UniGe | MaLGa

# Generative models

**Nicoletta Noceti**
Nicoletta.noceti@unige.it

UniGe | MaLGa

# Introduction

# Let's change the framework

- In supervised settings both input and output are available in our training set

- We now work with datasets for which the output is not known, i.e. in **unsupervised scenarios**

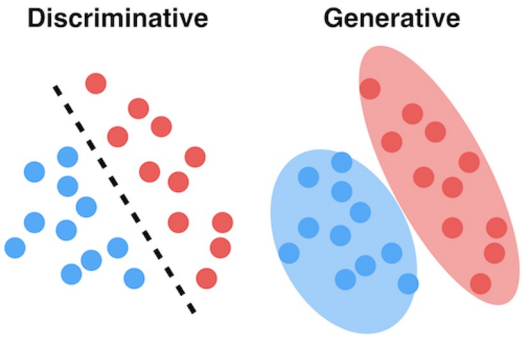
- Examples of applications: clustering, extracting hidden structures in data, retrieving similar data, generating new examples

# Generative modeling

Given a Training set $X$ with the associated labels $Y$:

Discriminative models $p(Y|X)$

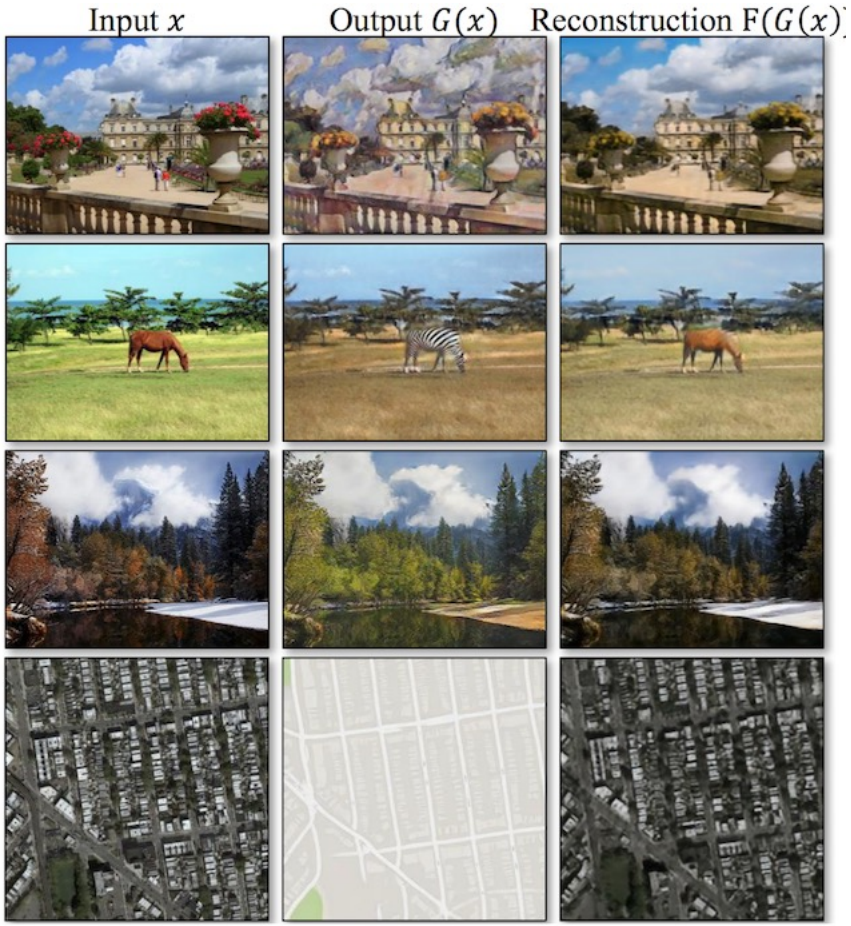
Generative models $p(X)$



# Generative modeling

**Goal**: Take as input unlabeled training samples from some distribution and learn a model that represents that distribution

- density estimation
- learn appropriate representations or embeddings
- generate new data


Input $x$
Output $G(x)$
Reconstruction F$(G(x)$

UniGe | MaLGa

# Autoencoders

# **Autoencoders** *(automatic encoders)*

- **Unsupervised** approach for learning **a lower dimensional feature representation** of an input from unlabelled training data

- It is composed by two parts:

  - An **encoder** function, h = f(x): it describes the lower dimensional code to represent the input

  - A **decoder** function, r = g(h), that produces the approximate reconstruction

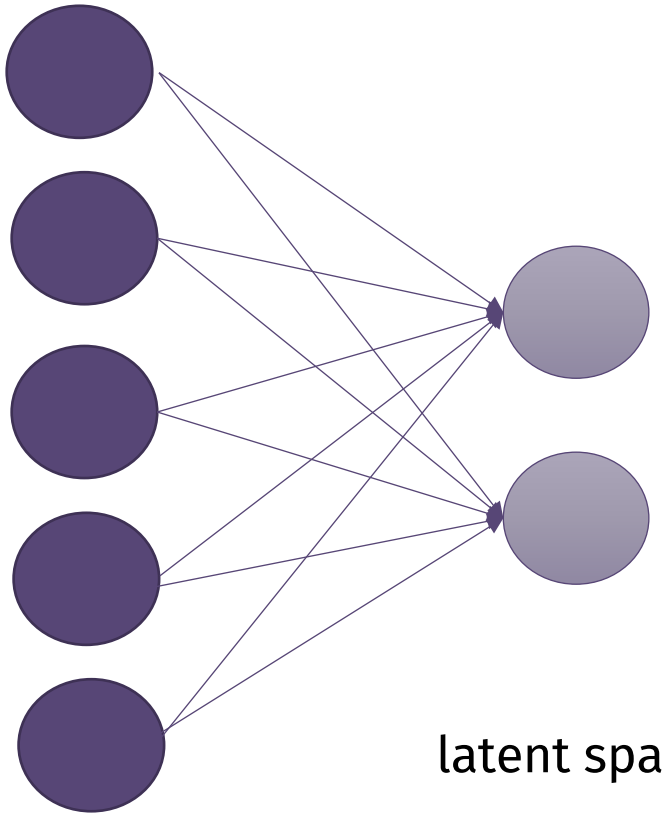
# Autoencoders

- Traditionally (1987... 1994), they were used for **dimensionality reduction** and **feature learning**

- More recently, they have been applied to **generative models**

- They may be thought of as a special case of feedforward networks, and they can be trained using the very same strategies

# Basic autoencoder



latent space h

Input x

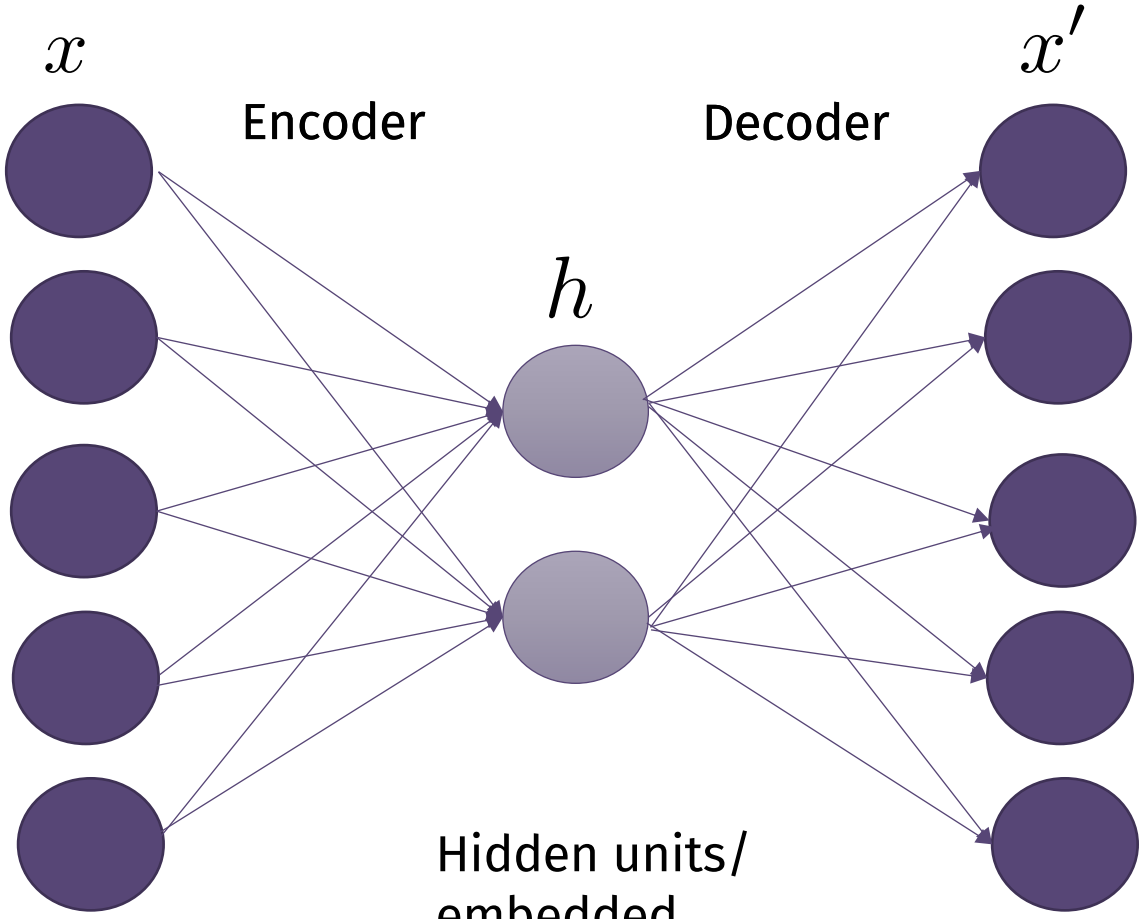
Autoencoders are an **unsupervised** approach for learning a **lower-dimensional** feature representation

The **encoder** learns a mapping from data *x* to a low-dimensional latent space, *h*

*How can we learn the latent space?*

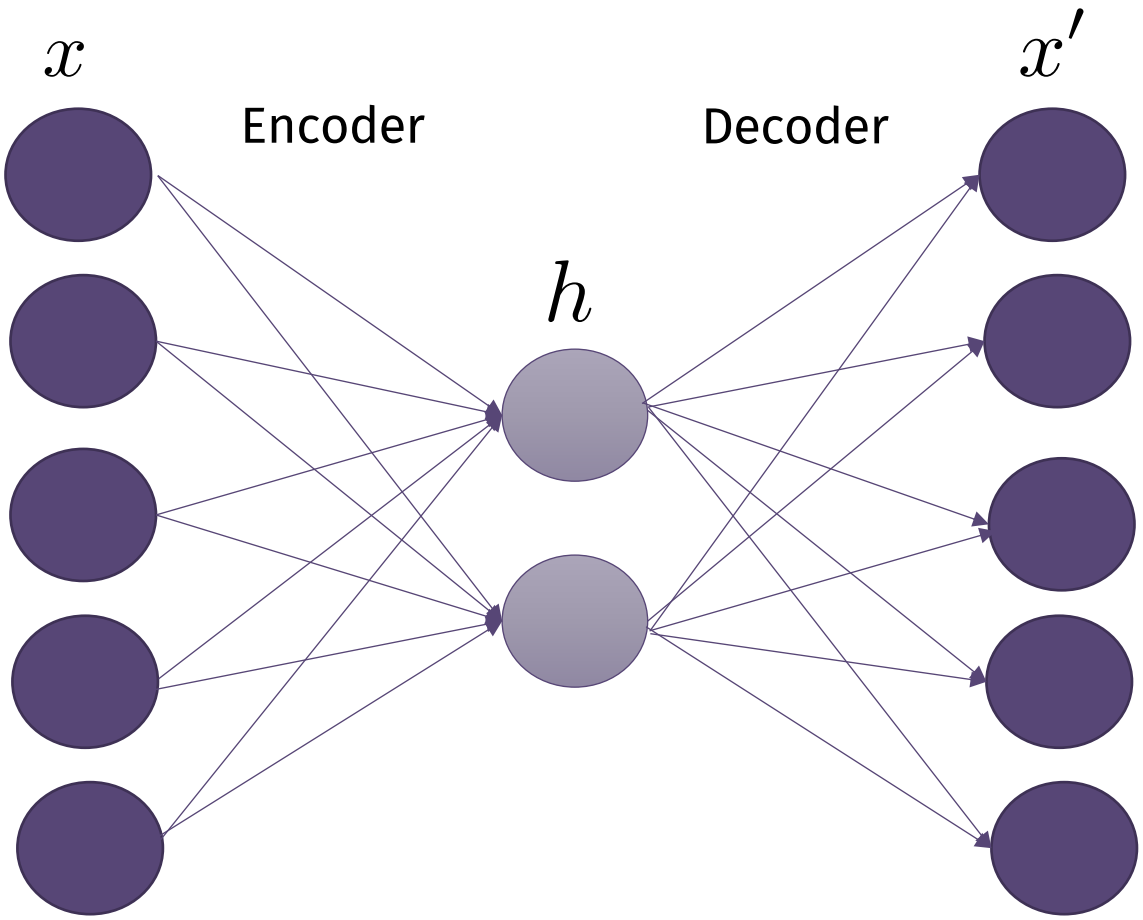
# Basic autoencoder

*How can we learn the latent space?*
Train the model to use it to reconstruct the original data




The **decoder** learns mapping back from latent *h* to a reconstructed observation *x'*

# Basic autoencoder




Encoder

$$h = f(x)$$

Decoder

$$x' = g(h)$$

Loss

$$L(x, g(f(x))$$

Example of a reconstruction loss (notice, no labels!)

$$L(x, x') = ||x - x'||^2$$

# Autoencoders and PCA

If we do not use non-linear activations and use a loss function based on MSE

$$L(x, x') = ||x - x'||_2^2 = \sum_i (x_i - x'_i)^2$$

we obtain something very similar to PCA

A difference is that the latent dimensions will not be necessarily orthogonal and will have (more or less) the same variance

# Undercomplete autoencoder

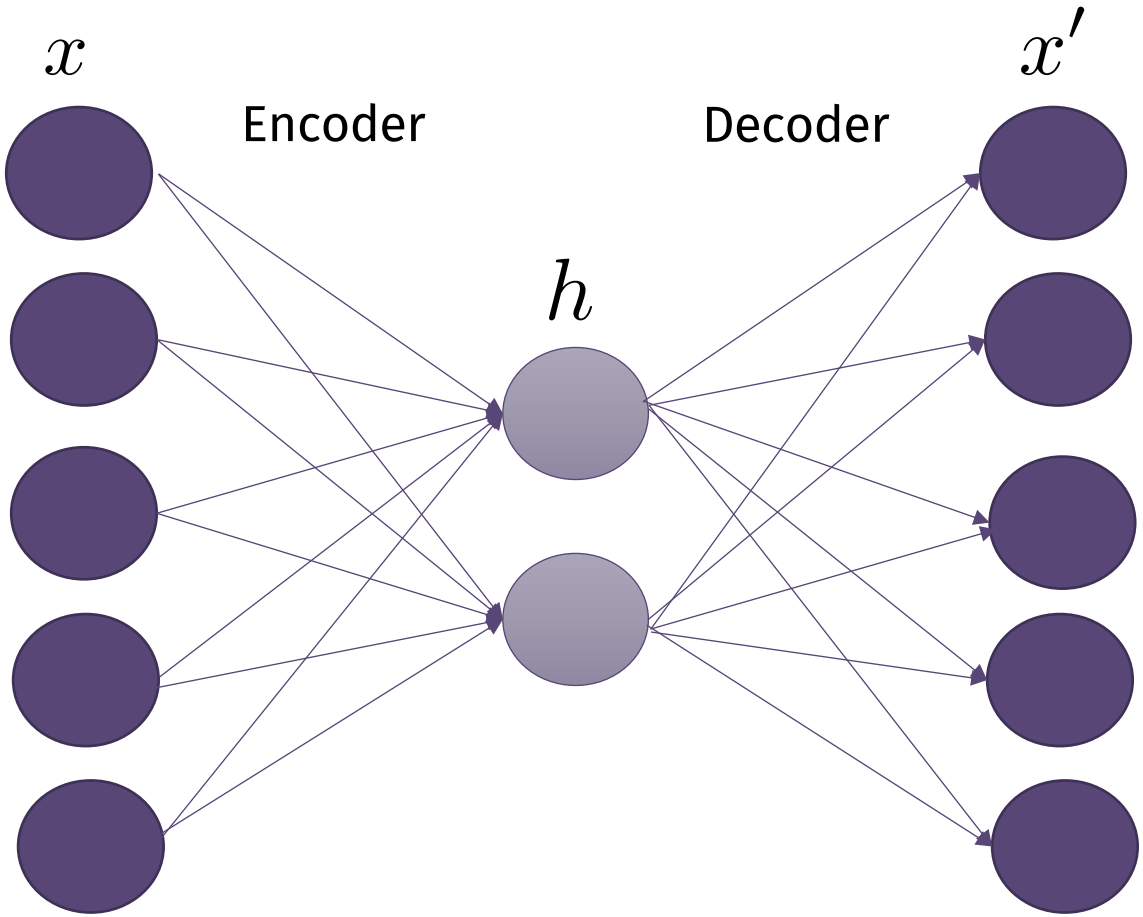
- A further constraint connecting this approach to PCA is to force h to have a smaller dimension than x

- It is commonly known as **undercomplete autoencoder:** learning an undercomplete representation forces the autoencoder **to capture the most salient features**

- Giving too much capacity to the model, it fails to learn anything useful (simple copy...)

# Basic autoencoder

## Beyond deterministic models

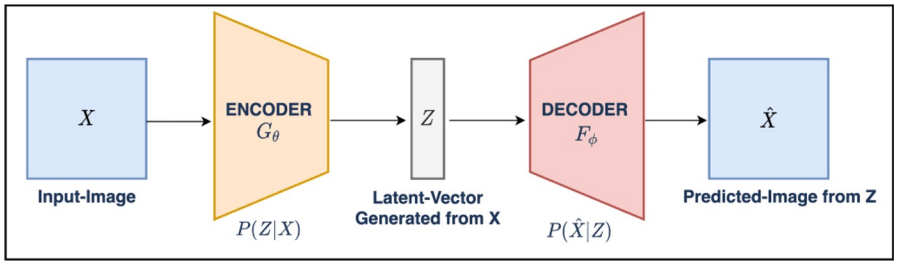



$$p_{encoder}(h|x)$$

$$p_{decoder}(x|h)$$

# Autoencoders

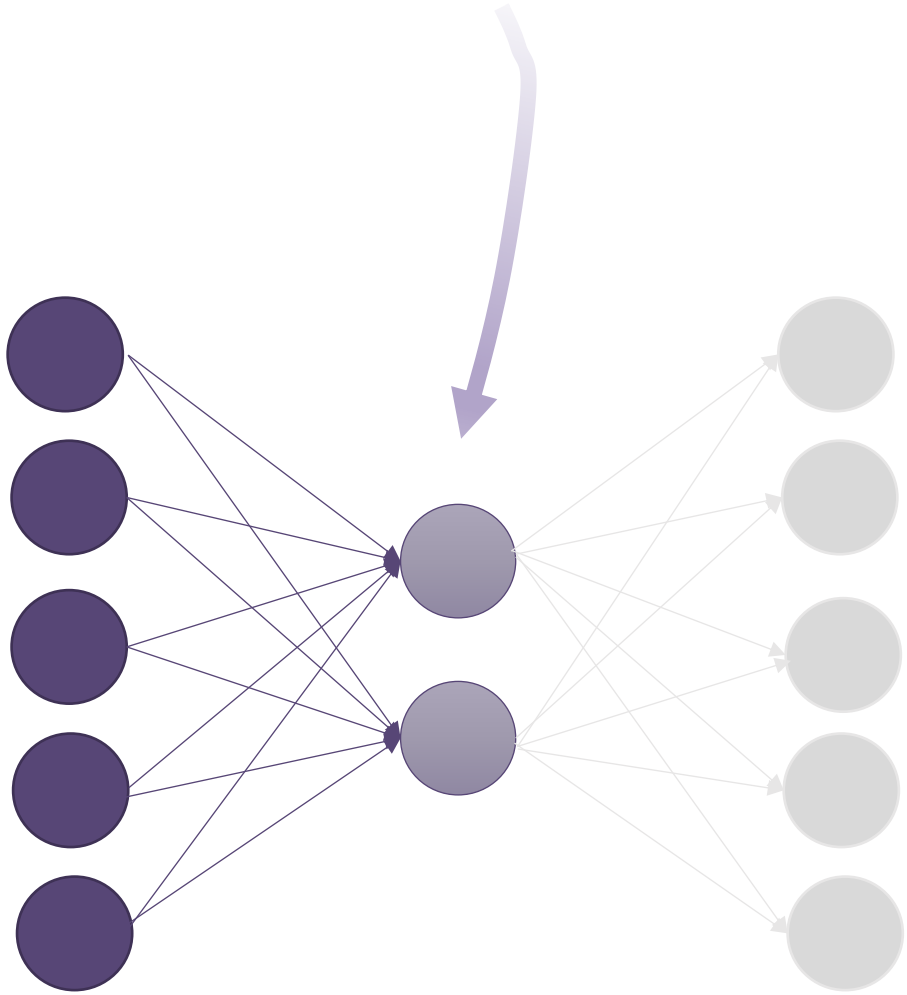
## More in general




$$loss : \underbrace{\mathbb{E}_{P_\phi(Z|X)}[\log P_\theta(\hat{X}|Z)]}_{\text{reconstruction error}}$$

- Each sample is modeled as a point in the latent space
- **No** Regularizer:
  - Close points not necessarily similar once decoded
  - Exist points of the latent space not meaningful

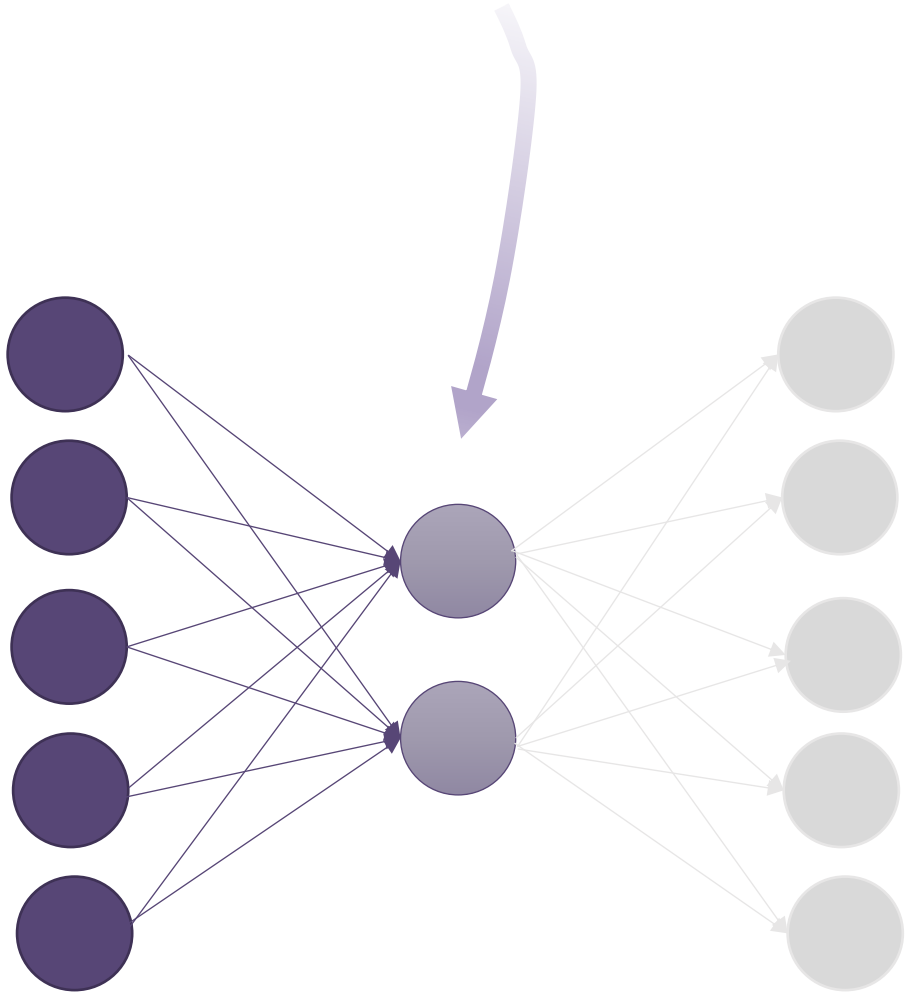
# How can we use the latent space?



- Autoencoders can be seen as an **unsupervised** approach for learning a **lower-dimensional** feature representation

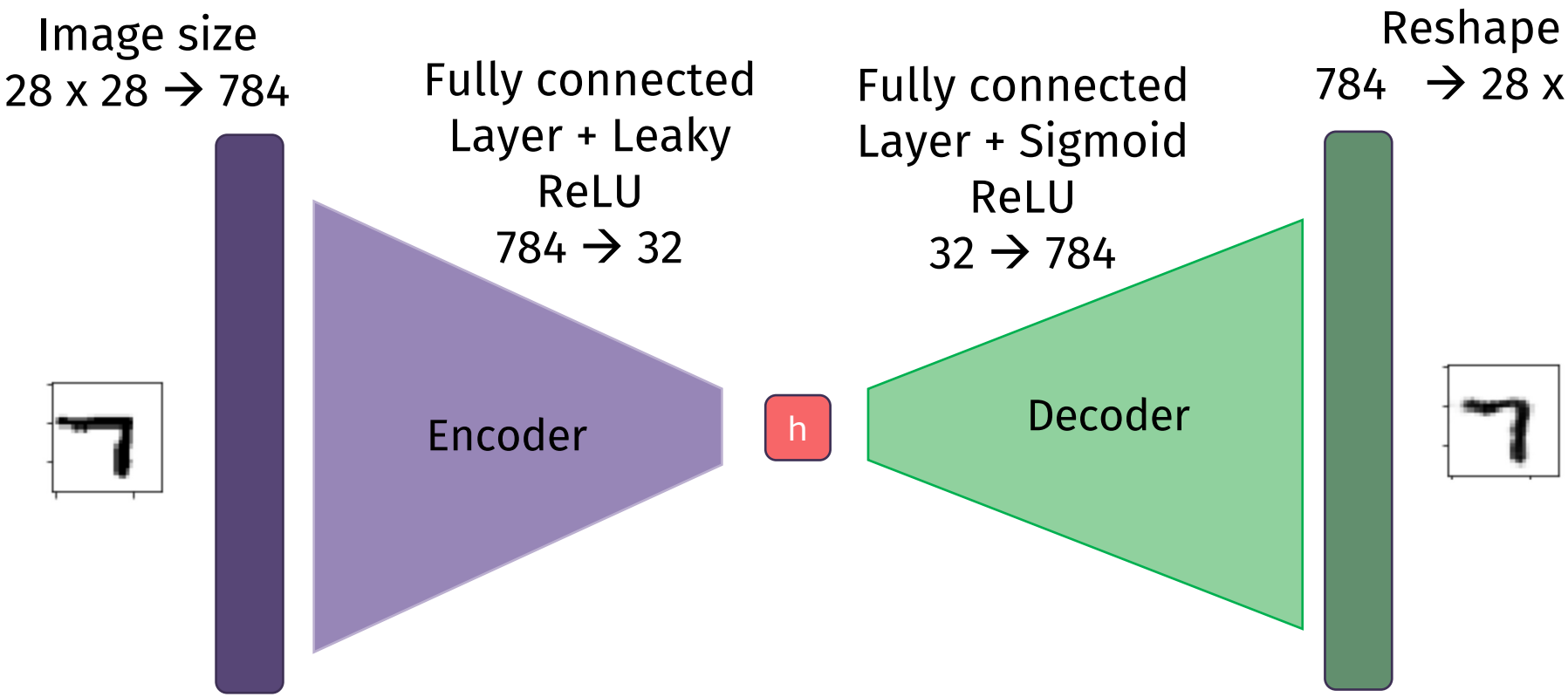
- Why do we need it?

# Embedding  or latent variables

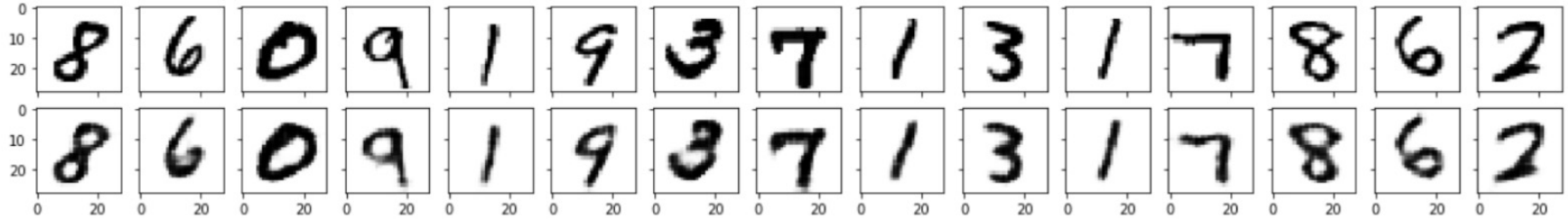


- Autoencoders can be seen as an **unsupervised** approach for learning a **lower-dimensional** feature representation
- After training, you can disregard the output and **use embedding as inputs** to classic machine learning methods
- **Transfer learning**: train autoencoders on large datasets and fine tune on your (smaller) dataset
- **Visualization** (projecting the embeddings in lower a dimensional space)

# Autoencoders: an example




Original



Reconstructed

# Dimensionality of the latent space

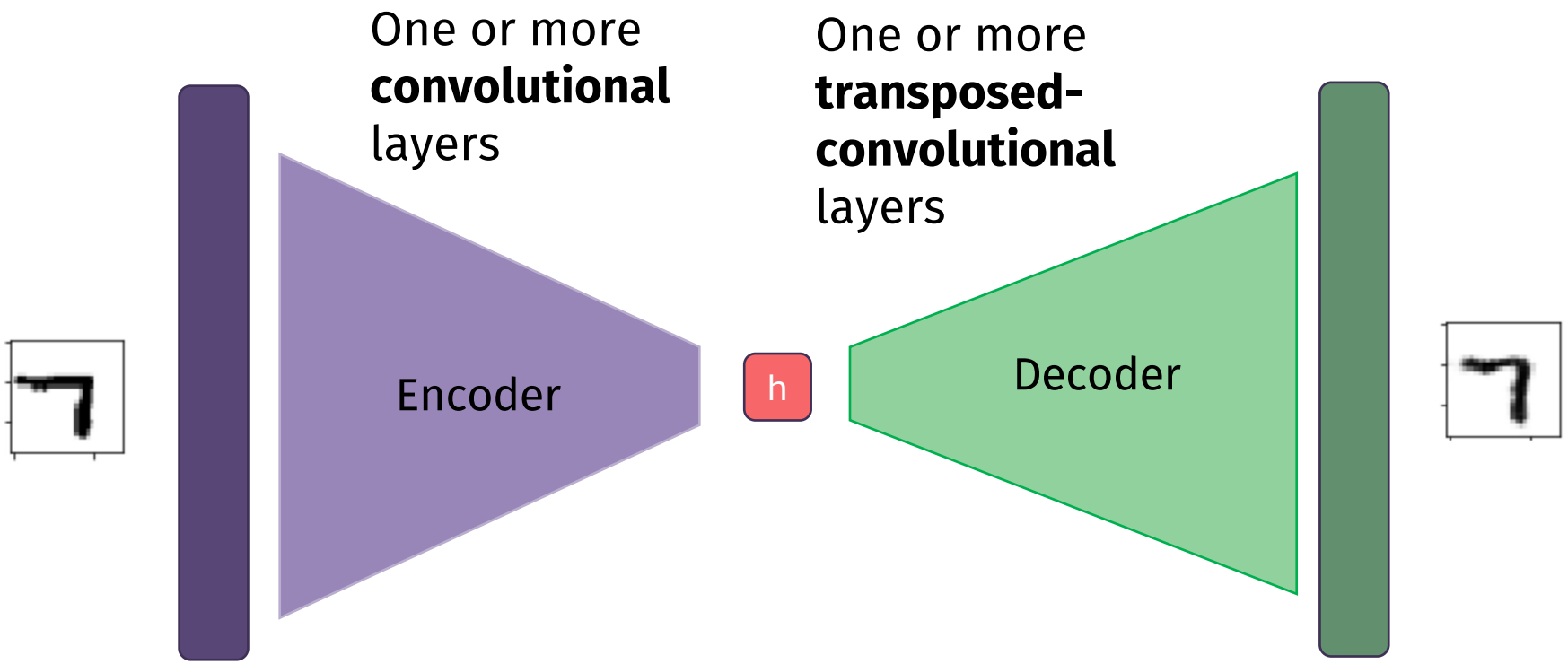
## reconstruction quality

2D Latent Space

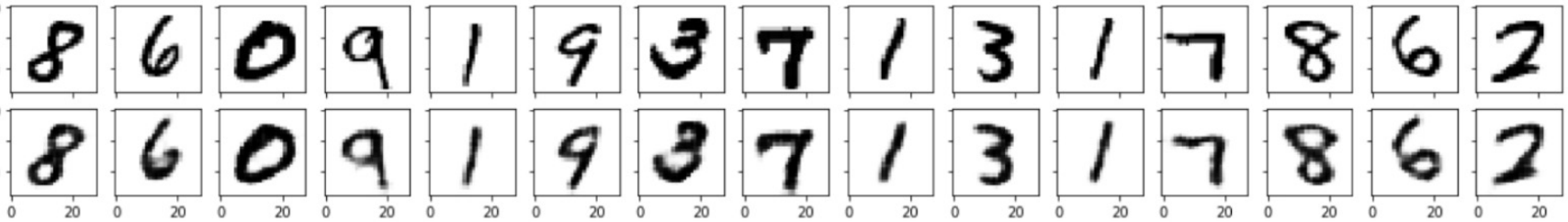
5D Latent Space

*Autoencoding compresses!*

# Convolutional autoencoders: the concept




Original



Reconstructed

# Transposed convolution

When managing image data, encoder and decoder are made of, respectively, convolutional and transposed-convolutional layers

**Transposed convolution** allows to go from a lower dimensional image to a higher dimensional image

# Regular convolution (in the encoder)

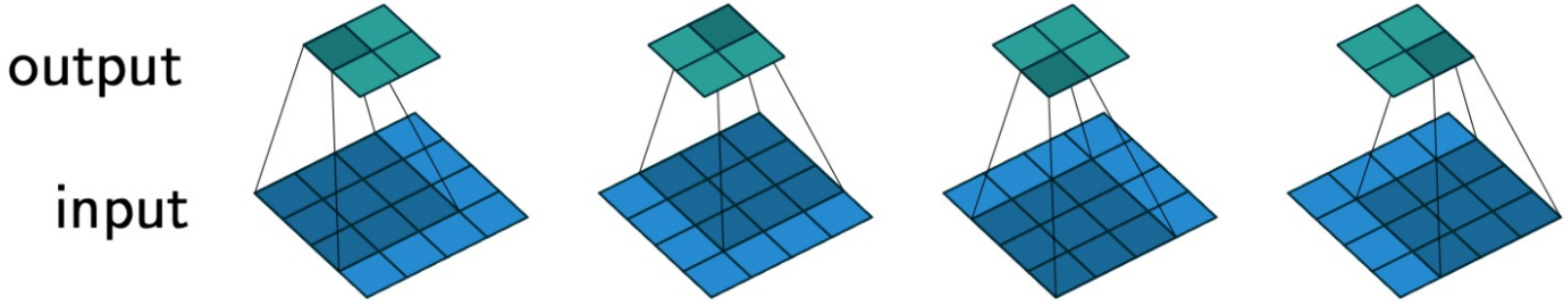
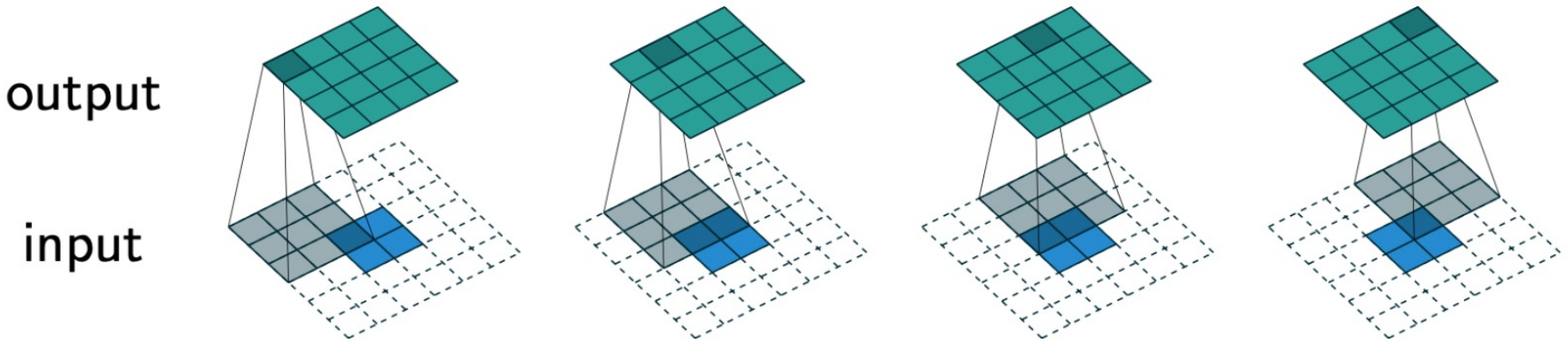



Figure 2.1: (No padding, unit strides) Convolving a $3 \times 3$ kernel over a $4 \times 4$ input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$).

*Image from https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L15_autoencoder/L15_autoencoder_slides.pdf*

# Transposed convolution (in the decoder)




*Dumoulin Visin https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L15_autoencoder/L15_autoencoder_slides.pdf*

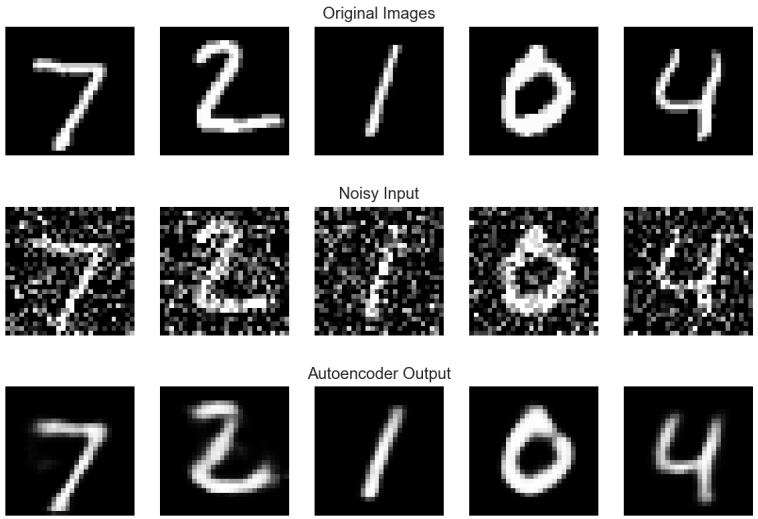
# Application example: denoising autoencoder (DAE)

It minimizes the loss

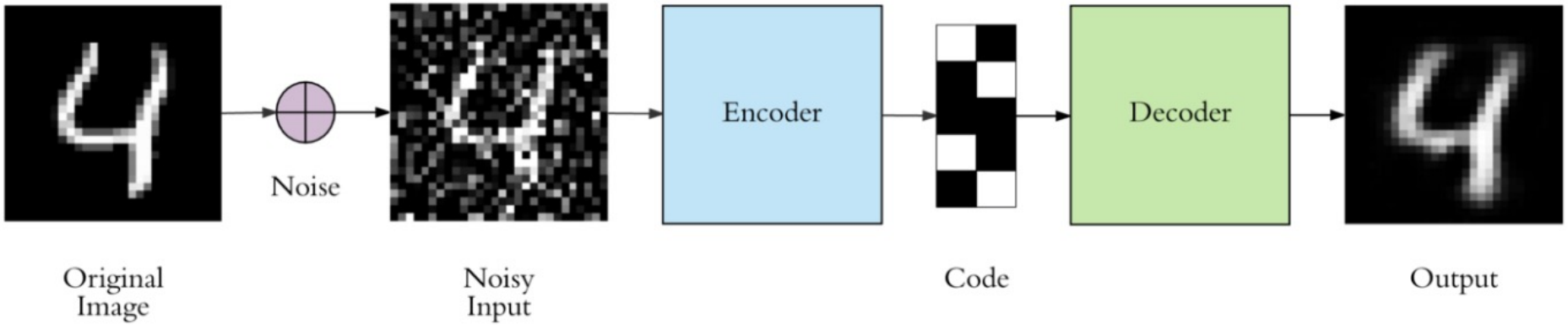
$$L(x, g(f(\tilde{x})))$$

where $\tilde{x}$ is a copy of $x$ corrupted by some form of noise

Thus, the encoder learns how to undo this corruption on the input rather than simply copying it

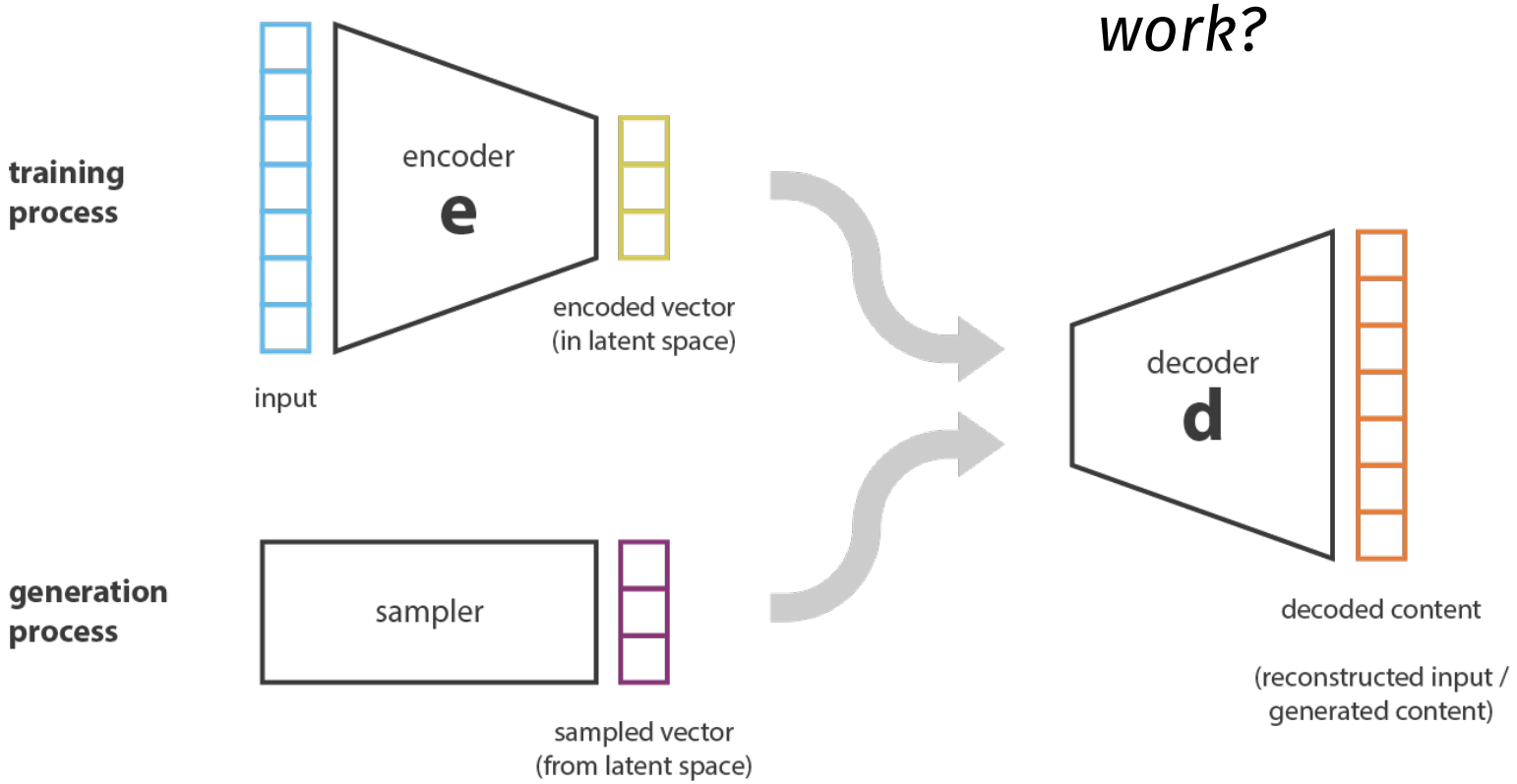


# Denoising autoencoders



# Autoencoder as data generator

*Does it work?*




*Picture from https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73*

# A latent space with no structure

- Difficult to ensure a priori an organization of the latent space that can allow for a generative process
- To produce latent space with a structure we may resort to the use of variational auto-encoders

# Variational autoencoders

- A variational autoencoder (VAE) is an autoencoder in which some good properties in the latent space are ensured
- Instead of encoding an input into a point in the latent space, a VAE encodes it as a distribution over the latent space.

# Autoencoders vs VAE

- Simple autoencoders

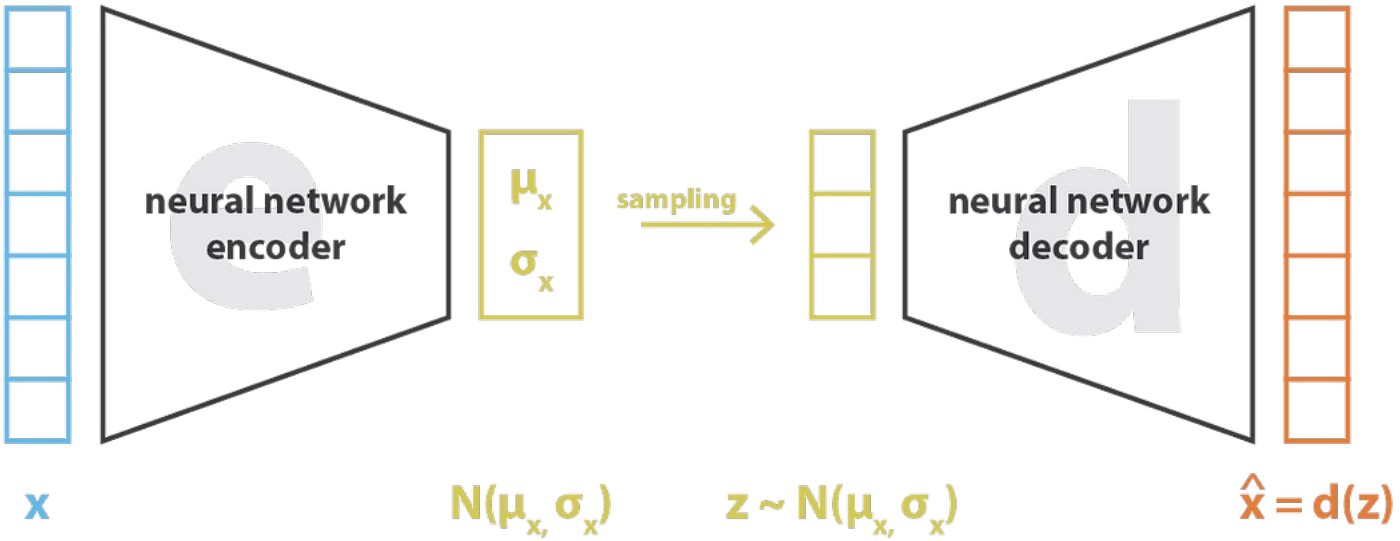
  Input x → encoding → latent representation z = e(x) → decoding → reconstruction of input d(z)

- VAE

  Input x → encoding → latent distribution p(z|x) → sampling → sampled latent representation z ~ p(z|x) → decoding → reconstruction of input d(z)

- The p distributions are chosen to be normal: the encoder can learn mean and the covariance matrix

# Variational Autoencoders




*Picture from https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73*

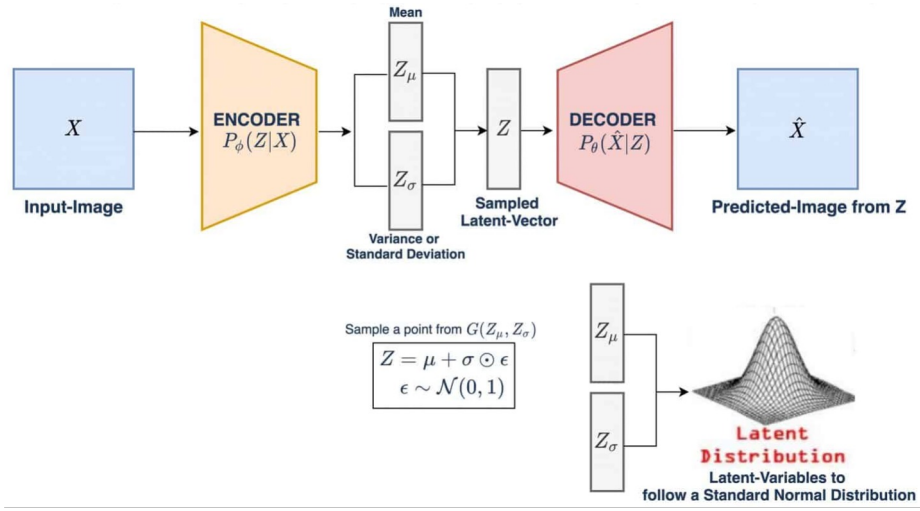
# Variational Autoencoder (VAE)

## More in general

$$\mathbb{E}_{P_\phi(Z|X)}[\log P_\theta(\hat{X}|Z)] - \beta KL(P_\phi(Z|X)||\mathcal{N}(0,1))$$

reconstruction error

regularizer




- Each sample is modelled as $\mathcal{N}(0,1)$
- Regularizer add properties:
  - *Continuity*: close points give similar content once decoded
  - *Completeness*: every point of the latent space should be meaningful

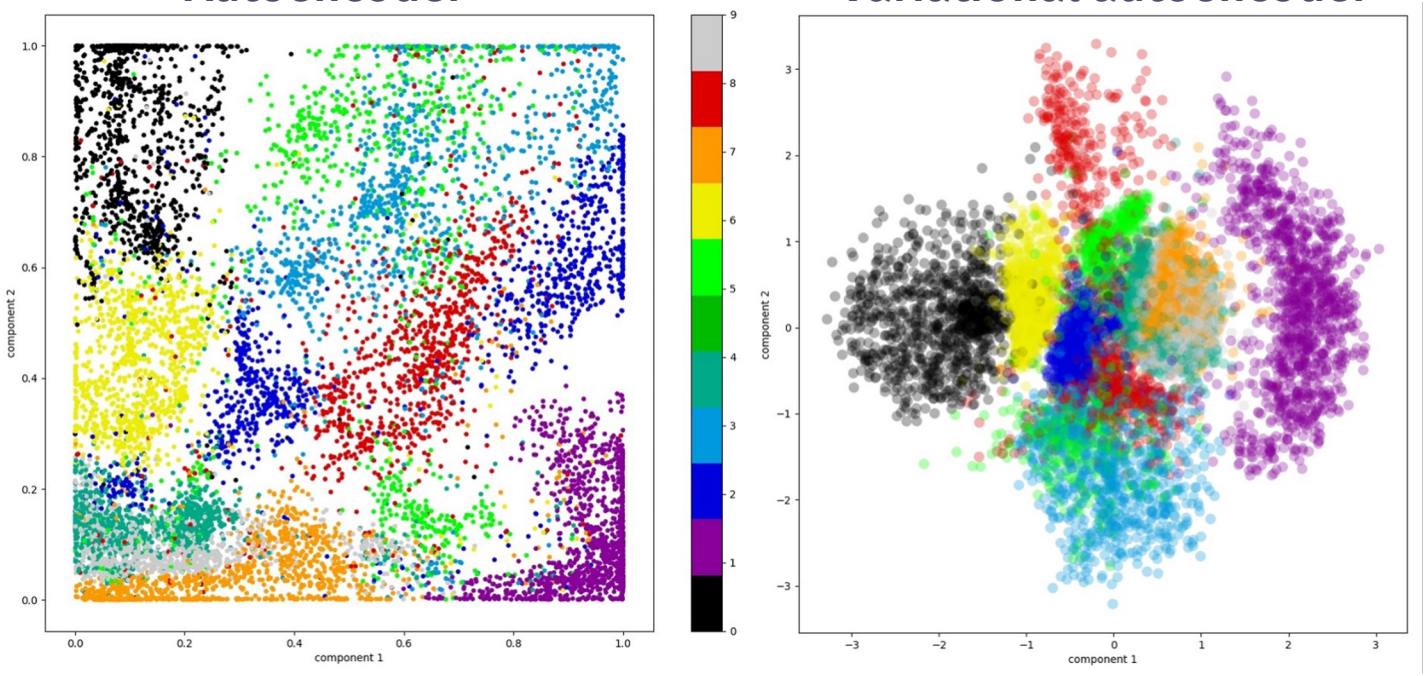




https://openreview.net/pdf?id=Sy2fzU9gl

# Latent representation

## Autoencoder

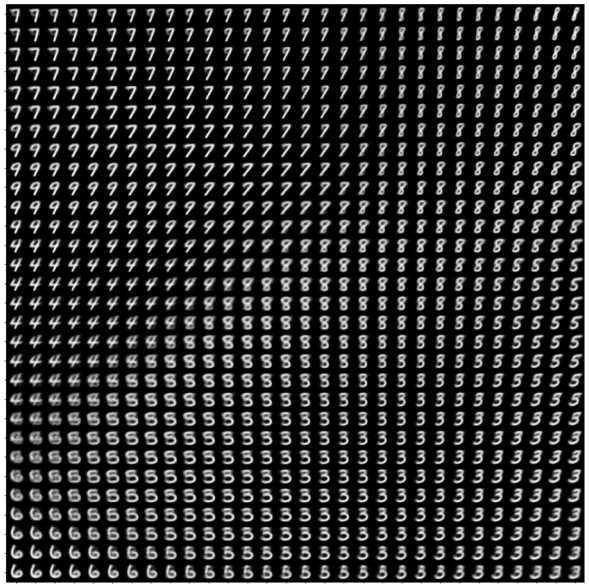



## Variational autoencoder

# Generate new samples from latent space

## Autoencoder

## Variational autoencoder



UniGe | MaLGa

# Generative Adversarial Networks
# GANs

# Generative Adversarial Networks (GANs)

- Their purpose is to **generate new data instances**

- They learn the distribution of the training set and can generate new data never seen before

- They are based on a game theoretic scenario in which a generator network must compete against an adversary

# GANs, already some years ago...

# A turing test

*A*

*B*

*Which one is real?*

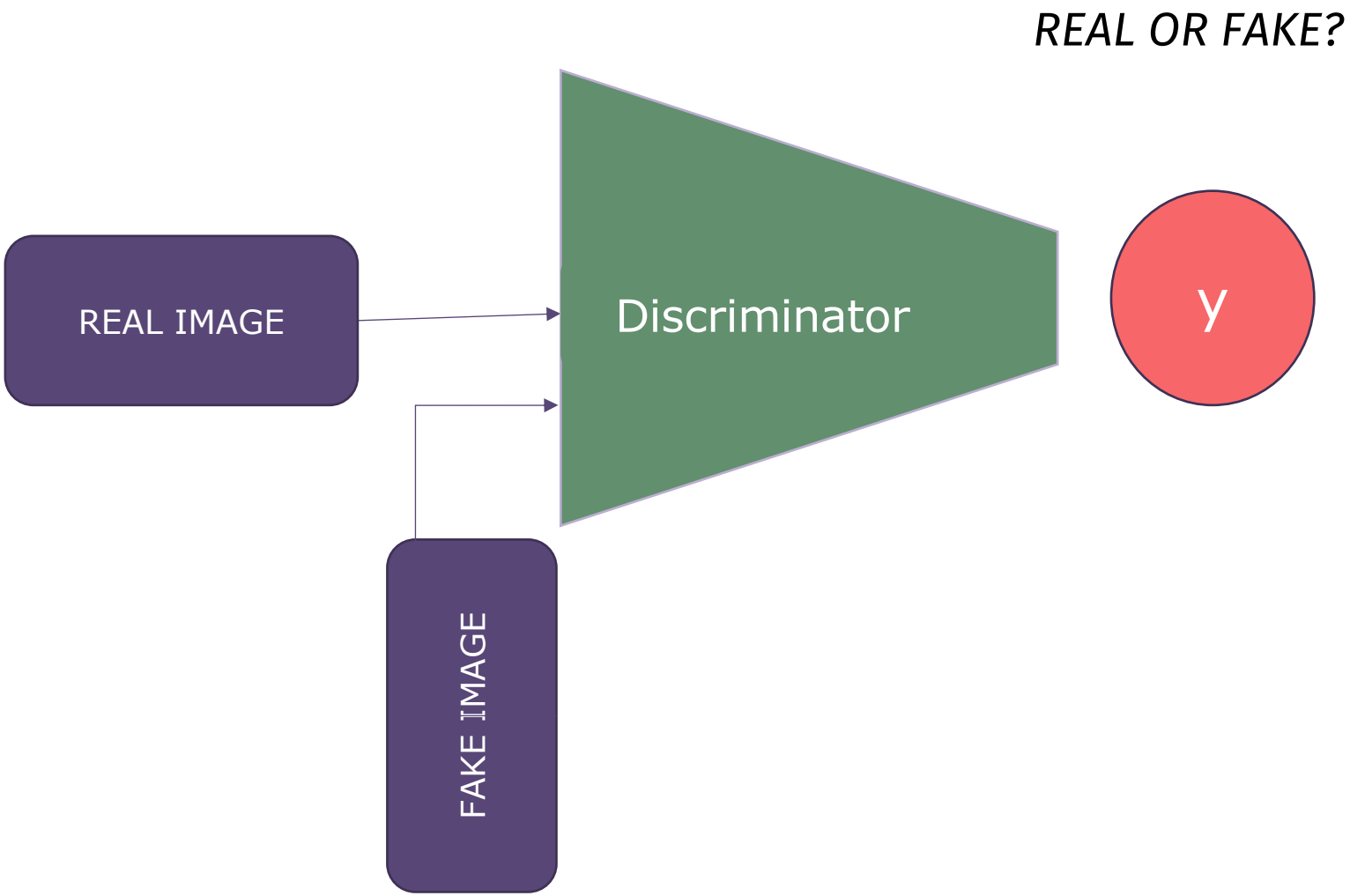
# GANs

- A **generator network** directly produces samples

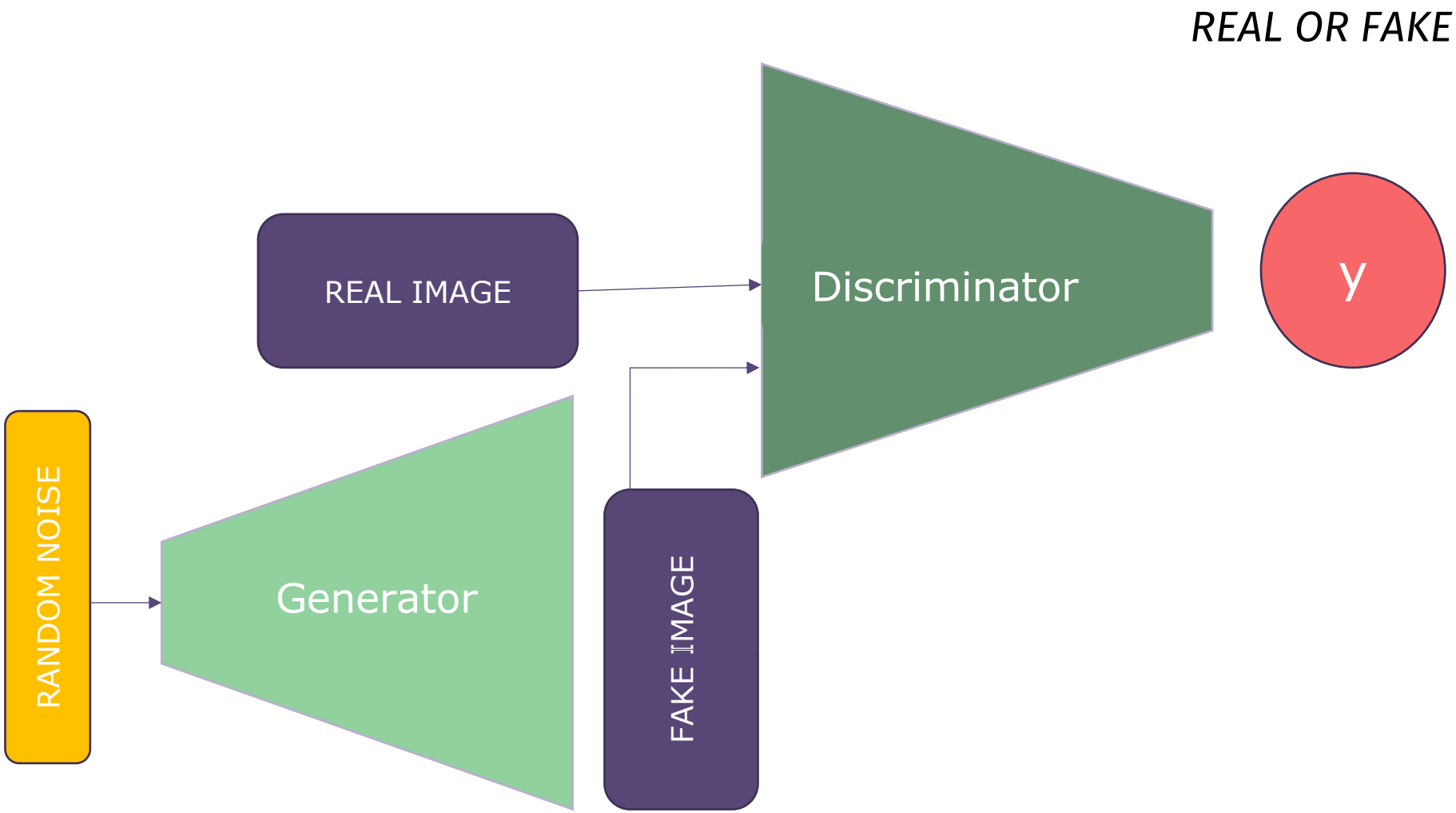
- Its adversary, the **discriminator network**, attempts to distinguish between samples drawn from the training data and samples drawn from the generator

- The discriminator estimates a probability values evaluating how likely is that the sample is a training example rather than a fake sample drawn from the model

# Discriminator

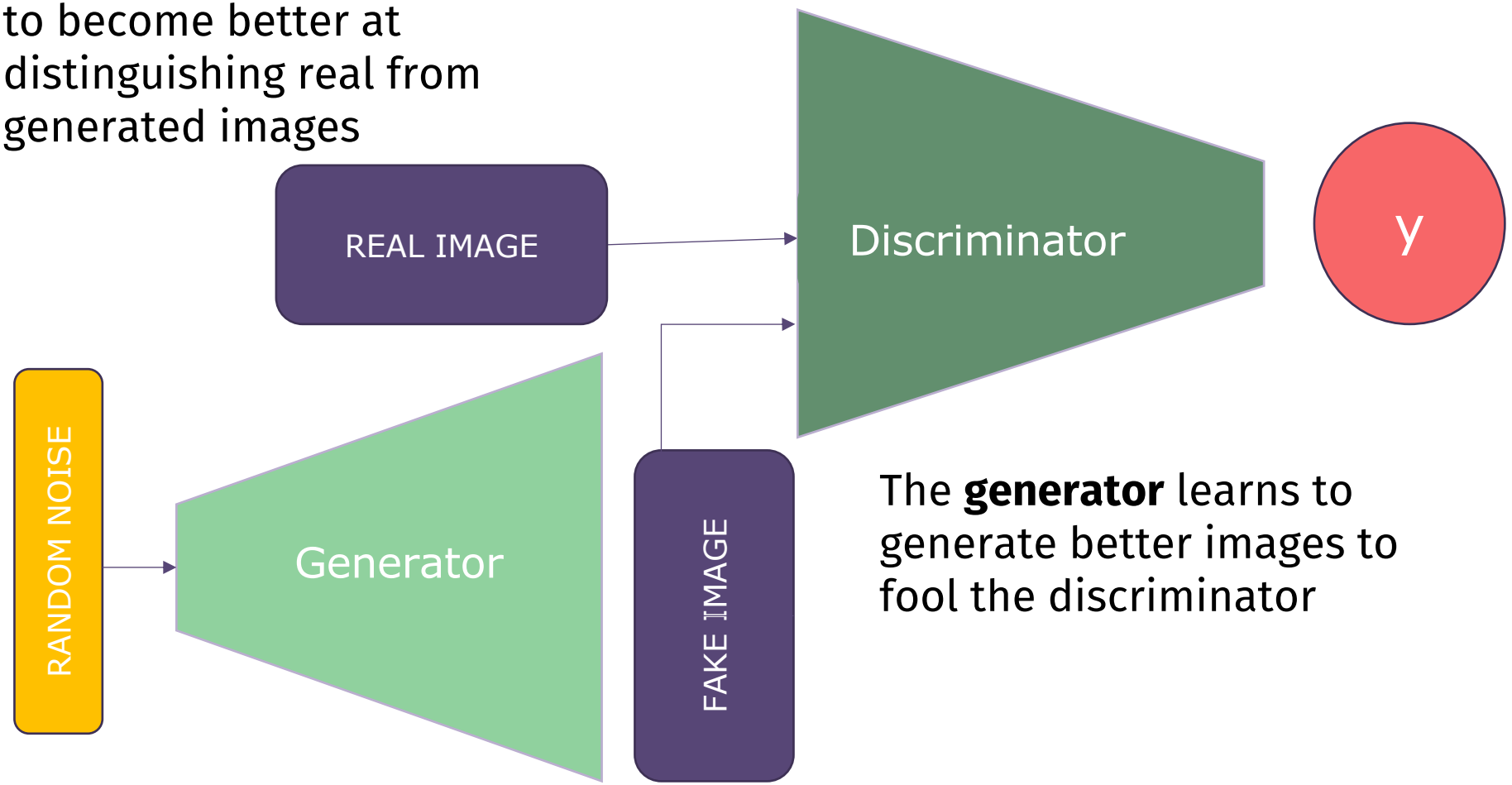


# Discriminator



# Discriminator

The **discriminator** learns to become better at distinguishing real from generated images




The **generator** learns to generate better images to fool the discriminator

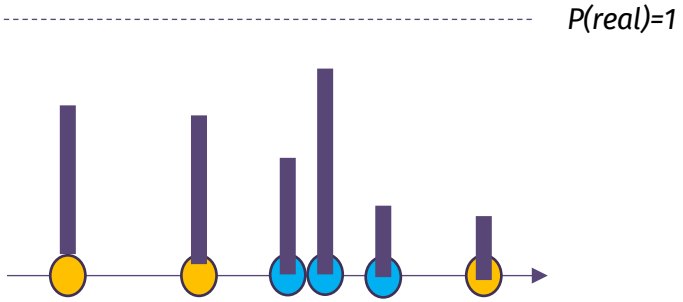
# Intuition

GENERATOR

# Intuition

GENERATOR

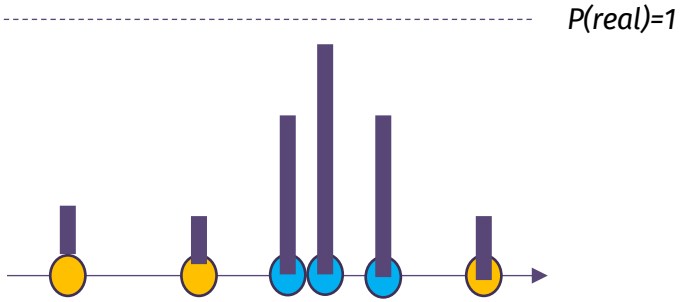
DISCRIMINATOR



# Intuition

GENERATOR

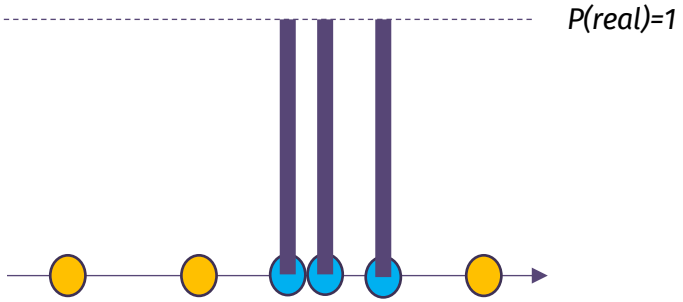
DISCRIMINATOR



# Intuition

GENERATOR

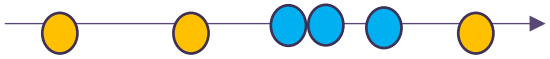
DISCRIMINATOR

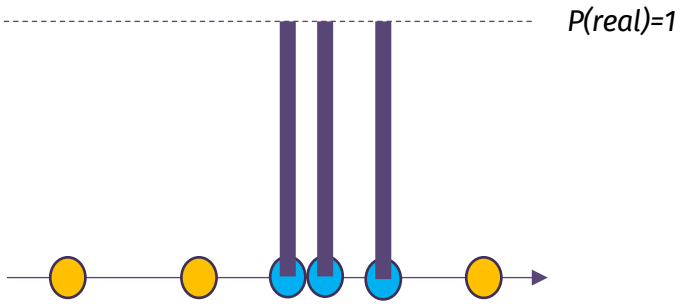


# Intuition

GENERATOR

DISCRIMINATOR

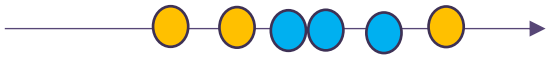


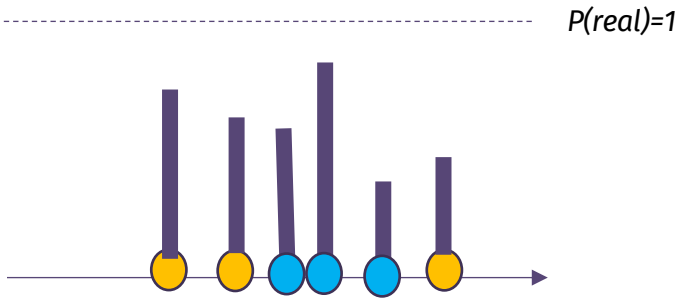


# Intuition

GENERATOR

DISCRIMINATOR

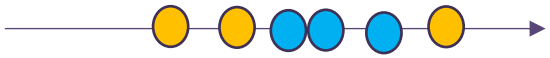


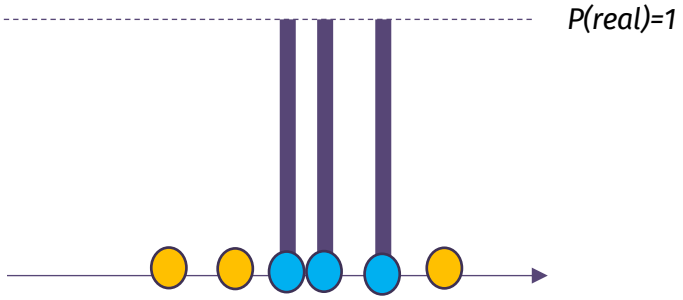


# Intuition

GENERATOR

DISCRIMINATOR





# Intuition

GENERATOR

DISCRIMINATOR


*...the process continues until the discriminator is unable to learn how to distinguish between real and fake*

# GAN model and training

The two players $G(z, \theta_g)$ and $D(x, \theta_d)$ are two differentiable functions implemented by Deep Neural Networks.

Two-player Minmax game with value function:

$$\min_G \max_D V(D, G) = \underbrace{\mathbb{E}_{x \sim p_{data}(x)} [\log D(x)]}_{\text{Real samples}} + \underbrace{\mathbb{E}_{z \sim p_z(z)} [\log (1 - D(G(z)))]}_{\text{Generated samples}}$$

The **Discriminator** wants D(x) = 1 and D(G(z))=0 → tries to maximize V

The **Generator** wants D(G(z))=1 → tries to minimize

Minmax is solved through alternating gradient descent: the parameters $\theta_d$ and $\theta_g$ are updated iteratively.

# GAN model and training

$$\min_G \max_D V(D, G)$$

- At convergence, the generator's samples are indistinguishable from real data, and the discriminator outputs 0.5 everywhere

- In other words the convergence is reached when the actions of one of the players do not change depending on the actions of the other players

- As you can imagine, training can be very slow

# Training a GAN model

**Algorithm 1** Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, $k$, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

**for** number of training iterations **do**

  **for** $k$ steps **do**

  - Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  **end for**

  - Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

**end for**

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

*From* "**Generative Adversarial Networks**", **by** Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, 2014

UniGe | MaLGa

# A glimpse of recent approaches

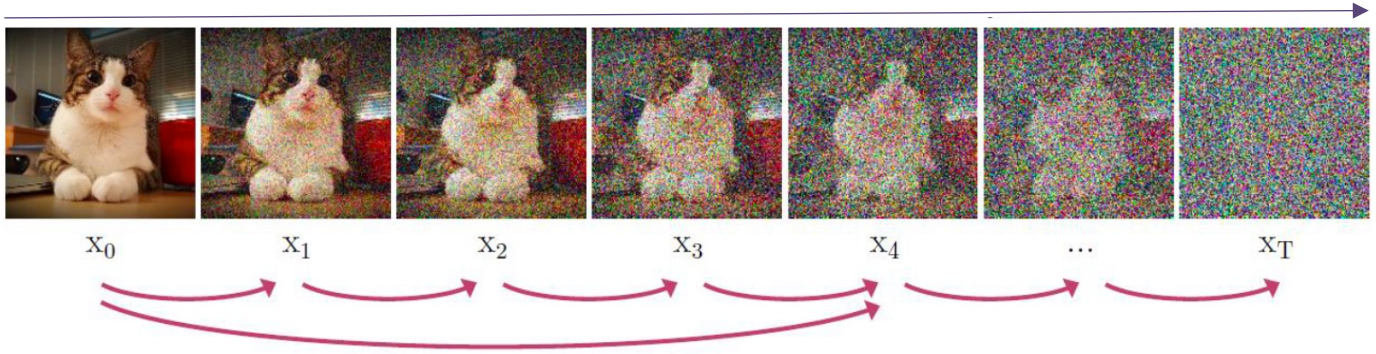
# Diffusion models

- A recent family of generative algorithms
- Training diffusion models is based on two main processes:
  - Corrupting data by adding Gaussian noise → **Forward diffusion process**
  - Learning how to recover the original data by reversing the noising process → **Reverse diffusion process**

# Diffusion models: processes

*Forward diffusion process*

*Data*

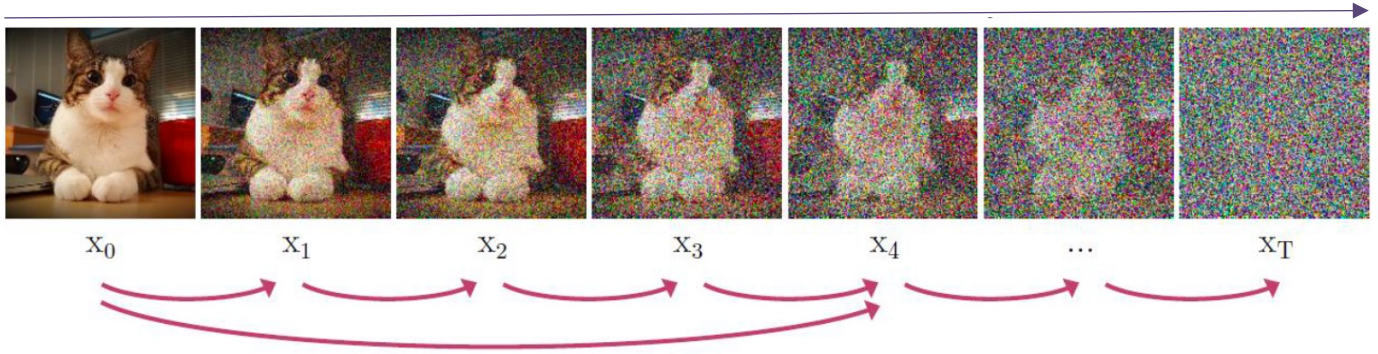



*Noise*

# Diffusion models: processes

*Forward diffusion process*

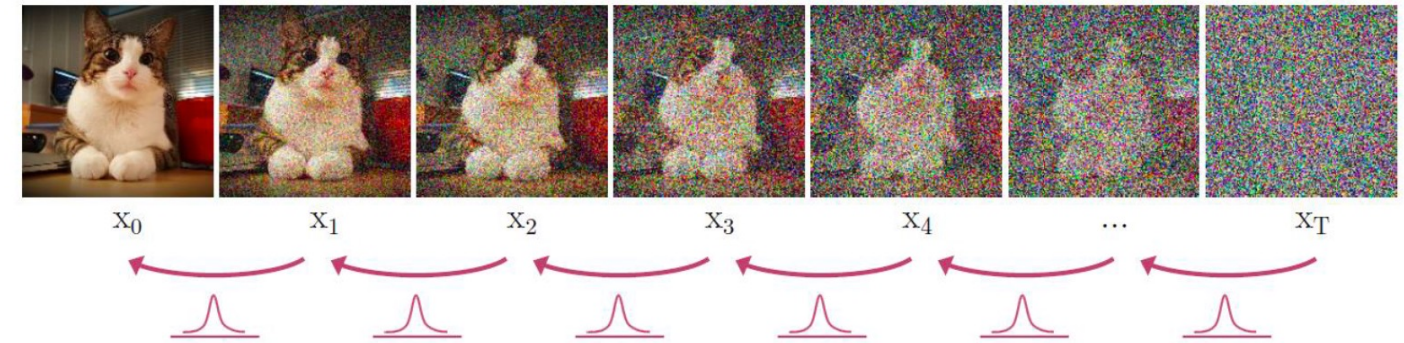
*Data*




*Noise*

*Reverse denoising process (generative)*

*Data*




*Noise*

# Diffusion models + LLM

- Nowadays diffusion models are mostly used in conjunction with language models

- A PROMPT is used to provide a “generation request”

A teddy bear on a playground

https://scale.com/guides/diffusion-models-guide

# Diffusion models + LLM

- Nowadays diffusion models are mostly used in conjunction with language models

painting of a human cyborg in a city salvador dali 8K highly detailed

- A PROMPT is used to provide a “generation request”

painting of a human cyborg in a city picasso 8K highly detailed

https://scale.com/guides/diffusion-models-guide

# Diffusion models + LLM


*A cute sloth holding a small treasure chest. A bright golden glow is coming from the chest*


*A photo of a Shiba Inu dog with a backpack riding a bike. It is wearing sunglasses and a beach hat*


*A dragon fruit wearing karate belt in the snow*

*From https://arxiv.org/pdf/2205.11487*

UniGe