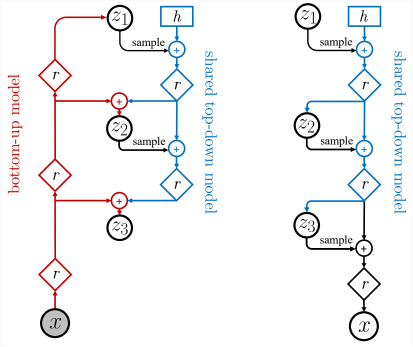
NVAE: Architecture details

- 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 that parameterizes $q(z|\boldsymbol{x})$ relative to $p(z)$:

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

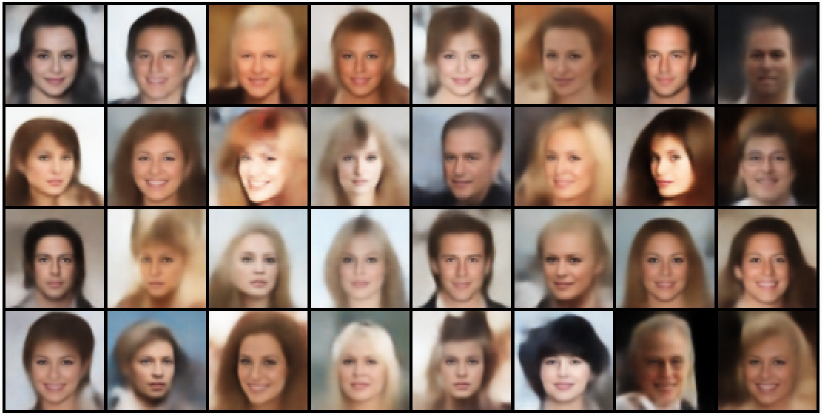
where $\Delta\mu(z_{<l},\boldsymbol{x})$ and $\Delta\boldsymbol{\sigma}^2(z_{<l},\boldsymbol{x})$ are the relative location and scale of the approximate posterior with respect to the prior.

- $\Delta\boldsymbol{\mu}(z_{<l},\boldsymbol{x})$ 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

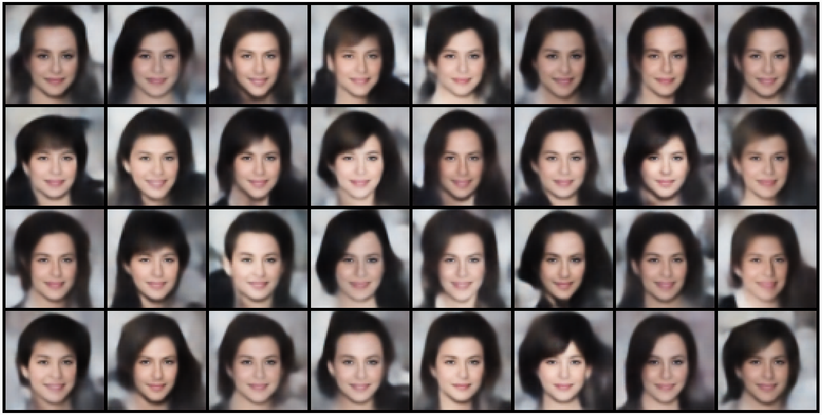
Samples of 64$\times$64 portraits



Sample all levels

NVAE: Results of toy implementation

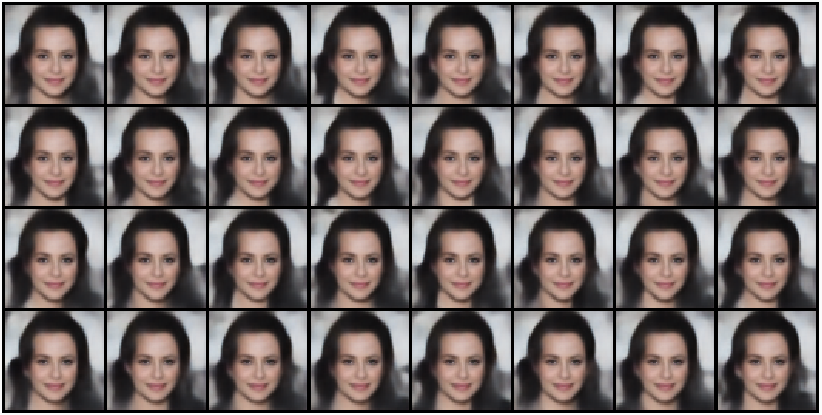
Samples of 64×64 portraits



Fixed z_1

NVAE: Results of toy implementation

Samples of 64×64 portraits



Fixed z_1 and z_2

VAE in practice

Practice session based on this hierarchical architecture (see my MVA course).

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`

Generative Adversarial Networks (GAN)

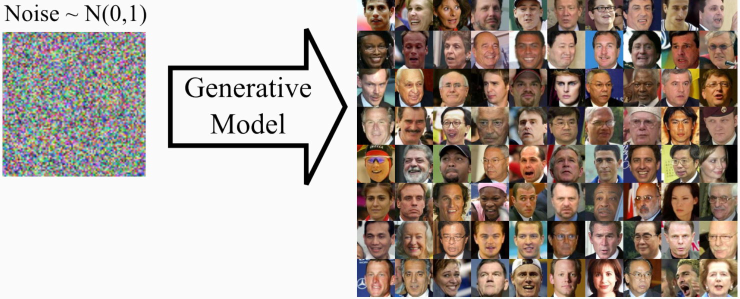
Generative Adversarial Networks (GAN)

Main references:

1. Original paper: (Goodfellow et al., 2014)
2. NIPS 2016 tutorial: (Goodfellow, 2017)

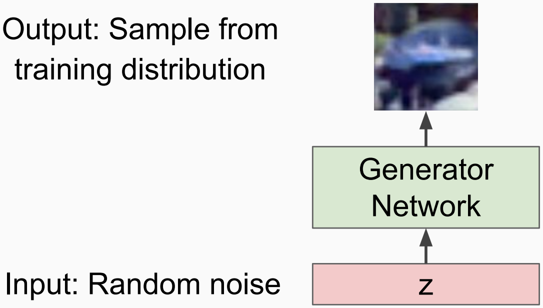
Generative Adversarial Networks (GAN)

Image generation – Beyond Gaussian models



Generative Adversarial Networks (GAN)

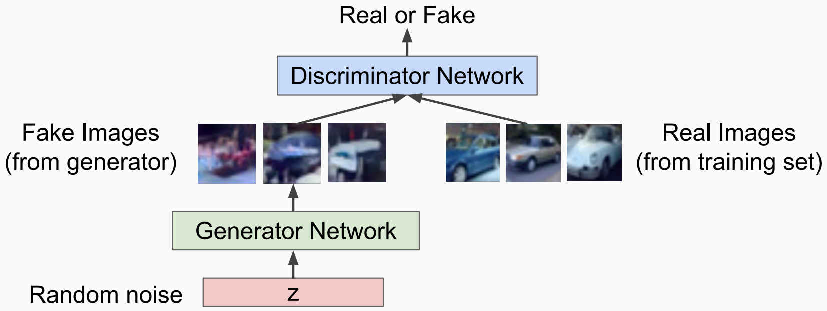
- **Goal:** design a complex model with high capacity able to map latent random noise vectors $z \in \mathbb{R}^k$ to a realistic image $\boldsymbol{x} \in \mathbb{R}^d$.
- **Idea:** Take a deep neural network



- **What about the loss?** Measure if the generated image is **photo-realistic**.

Generative Adversarial Networks (GAN)

Define a loss measuring how much you can fool a classifier that has learned to distinguish between real and fake images.



- **Discriminator network:** Try to distinguish between real and fake images.
- **Generator network:** Fool the discriminator by generating realistic images.

Recap on binary classification

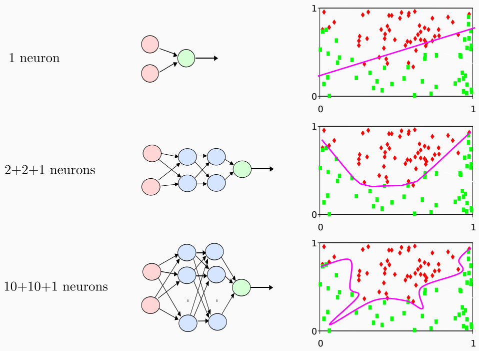
- Given a labeled dataset

$$\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)}),\ i = 1, \dots, N\} \subset \mathbb{R}^d \times \{0, 1\}$$

with **binary labels** $y^{(i)} \in \{0, 1\}$ that corresponds to two classes C_0 and C_1.

- A **parametric classifier** $f_{\boldsymbol{\theta}} : \mathbb{R}^d \to [0, 1]$ outputs a probability such that

$$\boldsymbol{p} = f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathbb{P}(\boldsymbol{x} \in C_1) \quad \text{and} \quad 1 - \boldsymbol{p} = 1 - f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathbb{P}(\boldsymbol{x} \in C_0)$$



Estimated decision regions:
$\hat{C}_1 = \{\boldsymbol{x} \in \mathbb{R}^d,\ f_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq \frac{1}{2}\}$ and
$\hat{C}_0 = \mathbb{R}^d \setminus \hat{C}_1$.

Complexity/capacity of the network
$\Rightarrow$
Trade-off between generalization and overfitting.

Recap on binary classification

- **Training:** Logistic regression for binary classification: Maximum likelihood of the dataset (opposite of binary crosss-entropy : `BCELoss` in PyTorch):

$$\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \left[y^{(i)} \log f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \log \left(1 - f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \right) \right]$$

- For neural networks, the probability $f_{\boldsymbol{\theta}}$ is obtained using the **sigmoid function** $\sigma(t) = \frac{e^t}{1+e^t} = \frac{1}{1+e^{-t}}$ as the activation function of the last layer.
- Beware that $y^{(i)} = 0$ or 1 so only one term is non-zero.
- One could instead regroup the terms of the sum according to the label values:

$$\max_{\boldsymbol{\theta}} \sum_{\substack{(\boldsymbol{x}^{(i)}, y^{(i)}) \in \mathcal{D} \\ \text{s.t. } y^{(i)}=1}}^{N} \log f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + \sum_{\substack{(\boldsymbol{x}^{(i)}, y^{(i)}) \in \mathcal{D} \\ \text{s.t. } y^{(i)}=0}}^{N} \log \left(1 - f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \right)$$

Generative Adversarial Networks (GAN)

- **Discriminator network:** Consider two sets
  - $\mathcal{D}_{\text{real}}$: a dataset of n real images **(real = labeled with $y^{(i)} = 1$),**
  - $\mathcal{D}_{\text{fake}}$: a dataset of m fake images $\boldsymbol{x} = G_{\theta_g}(z)$ **(fake = labeled with $y^{(i)} = 0$).**

- **Goal:** Find the parameters θ_d of a binary classification network $\boldsymbol{x} \mapsto D_{\theta_d}(\boldsymbol{x})$ meant to classify real and fake images.
  Minimize the binary cross-entropy, or maximize its negation

$$\max_{\theta_d} \underbrace{\sum_{\boldsymbol{x}_{\text{real}} \in \mathcal{D}_{\text{real}}} \log D_{\theta_d}(\boldsymbol{x}_{\text{real}})}_{\text{force predicted labels to be 1 for real images}} + \underbrace{\sum_{\boldsymbol{x}_{\text{fake}} \in \mathcal{D}_{\text{fake}}} \log(1 - D_{\theta_d}(\boldsymbol{x}_{\text{fake}}))}_{\text{force predicted labels to be 0 for fake images}}$$

- **How:** Use gradient ascent (Adam).

Generative Adversarial Networks (GAN)

- **Generator network:** Consider a given discriminative model $\boldsymbol{x} \mapsto D_{\theta_d}(\boldsymbol{x})$ and consider $\mathcal{D}_{\text{rand}}$ a set of m random latent vectors.
- **Goal:** Find the parameters θ_g of a network $z \mapsto G_{\theta_g}(z)$ generating images from random vectors z such that it fools the discriminator

$$\min_{\theta_g} \quad \underbrace{\sum_{z \in \mathcal{D}_{\text{rand}}} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))}_{\substack{\text{force the discriminator to think that} \\ \text{our generated fake images are not fake (away from 0)}}} \tag{1}$$

or alternatively (works better in practice)

$$\max_{\theta_g} \quad \underbrace{\sum_{z \in \mathcal{D}_{\text{rand}}} \log D_{\theta_d}(G_{\theta_g}(z)))}_{\substack{\text{force the discriminator to think that} \\ \text{our generated fake images are real (close to 1)}}} \tag{2}$$

- **How:** Gradient descent for (1) or gradient ascent for (2) (Adam)

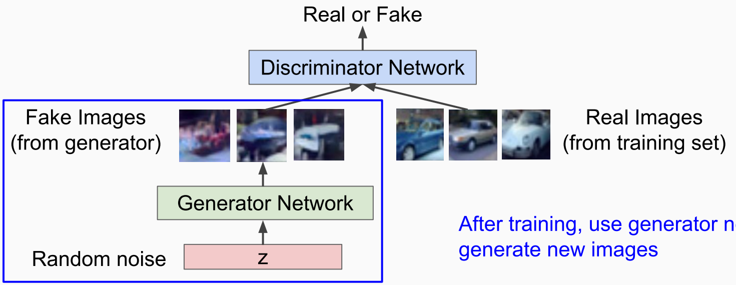
Generative Adversarial Networks (GAN)

- **Train both networks jointly**.
- Minimax loss in a two player game (each player is a network):

$$\min_{\theta_g} \max_{\theta_d} \sum_{\boldsymbol{x} \in \mathcal{D}_{\text{real}}} \log D_{\theta_d}(\boldsymbol{x}) + \sum_{\boldsymbol{z} \in \mathcal{D}_{\text{rand}}} \log(1 - D_{\theta_d}(\underbrace{G_{\theta_g}(\boldsymbol{z})}_{\text{fake}}))$$

- **Training algorithm:** Repeat until convergence
  1. Fix θ_g, update θ_d with one step of gradient ascent,
  2. Fix θ_d, update θ_g with one step of gradient descent for (1),
     (or one step of gradient ascent for (2).)

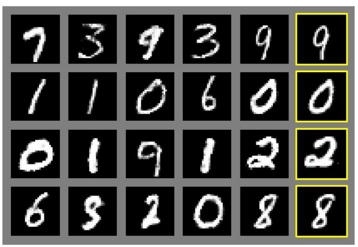
Generative Adversarial Networks (GAN)

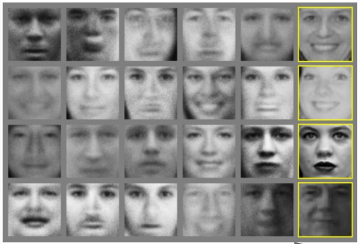


After training, use generator network to generate new images

Generative Adversarial Networks (GAN)

Generated samples

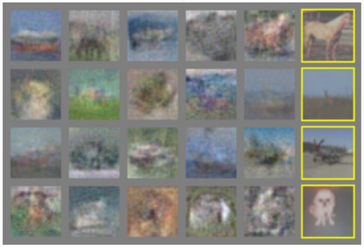


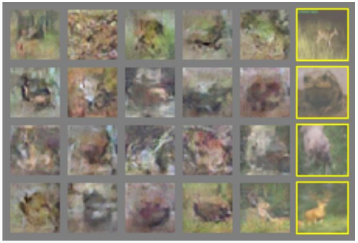


Nearest neighbor from training set

Generative Adversarial Networks (GAN)

Generated samples (CIFAR-10)





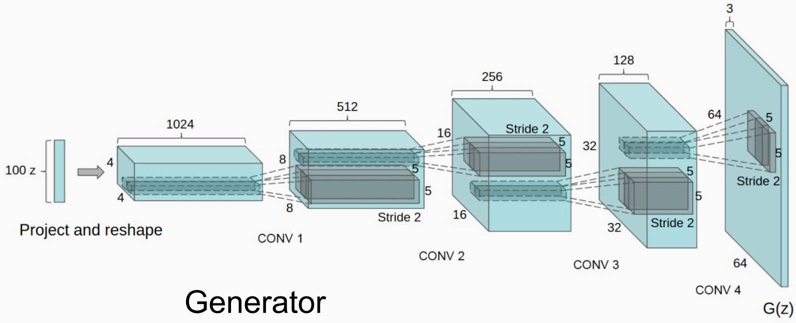
Nearest neighbor from training set

Generative Adversarial Networks (GAN)

Convolutional GAN

(Radford et al., 2016)

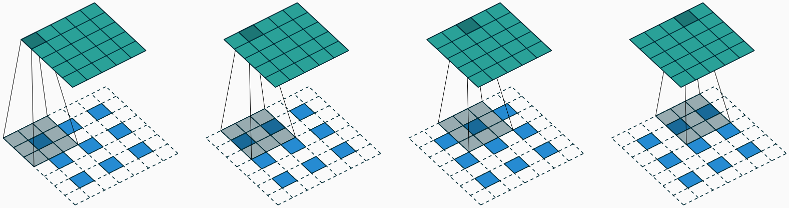
- **Generator:** upsampling network with **fractionally strided convolutions** (i.e. the transpose operator of convolution+subsampling , called `ConvTranspose2d` in PyTorch),
- **Discriminator:** convolutional network with strided convolutions.



Transposed convolution arithmetic

Fractionally strided convolutions:

- This is the transpose operator of convolution+subsampling (convolution with stride).
- Called `ConvTranspose2d` in PyTorch



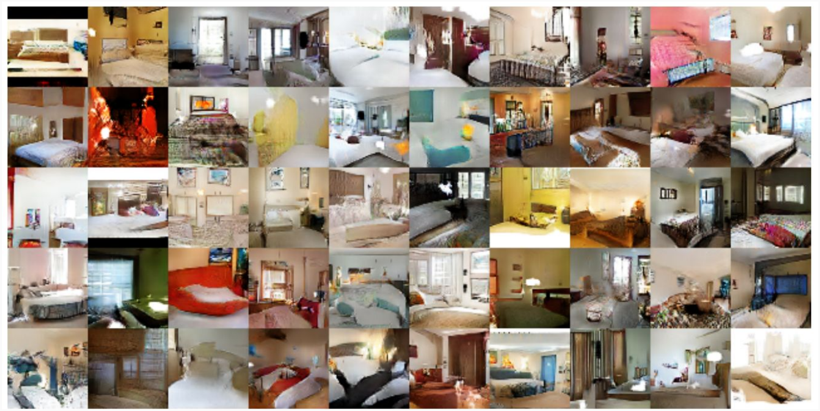
The transpose of convolving a 3×3 kernel over a 5×5 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 1$). It is equivalent to convolving a 3×3 kernel over a 3×3 input (with 1 zero inserted between inputs) padded with a 1×1 border of zeros using unit strides (i.e., $i' = 3$, $\tilde{i}' = 5$, $k' = k$, $s' = 1$ and $p' = 1$).

(source: From (Dumoulin and Visin, 2016))

Generative Adversarial Networks (GAN)

Convolutional GAN

(Radford et al., 2016)

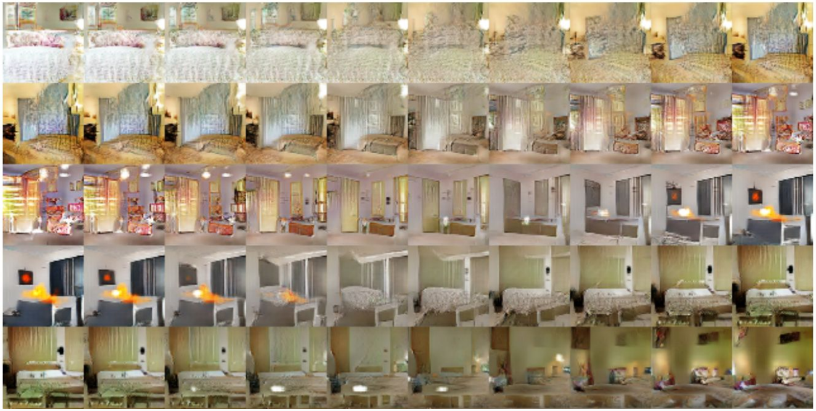


Generations of realistic bedrooms pictures, from randomly generated latent variables.

Generative Adversarial Networks (GAN)

Convolutional GAN

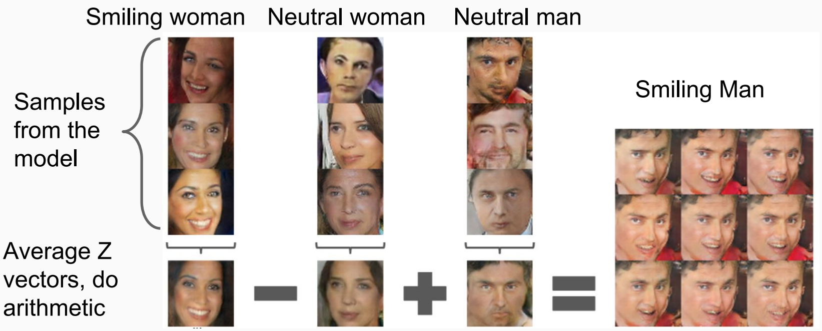
(Radford et al., 2016)



Interpolation in between points in latent space.

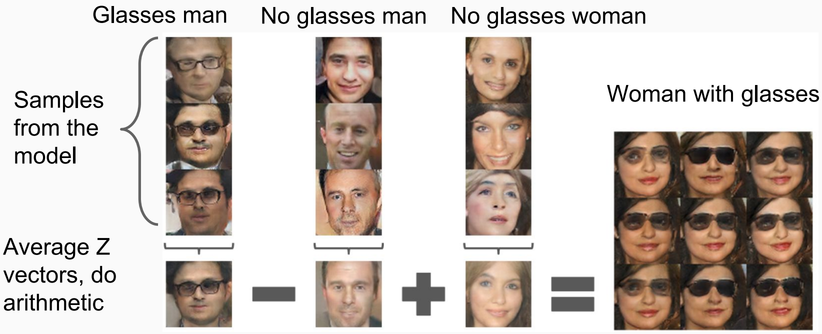
Convolutional GAN – Arithmetic

(Radford et al., 2016)



Convolutional GAN – Arithmetic

(Radford et al., 2016)



Generative Adversarial Networks (GAN)

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

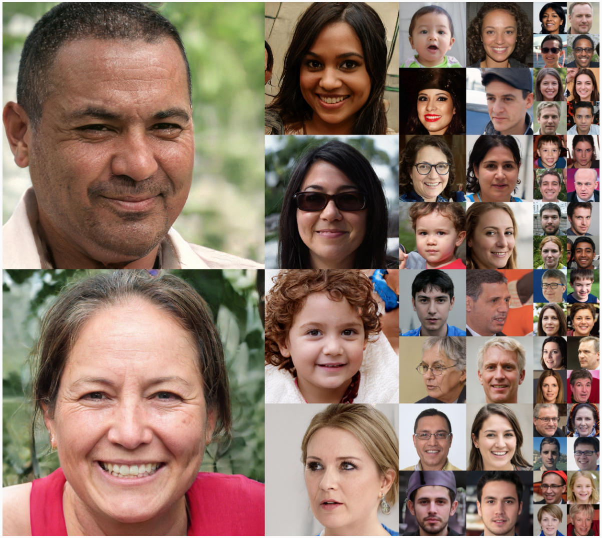


Image size:
1024× 1024 px
(source: Karras et al.)

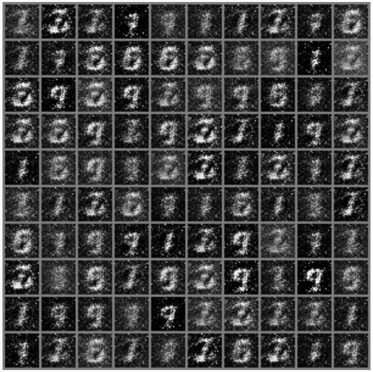
GAN Training

Training with MNIST (60 000 images)

- Adam optimizer
- Learning rate 0.0002 for both the discriminator and the generator

Real images:

Fake images, epoch 1:



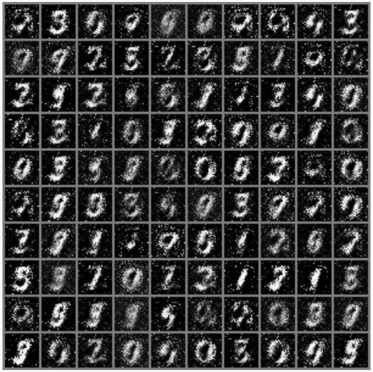
GAN Training

Training with MNIST (60 000 images)

- Adam optimizer
- Learning rate 0.0002 for both the discriminator and the generator

Real images:

Fake images, epoch 2:



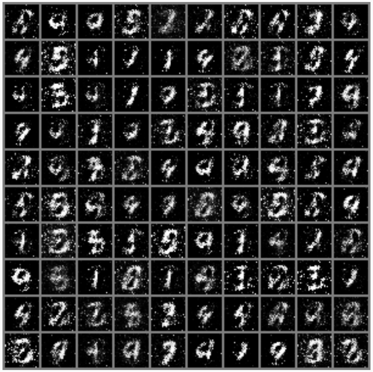
GAN Training

Training with MNIST (60 000 images)

- Adam optimizer
- Learning rate 0.0002 for both the discriminator and the generator

Real images:

Fake images, epoch 3:



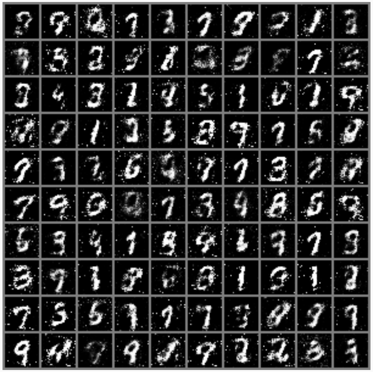
GAN Training

Training with MNIST (60 000 images)

- Adam optimizer
- Learning rate 0.0002 for both the discriminator and the generator

Real images:

Fake images, epoch 10:



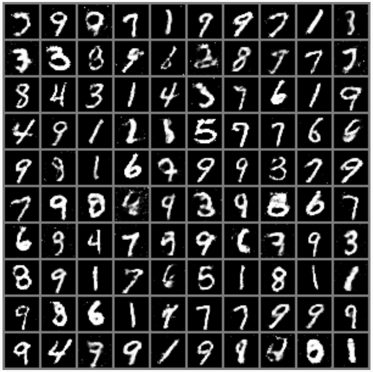
GAN Training

Training with MNIST (60 000 images)

- Adam optimizer
- Learning rate 0.0002 for both the discriminator and the generator

Real images:

Fake images, epoch 100:



GAN Training

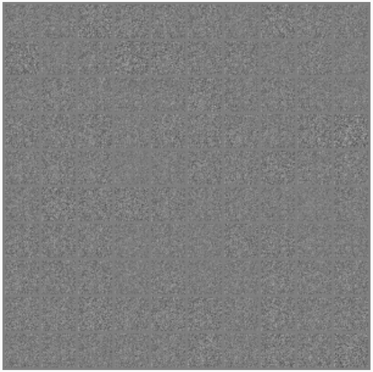
Training GANs is quite unstable!

The generator can suffer *mode collapse*: It always produces the same image (one mode only).

Same as before **but with SGD instead of Adam**.

Real images:

Fake images, epoch 1:



GAN Training

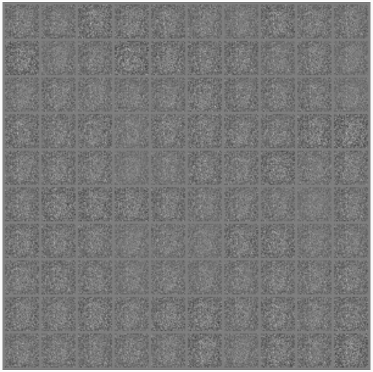
Training GANs is quite unstable!

The generator can suffer *mode collapse*: It always produces the same image (one mode only).

Same as before **but with SGD instead of Adam**.

Real images:

Fake images, epoch 2:



GAN Training

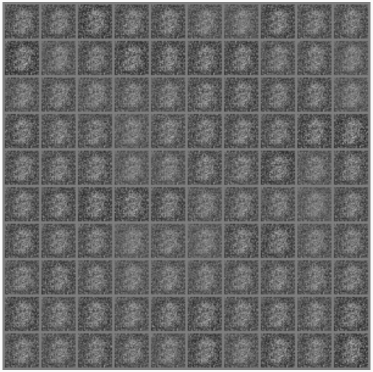
Training GANs is quite unstable!

The generator can suffer *mode collapse*: It always produces the same image (one mode only).

Same as before **but with SGD instead of Adam**.

Real images:

Fake images, epoch 3:



GAN Training

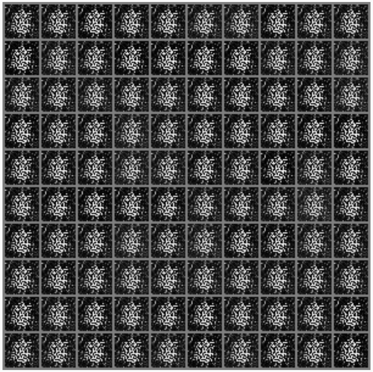
Training GANs is quite unstable!

The generator can suffer *mode collapse*: It always produces the same image (one mode only).

Same as before **but with SGD instead of Adam**.

Real images:

Fake images, epoch 10:



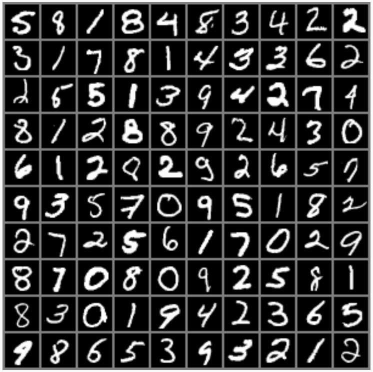
GAN Training

Training GANs is quite unstable!

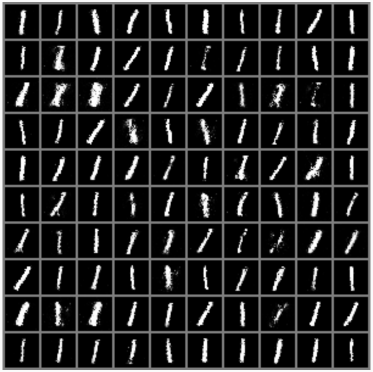
The generator can suffer *mode collapse*: It always produces the same image (one mode only).

Same as before **but with SGD instead of Adam**.

Real images:



Fake images, epoch 100:



GAN Training

Some heuristics inspired by optimal transport theory have been proposed and called **Wasserstein GAN** (Arjovsky et al., 2017) (Gulrajani et al., 2017).

Wasserstein GAN

Optimal transport

- The optimal transport theory provides mathematical tools to compare or interpolate between probability distributions.
- Given two probability distributions μ_0 and μ_1 in $\mathcal{P}_2(\mathbb{R}^d)$ (the set of probability measures with finite second moments on $\mathbb{R}^d$), and a positive cost function $c : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$

$$MK_c(\mu_0, \mu_1) := \inf_{\gamma \in \Pi(\mu_0,\mu_1)} \int_{\mathbb{R}^d \times \mathbb{R}^d} c(y_0, y_1) d\gamma(y_0, y_1),$$

where $\Pi(\mu_0, \mu_1)$ is the set of probability distributions γ on $\mathbb{R}^d \times \mathbb{R}^d$ with marginal distributions μ_0 and μ_1.

Proposition (Duality)

$$MK_c(\mu_0, \mu_1) = \sup_{\phi, \psi \in \Phi_c(\mu_0,\mu_1)} \int \phi d\mu_0 + \int \psi d\mu_1,$$

where

$$\Phi_c(\mu_0, \mu_1) = \{\phi, \psi \in C_b(\mathbb{R}^d) \textit{ s.t. } \forall x, y,\ \phi(x) + \psi(y) \leq c(x, y)\}.$$

Optimal transport

Wasserstein distances: When using $c(x,y) = \|x-y\|^p$ one defines Wasserstein distances:

Definition
The p-*Wasserstein distance* W_p between μ_0 and μ_1 is defined as

$$W_p^p(\mu_0,\mu_1) := \inf_{Y_0\sim\mu_0; Y_1\sim\mu_1} \mathbb{E}\left(\|Y_0 - Y_1\|^p\right) = \inf_{\gamma\in\Pi(\mu_0,\mu_1)} \int_{\mathbb{R}^d\times\mathbb{R}^d} \|y_0 - y_1\|^p d\gamma(y_0,y_1).$$

1-Wasserstein distance and duality: See eg (Santambrogio, 2015)

For $p=1$, one has

$$W_1(\mu_0,\mu_1) = \sup_{\phi\in\mathrm{Lip}_1} \int \phi d\mu_0 - \int \phi d\mu_1 = \sup_{\phi\in\mathrm{Lip}_1} \mathbb{E}_{\boldsymbol{x}\sim\mu_0}(\phi(\boldsymbol{x})) - \mathbb{E}_{\boldsymbol{x}\sim\mu_1}(\phi(\boldsymbol{x}))$$

where

$$\mathrm{Lip}_1 = \{f : \mathbb{R}^d \to \mathbb{R},\ \text{s.t. } \forall x,y,\ |f(x)-f(y)| \leq \|x-y\|\}.$$

Optimal transport and GANs

Back to GANs:

- The role of the discriminator D is to differentiate the distribution $p_{\text{real}}(\boldsymbol{x})$ of real images from the distribution $p_{\text{gen}}(\boldsymbol{x})$ of generated images.
- Ideally, one would like to optimize the generator to minimize $W_1(p_{\text{real}}, p_{\text{gen}})$.
- However it is not possible to compute this Wasserstein distance W_1 because taking the $\sup$ over Lip_1 is not tractable.
- (Arjovsky et al., 2017) proposes to restrict Lip_1 to the set of Lip_1 functions that are parameterized with some neural network:

$$\begin{aligned} W_1(p_{\text{real}}, p_{\text{gen}}) &= \sup_{\phi \in \mathrm{Lip}_1} \mathbb{E}_{\boldsymbol{x}\sim p_{\text{real}}}(\phi(\boldsymbol{x})) - \mathbb{E}_{\boldsymbol{x}\sim p_{\text{gen}}}(\phi(\boldsymbol{x})) \\ &\geq \sup_{D_{\theta_d} \in \mathrm{Lip}_1} \mathbb{E}_{\boldsymbol{x}\sim p_{\text{real}}}(D_{\theta_d}(\boldsymbol{x})) - \mathbb{E}_{\boldsymbol{x}\sim p_{\text{gen}}}(D_{\theta_d}(\boldsymbol{x})) \end{aligned}$$

Wassertsein GAN

GAN (Vanilla):

$$\min_{\theta_g} \max_{\theta_d} \sum_{\boldsymbol{x} \in \mathcal{D}_{\text{real}}} \log D_{\theta_d}(\boldsymbol{x}) + \sum_{z \in \mathcal{D}_{\text{rand}}} \log(1 - D_{\theta_d}(G_{\theta_g}(\boldsymbol{z}))$$

Wassestein GAN:

$$\min_{\theta_g} \max_{\substack{\theta_d \text{ s.t.} \\ D_{\theta_d} \in \mathrm{Lip}_1}} \sum_{\boldsymbol{x} \in \mathcal{D}_{\text{real}}} D_{\theta_d}(\boldsymbol{x}) - \sum_{z \in \mathcal{D}_{\text{rand}}} D_{\theta_d}(G_{\theta_g}(\boldsymbol{z}))$$

- We just got rid of the $\log$ and $D_{\theta_d}(\boldsymbol{x})$ is not a probability... but we now have a constrained optimization "$D_{\theta_d} \in \mathrm{Lip}_1$".
- The original WGAN paper (Arjovsky et al., 2017) uses weight clipping to restrict the Lipschitz constant (heuristic).

Wassertsein GAN: Gradient Penalty

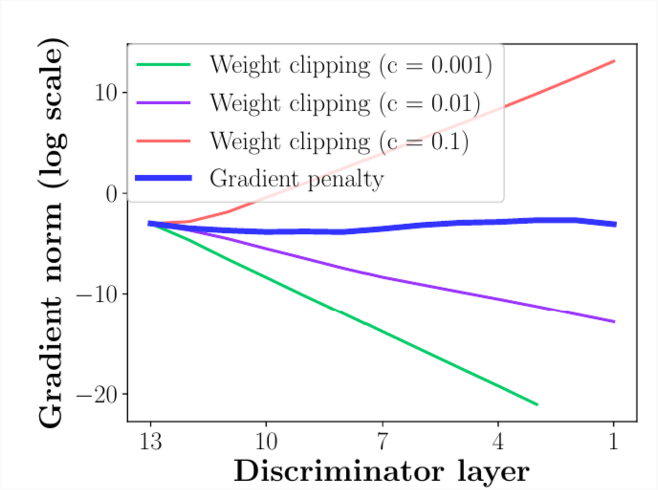
- It is known that optimal potential ϕ satisfies $\|\nabla_{\boldsymbol{x}}\phi(\boldsymbol{x})\| = 1$ a.e. (Santambrogio, 2015).
- (Gulrajani et al., 2017) propose to use this property by minimizing:

$$\min_{\theta_g} \max_{\theta_d} \sum_{\boldsymbol{x}\in\mathcal{D}_{\text{real}}} D_{\theta_d}(\boldsymbol{x}) - \sum_{z\in\mathcal{D}_{\text{rand}}} D_{\theta_d}(G_{\theta_g}(z)) + \lambda \sum_{\boldsymbol{x}_t} (\|\nabla_{\boldsymbol{x}} D_{\theta_d}(\boldsymbol{x}_t)\| - 1)^2$$

  where each $\boldsymbol{x}_t$ is a point from a segment joining a real and a fake image.
- This training procedure is referred as **WGAN-GP**.
- Note that the gradient is with respect to the image variable $\boldsymbol{x}$ and not the parameters θ_g.

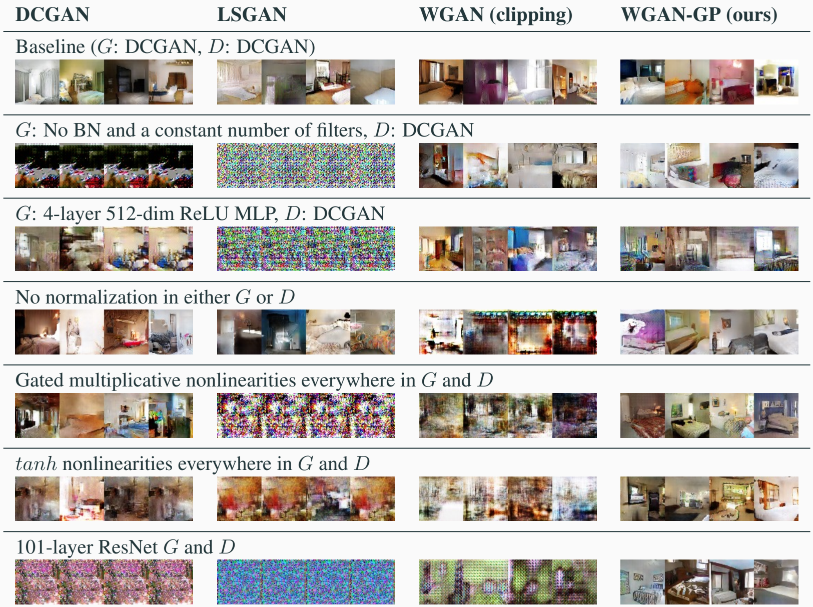
Wassertsein GAN: Gradient Penalty

Gradient Penalty VS. weight clipping:



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

Wassertsein GAN



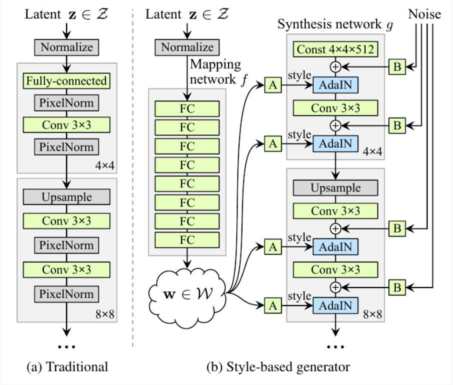
DCGAN	LSGAN	WGAN (clipping)	WGAN-GP (ours)
Baseline (G: DCGAN, D: DCGAN)			
G: No BN and a constant number of filters, D: DCGAN			
G: 4-layer 512-dim ReLU MLP, D: DCGAN			
No normalization in either G or D			
Gated multiplicative nonlinearities everywhere in G and D			
$tanh$ nonlinearities everywhere in G and D			
101-layer ResNet G and D			

Wassertsein GAN

Wassertsein GAN using the gradient penalty is a more stable way to train deep convolutional generators/discriminators.

Style GAN

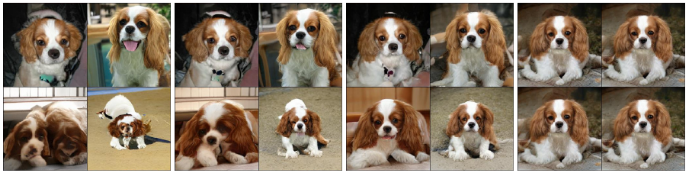
- Style GAN uses the loss of a **WGAN-GP**.
- Main innovation is the **architecture of the generator**.
- Open source.



Style GAN architecture and results from (Karras et al., 2019).

Big GAN

- Another state-of-the-art GAN: **BigGAN** (Brock et al., 2019).
  - Trained with vanilla GAN
  - Large models and large batch size improve the results.
- **Truncation trick:**
  - Train model with standard Gaussian in the latent space.
  - Sample with truncated Gaussian.
  - This improves the quality of results (but reduces the diversity).



- There are still problems with training instablities.

(source: (Brock et al., 2019))

A cautionary tale

- Most of the recent **improvements come from the architecture**.
- It has been reported that Vanilla GAN performs as well as other GANs upon fair comparison (Lucic et al., 2018).

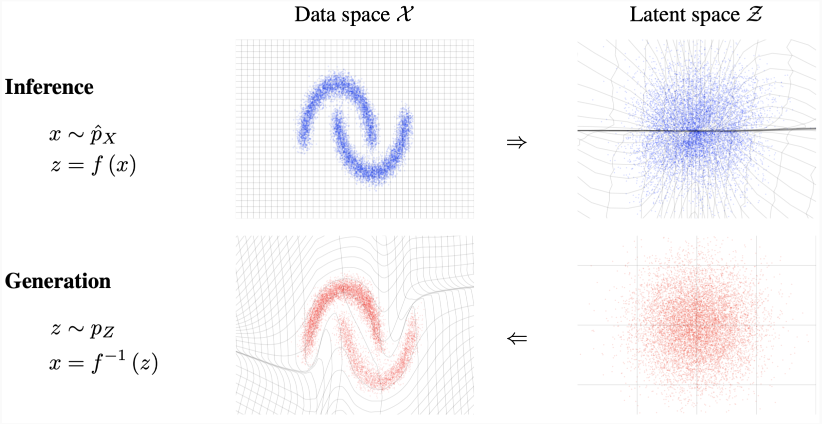
Summary of GANs

- **Advantages:**
    - GANs provide(d) state-of-the-art results
    - They provide interesting latent representations.
    - They allows flexible losses and formulations.
    - They allow for **fast generation**.
- **Problems:**
    - GANs are **very hard to train** (collapse during training).
    - Diversity is a problem (mode collapse).
    - Theoretical analysis is hard.

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 $f \in \mathcal{C}_c(\mathbb{R}^d, \mathbb{R})$

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

where $|J(g_\theta^{-1})(x)| = \left| \det \left(\frac{\partial g_{\theta-1,m}}{\partial x_n}(x) \right)_{1 \le m,n \le d} \right|$ 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**.
- 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 $g_\theta(x) = (x_0, \exp[s_\theta(x_0)] \odot x_1 + t_\theta(x_0))$.
  - **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.

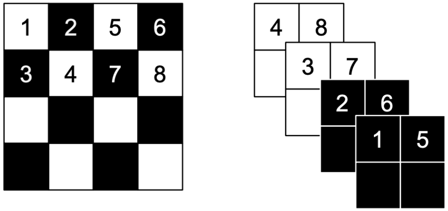
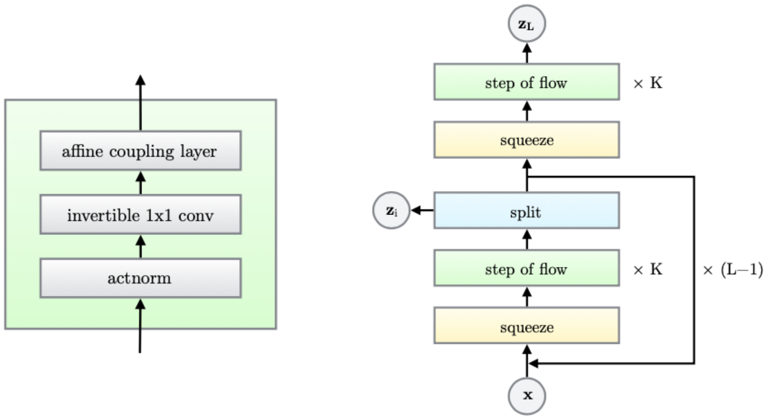


Figure 3: Masking schemes for affine coupling layers. On the left, a spatial checkerboard pattern mask. On the right, a channel-wise masking. The squeezing operation reduces the $4 \times 4 \times 1$ tensor (on the left) into a $2 \times 2 \times 4$ tensor (on the right). Before the squeezing operation,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

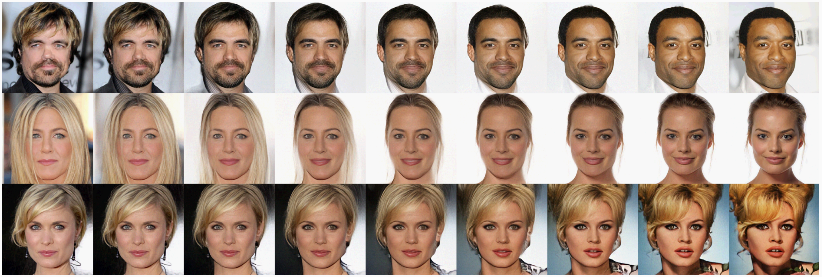
(source: From (Kingma and Dhariwal, 2018))

- 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

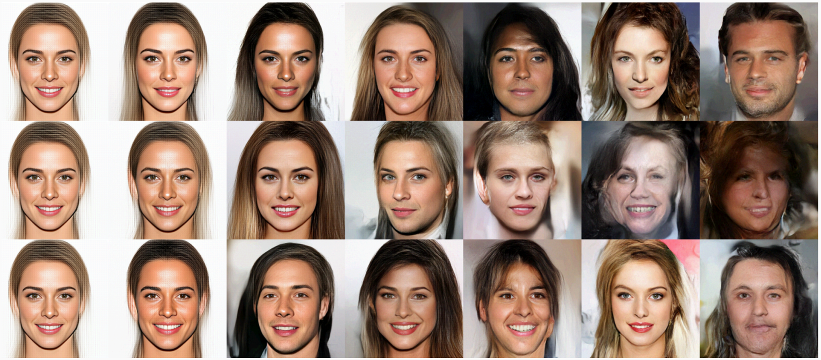
Generative Flow (GLOW) (Kingma and Dhariwal, 2018)



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 (we'll talk about inference for GANs next week).

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 $\sigma < 1$.

References i

References

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR.

Brock, A., Donahue, J., and Simonyan, K. (2019). Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*.

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*.

References ii

Chung, H., Kim, J., Mccann, M. T., Klasky, M. L., and Ye, J. C. (2023). Diffusion posterior sampling for general noisy inverse problems. In *The Eleventh 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.

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *ArXiv e-prints*.

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. (2017). Nips 2016 tutorial: Generative adversarial networks.

References iii

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.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. 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 iv

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 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.

References v

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 TrendsÂ® in Machine Learning*, 12(4):307–392.

Lucic, M., Kurach, K., Michalski, M., Gelly, S., and Bousquet, O. (2018). Are GANs created equal? A large-scale study. 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.

Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

References vi

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.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695.

Ruiz, N., Li, Y., Jampani, V., Pritch, Y., Rubinstein, M., and Aberman, K. (2023). Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 22500–22510.

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., Ghasemipour, S. K. S., Ayan, B. K., Mahdavi, S. S., Lopes, R. G., Salimans, T., Ho, J., Fleet, D. J., and Norouzi, M. (2022). Photorealistic text-to-image diffusion models with deep language understanding.

References vii

Santambrogio, F. (2015). *Optimal Transport for Applied Mathematicians: Calculus of Variations, PDEs, and Modeling*. Birkhäusser.

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.

Song, J., Vahdat, A., Mardani, M., and Kautz, J. (2023). Pseudoinverse-guided diffusion models for inverse problems. In *International Conference on Learning Representations*.

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.

References viii

Zhang, L., Rao, A., and Agrawala, M. (2023). Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3836–3847.