

Generative Networks

Session outline

Autoencoders

- Deterministic encoding

Variational Autoencoders (VAE)

- Stochastic encoding

Generalised Adversarial Networks (GAN)

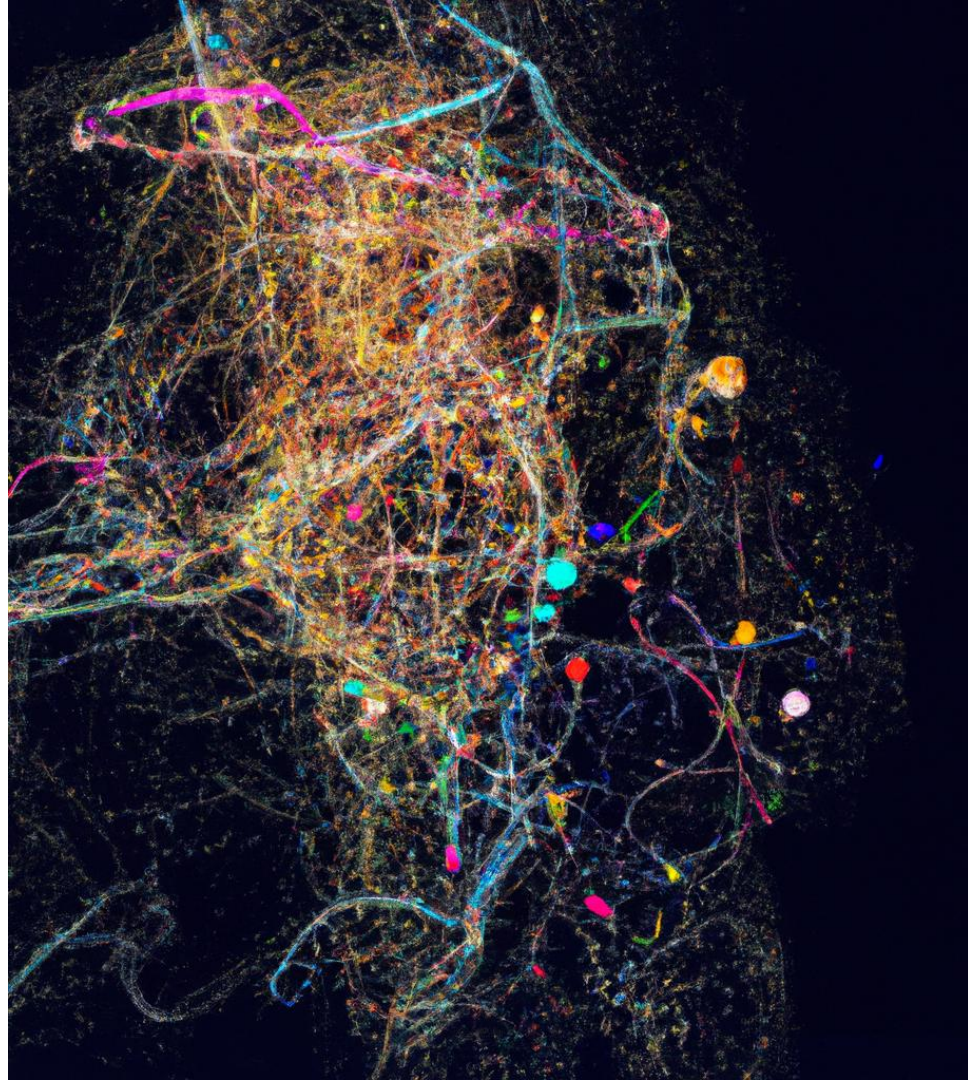
- Two networks in one

Diffusion Models (DM)

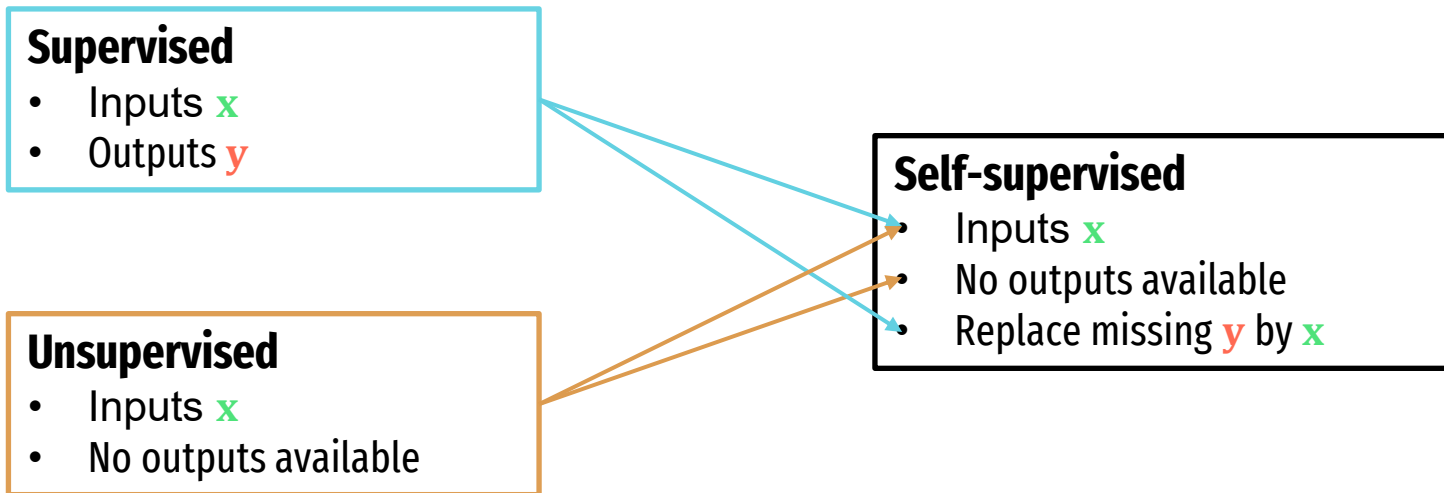
- State of the art

Autoencoders

Deterministic encoding



The self-supervised setting



*Not to be confused with semi-supervised or weakly-supervised that refer to a setting where the output y is partially available

The origins of AE

Autoencoders were originally proposed in 1986 by Rumelhart, Hinton and Williams

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Learning internal representations by error propagation. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations. MIT Press, Cambridge, MA, USA, 318–362.

General Definition

An **autoencoder** is a type of algorithm with the primary purpose of learning an **informative representation** of the data that can be used for different applications by learning to **reconstruct** a set of input observations.

Bank, Dor, Noam Koenigstein, and Raja Giryes. "Autoencoders." Machine learning for data science handbook: data mining and knowledge discovery handbook (2023): 353-374.

The origins of AE

Autoencoders were originally proposed in 1986 by Rumelhart, Hinton and Williams

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Learning internal representations by error propagation. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations. MIT Press, Cambridge, MA, USA, 318–362.

Practical Definition

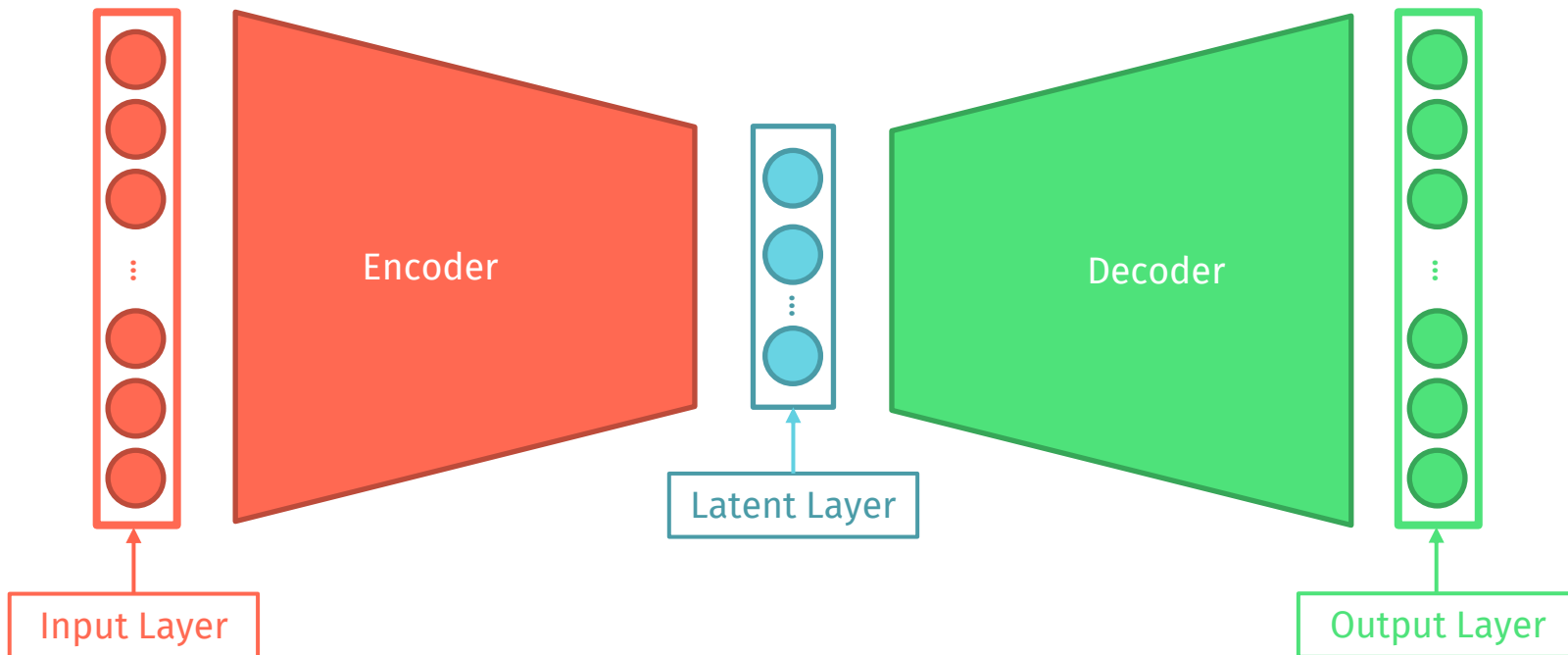
An **autoencoder** is a specific type of a neural network which is mainly designed to **encode** the input into a **compressed and meaningful representation** and then **decode** it back such that the reconstructed input is [as] similar as possible to the original one.

Bank, Dor, Noam Koenigstein, and Raja Giryes. "Autoencoders." Machine learning for data science handbook: data mining and knowledge discovery handbook (2023): 353-374.

General Architecture

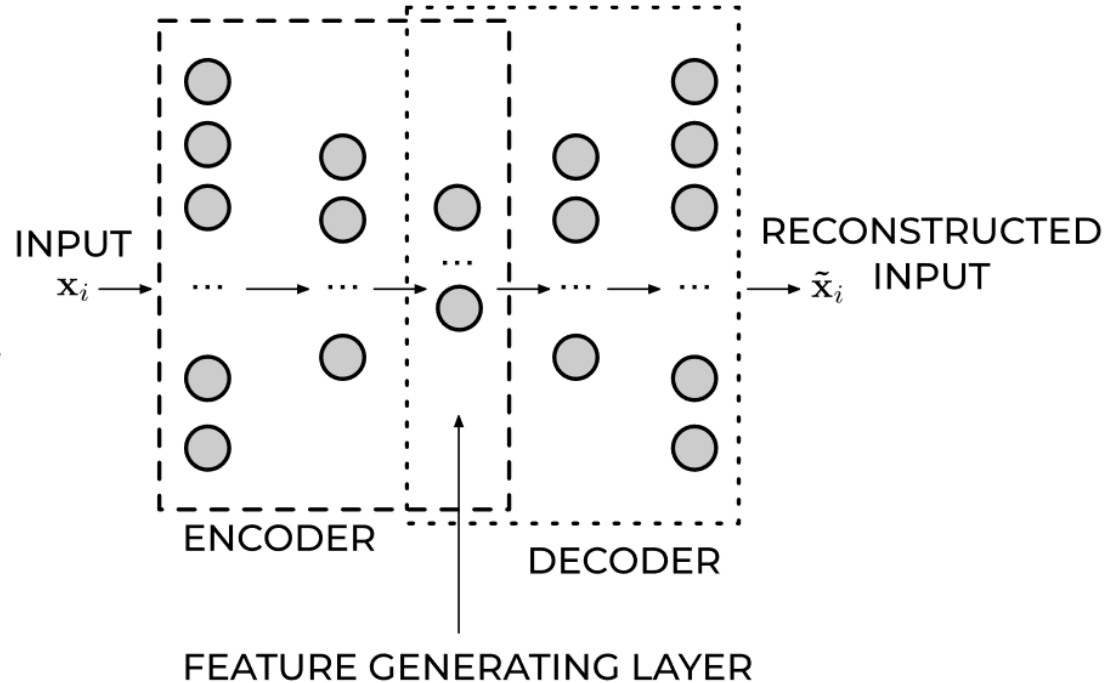
Autoencoders

An **encoder** and a **decoder** with a **latent** layer in between them



Feed-Forward AE

- A typical FFAE architecture has an **odd** number of layers and is symmetrical with respect to the middle layer
- The **encoder** can reduce the number of dimensions of the input observation n and create a learned representation h_i of the input that has a smaller dimension $q < n$



Convolutional AE

MNIST data (28x28x1)

The encoding part is pretty much what you would expect for a traditional CNN

Layer	Output
Input	(B,28,28,1)
Conv2D	(B,14,14,32)
Conv2D	(B,7,7,64)
Flatten	(B,3136)
Dense	(B,16)

Encoder

Layer	Output
Dense	(B,10)

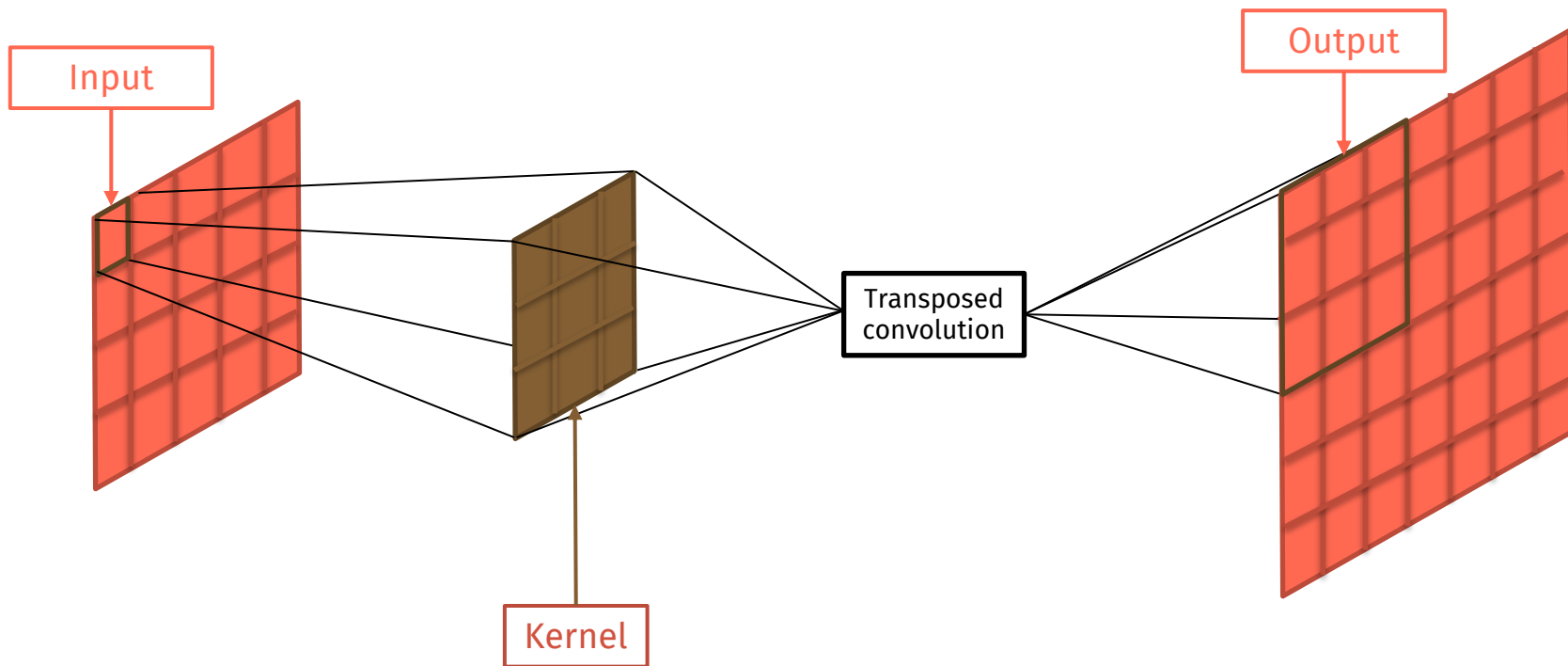
Latent

?

Question
What about the decoder?
How can we mirror a CNN?

Transposed Convolution

Autoencoders



$$\text{Output} = a(\sum \text{Input} \otimes \text{kernel} + b), \text{ with } a \text{ an activation function and } b \text{ a bias}$$

Transposed Convolution

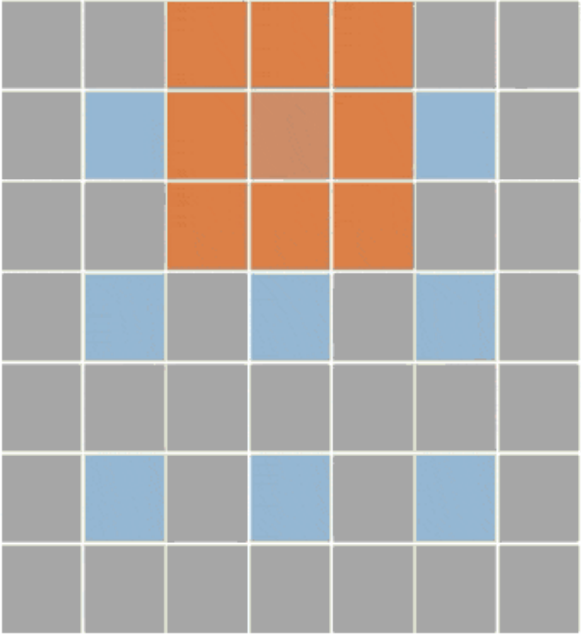
Basic example :

Input image : (2,2)
Kernel : (2,2)
Output image : (3,3)

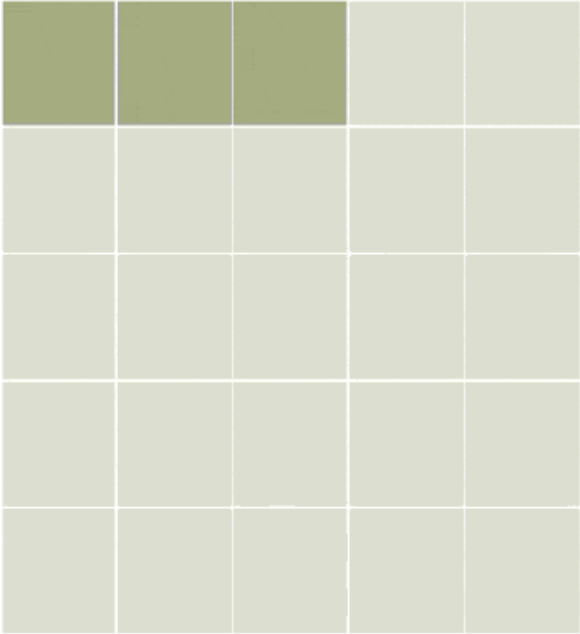
$$\begin{bmatrix} 1 & 3 \\ 0 & 2 \end{bmatrix} \otimes \begin{bmatrix} 1 & 3 \\ 0 & 2 \end{bmatrix} = \sum$$

$$\left(\begin{array}{cc} \begin{bmatrix} 1 & 3 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 3 & 9 \\ 0 & 0 & 6 \\ 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 6 \\ 0 & 0 & 4 \end{bmatrix} \end{array} \right) = \begin{bmatrix} 1 & 6 & 9 \\ 0 & 4 & 12 \\ 0 & 0 & 4 \end{bmatrix}$$

Transposed Convolutions Illustrated



Input



Output

Equivalence with Convolutions

For each **transposed convolution
there exist an equivalent **convolution****

An idea of the proof lies in the fact that the **kernel** is simply a matrix operator M for a simple **convolution**, and M^T is the equivalent **transposed convolution** operator

Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." arXiv preprint arXiv:1603.07285 (2016).

Upsampling

Many ways to upsample (e.g., nearest, linear, bilinear, bicubic, trilinear)

2	0
3	4



An example of 'nearest' upsampling

2	2	0	0
2	2	0	0
3	3	4	4
3	3	4	4

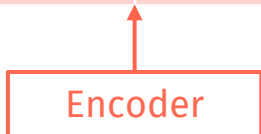
In practice, **Upsampling** is more commonly used than **Transposed Convolutions**

A simple architecture

Autoencoders

MNIST data (28x28x1)

Layer	Output
Input	(B,28,28,1)
Conv2D	(B,14,14,32)
Conv2D	(B,7,7,64)
Flatten	(B,3136)
Dense	(B,16)



Layer	Output
Dense	(B,10)



Layer	Output
Dense	(B,3136)
Reshape	(B,7,7,64)
Conv2D_T	(B,14,14,64)
Conv2D_T	(B,28,28,32)
Conv2D_T	(B,28,28,1)



The reconstruction loss depends on the output layer activation function

- Linear or ReLU :

$$l^{MSE}(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{n} \sum_{k=1}^n (x_k - \hat{x}_k)^2 \Rightarrow \text{Mean Squared Error (MSE)}$$

- Sigmoid

$$l^{BCE}(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^p x_{i,j} \log \tilde{x}_{i,j} + (1 - x_{i,j}) \log(1 - \tilde{x}_{i,j}) \Rightarrow \text{Binary Cross-Entropy (BCE)}$$

Equivalence between AE and PCA

A link between AE and PCA is formulated in 1989 by Baldi and Hornik

- Baldi, Pierre, and Kurt Hornik. "Neural networks and principal component analysis: Learning from examples without local minima." *Neural networks* 2.1 (1989): 53-58.

The equivalence between AE and PCA is shown in Plaut (2018) and rewritten by Umberto (2022) as follows

- Plaut, Elad. "From principal subspaces to principal components with linear autoencoders." arXiv preprint arXiv:1804.10253 (2018).
- Michelucci, Umberto. "An introduction to autoencoders." arXiv preprint arXiv:2201.03898 (2022).

Under the following conditions

- The encoder is a linear function
- The decoder is a linear function
- The loss function is the *MSE*
- The inputs are normalized as follows

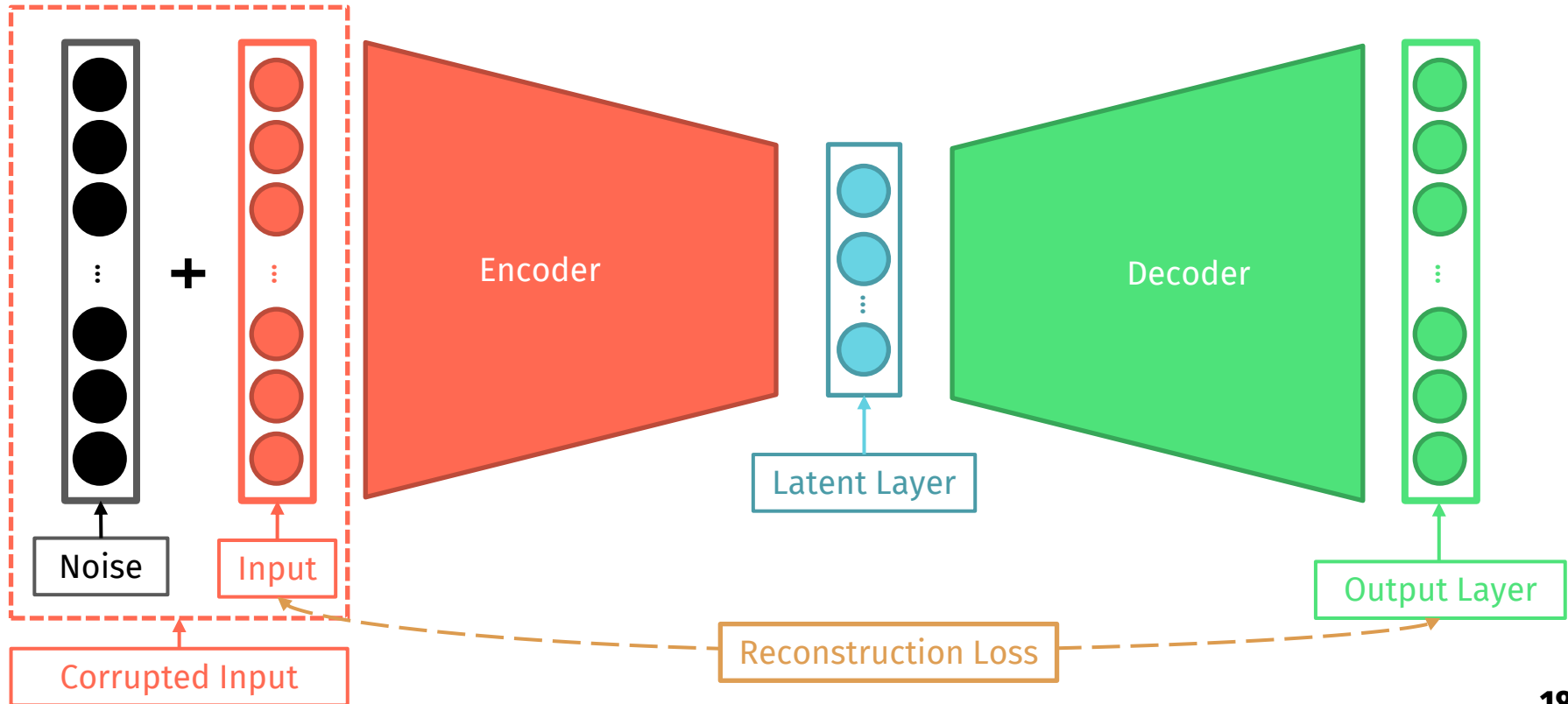
$$\hat{x}_{i,j} = \frac{1}{\sqrt{M}} \left(x_{i,j} - \frac{1}{M} \sum_{k=1}^M x_{k,j} \right)$$

$i \in \{1 \dots n\}$ with n the number of observations and $j \in \{1 \dots p\}$ with p the number of features

Then we have **AE** \Leftrightarrow **PCA**

Denoising AE Architecture

Autoencoders



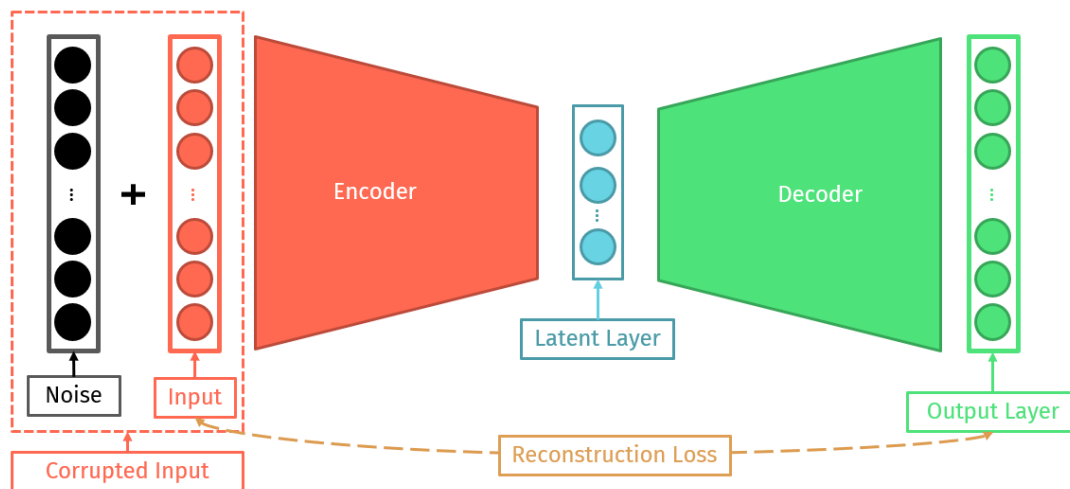
Denoising AE Principles

Denoising autoencoders can be viewed either as a regularization option or as robust autoencoders which can be used for error correction

In these architectures the input is disrupted by some noise and the autoencoder is expected to reconstruct the clean version of the input

Two common options for the noise are

- Additive white gaussian noise
- Erasures using dropout



Additional materials

Autoencoders

Regularisation

- Sparse AE
- Contractive AE

Michelucci, Umberto. "An introduction to autoencoders." arXiv preprint arXiv:2201.03898 (2022).

Other applications

- Dimensionality Reduction
- Classification with Latent Features
- Anomaly Detection

e.g, Extreme Weather Events

Variational AE

Stochastic encoding



The limits of AE

AE yields great results for denoising simple images but is quite **limited** in practice

Deterministic Encoding

Autoencoders provide a deterministic mapping from input to latent space, meaning each input is mapped to a fixed point in the latent space.

This can limit their ability to generate diverse and novel samples.

The limits of AE

AE yields great results for denoising simple images but is quite **limited** in practice

Stochastic

~~Deterministic~~ Encoding

Autoencoders provide a deterministic mapping from input to latent space, meaning each input is mapped to a fixed point in the latent space.

This can limit their ability to generate diverse and novel samples.

How can we make the encoding random ?

A quick overview of Variational Inference (VI)

The goal of VI is to approximate distributions that are hard to compute

Posterior distributions

$$p(z|x) = \frac{p(x|z)p(z)}{\int p(x|z)p(z)dz}$$

hard to compute*

*in most cases the denominator (also called the evidence) is intractable as it requires to integrate over all possible values in the latent space

A quick overview of Variational Inference (VI)

Let's introduce a variational distribution $q(z)$ with parameters θ we control (e.g., Gaussian)

Approximation

$$q(z) \approx p(z|x) = \frac{p(x|z)p(z)}{\int p(x|z)p(z)dz}$$

It is possible to optimize the variational parameters θ to approximate the true posterior

In this case, optimising θ means minimising the « distance » between $q(z)$ and $p(z|x)$

A quick overview of Variational Inference (VI)

Let's introduce a metric **KL** independent of the evidence

Kullback-Leibler Divergence

$$KL(q(z) || p(z|x)) = \int_z q(z) \log \frac{q(z)}{p(z|x)} dz = \mathbb{E}_{z \sim q}[\log q(z)] - \mathbb{E}_{z \sim q}[\log p(z|x)]$$

And with Bayes Formula we finally get

Independent of z

$$KL(q(z) || p(z|x)) = \mathbb{E}_{z \sim q}[\log q(z)] - \mathbb{E}_{z \sim q}[\log p(x|z)p(z)] + \log p(x)$$

A quick overview of Variational Inference (VI)

Minimising the KL is equivalent to maximising the ELBO with regards to θ

Evidence Lower Bound (ELBO) or Variational Lower Bound (VLB)

$$ELBO = \mathbb{E}_{z \sim q_{\theta}(z)} [\log p(x|z)p(z) - \log q(z)]$$

Using additional assumptions we can derive a coordinate-ascent VI (using mean-field)

Gradient descent can also be derived assuming q belongs to the exponential family

A quick overview of Variational Inference (VI)

Without any additional assumption we can try and calculate the **ELBO gradient**

ELBO gradient

$$\begin{aligned} & \nabla_{\theta} \mathbb{E}_{z \sim q_{\theta}(z)} [\log p(x|z)p(z) - \log q_{\theta}(z)] \\ &= \nabla_{\theta} \int_z q_{\theta}(z) [\log p(x|z)p(z) - \log q_{\theta}(z)] dz \end{aligned}$$

A natural thing to do here ...

A quick overview of Variational Inference (VI)

Without any additional assumption we can try and calculate the **ELBO gradient**

ELBO gradient

$$\begin{aligned} & \nabla_{\theta} \mathbb{E}_{z \sim q_{\theta}(z)} [\log p(x|z)p(z) - \log q_{\theta}(z)] \\ &= \nabla_{\theta} \int_z q_{\theta}(z) [\log p(x|z)p(z) - \log q_{\theta}(z)] dz \end{aligned}$$

A natural thing to do here ...

Pushing the derivative inside the integral

A quick overview of Variational Inference (VI)

Kingma and **Welling** introduced a more general purpose algorithm in **2013**

Reparameterisation trick

- Let us take a known distribution p' that does not depend on θ
- Let us also take a differentiable transform g_θ such that

$$\begin{aligned}\epsilon &\sim p'(\epsilon) \\ z &= g_\theta(\epsilon, x)\end{aligned}$$

$$\nabla_\theta ELBO = \nabla_\theta \mathbb{E}_{q_\theta} [\log p(x, z) - \log q(z)] = \nabla_\theta \mathbb{E}_{p'} [\log p(x, g_\theta(\epsilon, x)) - \log q_\theta(g_\theta(\epsilon, x))]$$

Then we can write the following Monte-Carlo estimator

$$\nabla_\theta ELBO \approx \nabla_\theta \frac{1}{L} \sum_{l=1}^L \log p(x, g_\theta(\epsilon_l, x)) - \log q_\theta(g_\theta(\epsilon_l, x)) \quad \epsilon \sim p'(\epsilon)$$

A quick overview of Variational Inference (VI)

Let's roll down the calculations

ELBO gradient

$$\begin{aligned} & \nabla_{\theta} \int_{\mathbf{z}} q_{\theta}(\mathbf{z}) [\log p(x|\mathbf{z})p(\mathbf{z}) - \log q_{\theta}(\mathbf{z})] d\mathbf{z} \\ &= \int_{\mathbf{z}} \nabla_{\theta} (q_{\theta}(\mathbf{z}) [\log p(x|\mathbf{z})p(\mathbf{z}) - \log q_{\theta}(\mathbf{z})]) d\mathbf{z} \quad (\text{Leibniz rule}) \\ &= \int_{\mathbf{z}} \nabla_{\theta} (q_{\theta}(\mathbf{z})) [\log p(x|\mathbf{z})p(\mathbf{z}) - \log q_{\theta}(\mathbf{z})] d\mathbf{z} + \int_{\mathbf{z}} q_{\theta}(\mathbf{z}) \nabla_{\theta} [\log p(x|\mathbf{z})p(\mathbf{z}) - \log q_{\theta}(\mathbf{z})] d\mathbf{z} \\ &= \int_{\mathbf{z}} \nabla_{\theta} (q_{\theta}(\mathbf{z})) [\log p(x|\mathbf{z})p(\mathbf{z}) - \log q_{\theta}(\mathbf{z})] d\mathbf{z} + \int_{\mathbf{z}} q_{\theta}(\mathbf{z}) \left[0 - \frac{\nabla_{\theta}(q_{\theta}(\mathbf{z}))}{q_{\theta}(\mathbf{z})} \right] d\mathbf{z} \end{aligned}$$

Knowing that $\nabla_{\theta} (q_{\theta}(\mathbf{z})) = q_{\theta}(\mathbf{z}) \nabla_{\theta} \log q_{\theta}(\mathbf{z})$

$$\begin{aligned} &= \int_{\mathbf{z}} q_{\theta}(\mathbf{z}) \nabla_{\theta} (\log q_{\theta}(\mathbf{z})) [\log p(x|\mathbf{z})p(\mathbf{z}) - \log q_{\theta}(\mathbf{z})] d\mathbf{z} \\ &= \mathbb{E}_{\mathbf{z} \sim q_{\theta}(\mathbf{z})} [\nabla_{\theta} \log q_{\theta}(\mathbf{z}) (\log p(x|\mathbf{z})p(\mathbf{z}) - \log q_{\theta}(\mathbf{z}))] \\ &= \frac{1}{L} \sum_{i=1}^L \nabla_{\theta} \log q_{\theta}(\mathbf{z}_i) (\log p(x|\mathbf{z}_i)p(\mathbf{z}_i) - \log q_{\theta}(\mathbf{z}_i)), \quad \mathbf{z}_i \sim q_{\theta}(\mathbf{z}) \quad (\text{Monte-Carlo Simulation}) \end{aligned}$$

This estimator has high variance (bad gradients)

Reparameterisation trick assumptions

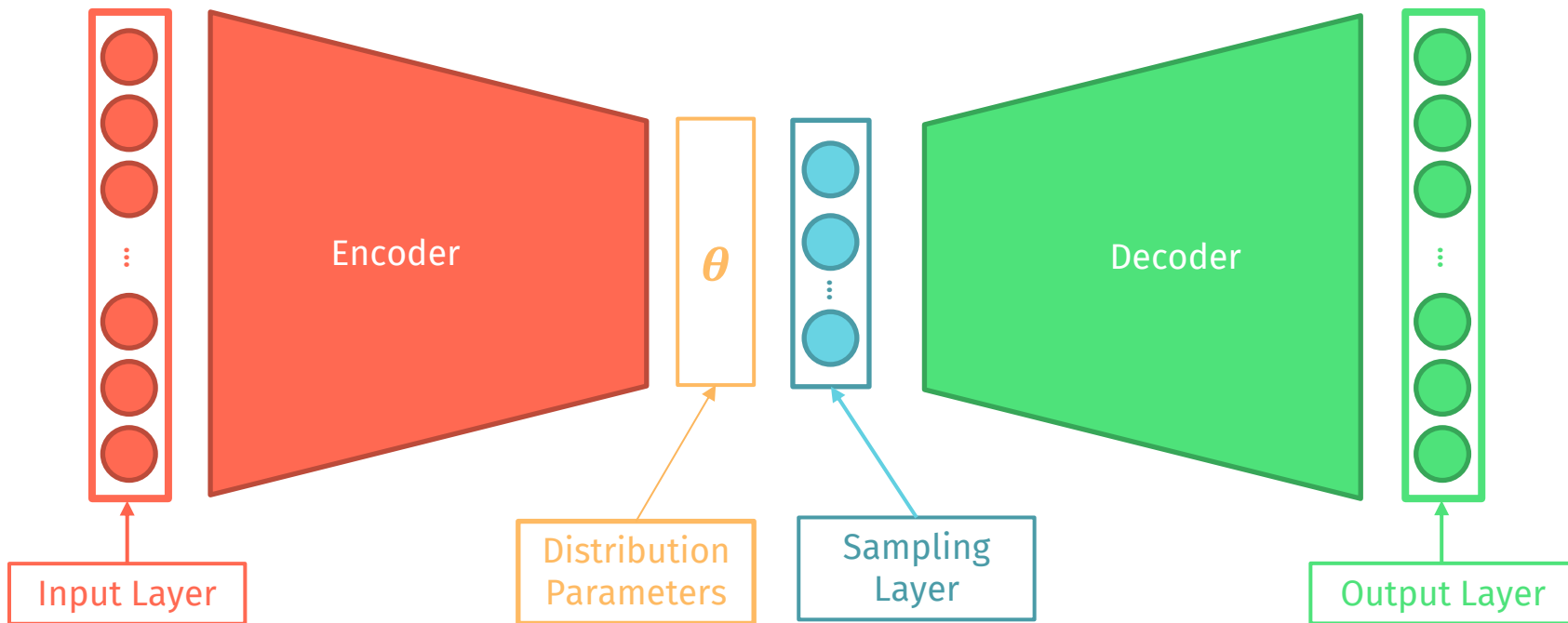
How to use the reparameterization trick in practice

Eligible examples

- Location/Scale family of distributions
If $q \sim N(\mu, \sigma)$ then we can use $p' \sim N(0,1)$ with $g_\theta(\epsilon, x) = \sigma\epsilon + \mu$
- Invertible and tractable CDF (inverse transform sampling)
e.g, exponential distribution
- Composition
e.g., gamma function as a sum of exponentially distributed variables
- Approximations to the inverse CDF

General Architecture

An **encoder** estimating **parameters*** and a **decoder** using **samples** from a distribution



* θ is a parameter **vector or matrix** of dimension equal to the one of the latent space

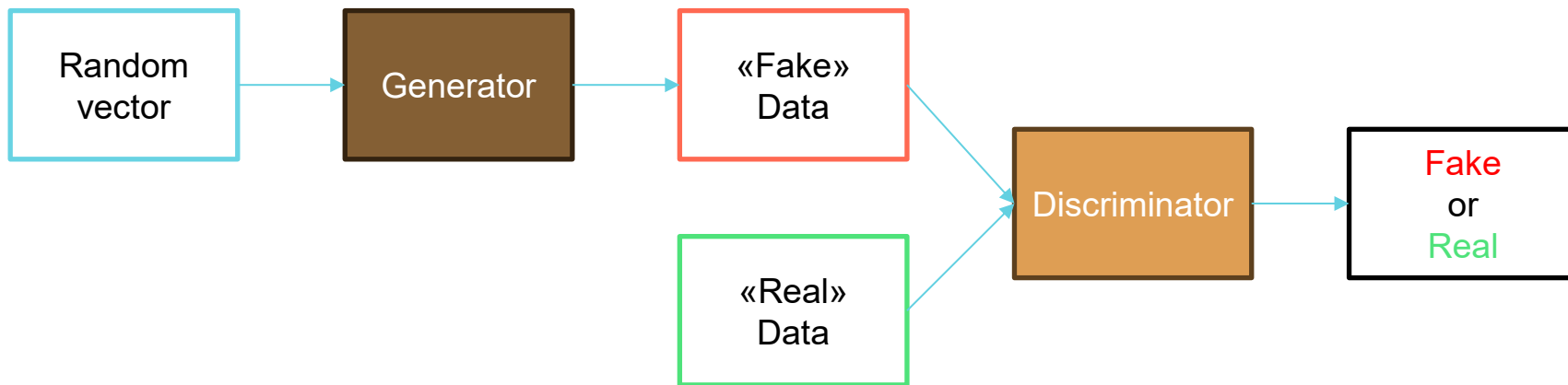
GANs

Generative Adversarial
Networks



General principle

The **discriminator** decides whether the input is **generated (fake)** or **“real”** (observed)



The **generator** tries to fool the **discriminator**

Jensen-Shannon Divergence

Kullback-Leibler

- KL divergence is asymmetric

$$KL(p||q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$$

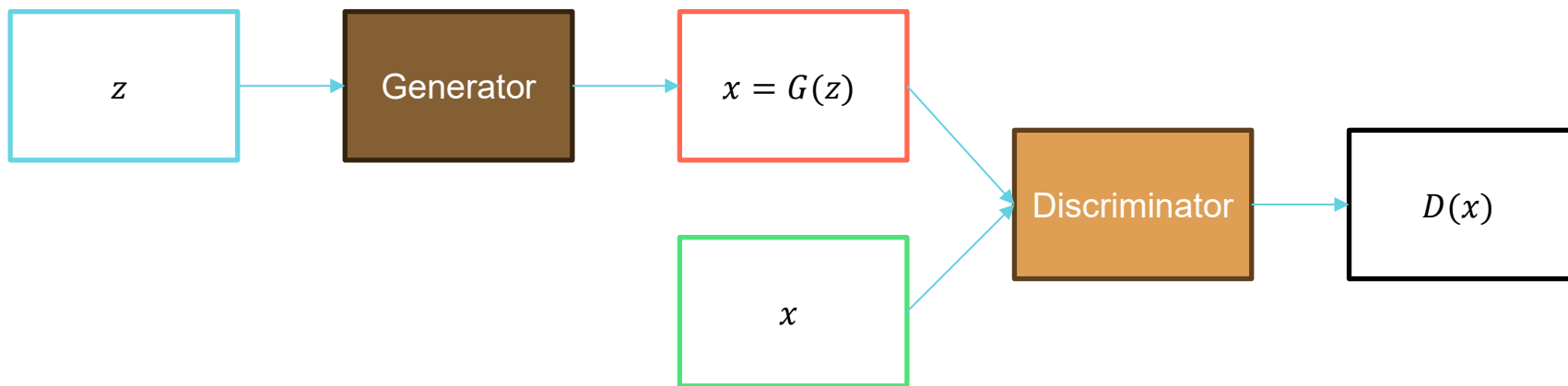
Jensen-Shannon

- JS divergence is symmetric

$$JS(p||q) = \frac{1}{2} KL(p||\frac{p+q}{2}) + \frac{1}{2} KL(q||\frac{p+q}{2})$$

Mathematical formulation

The **discriminator** decides whether the input is **generated (fake)** or **“real” (observed)**



The **generator** tries to fool the **discriminator**

Defining the function to optimise

“Loss” function

$$\begin{aligned}L(G, D) &= \mathbb{E}_{x \sim p_{real}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log(1 - D(G(z)))] \\ &= \mathbb{E}_{x \sim p_{real}} [\log D(x)] + \mathbb{E}_{x \sim p_{fake}} [\log(1 - D(x))]\end{aligned}$$

- p_{real} is the distribution of genuine data and p_{fake} the distribution of fake data

Discriminator

- Loss function – cross-entropy loss for task of classifying real vs. fake images

$$-L(G, D) \approx L_D = BCE_{loss}(1, D(x)) + BCE_{loss}(0, D(G(z))) = \sum_{i=1}^n y_i \log(D(x_i)) + (1 - y_i) \log(1 - D(x_i))$$

Generator

- One possible cost function for the generator: the opposite of the discriminator's $L(G, D) \approx L_G = -L_D$
- Minimax formulation with the generator and discriminator playing a zero-sum game against each other:

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

Optimal values

Discriminator

- Let's optimize $L(G, D)$ with regards to D

$$L(G, D) = \int_x (p_{real}(x) \log(D(x)) + p_{fake}(x) \log(1 - D(x))) dx$$

- We write $f(\tilde{x}) = p_{real}(x) \log(\tilde{x}) + p_{fake}(x) \log(1 - \tilde{x})$ and calculate $\frac{df(\tilde{x})}{d\tilde{x}}$

$$\frac{df(\tilde{x})}{d\tilde{x}} = \frac{1}{\ln(10)} \left(p_{real}(x) \frac{1}{\tilde{x}} - p_{fake}(x) \frac{1}{1 - \tilde{x}} \right)$$

- The optimum is reached for $\frac{df(\tilde{x})}{d\tilde{x}} = 0$ which gives us

$$D^*(x) = \tilde{x}^* = \frac{p_{real}(x)}{p_{real}(x) + p_{fake}(x)}$$

Optimal values

Generator

- The generator is trained to its optimal when $p_{real}(x) \approx p_{fake}(x)$ meaning that

$$D^*(x) = \tilde{x}^* = \frac{p_{real}(x)}{p_{real}(x) + p_{fake}(x)} \approx \frac{1}{2}$$

Global optimum

- When both the generator and the discriminator are at their optimal values

$$L(G^*, D^*) = \int_x (p_{real}(x) \log(D^*(x)) + p_{fake}(x) \log(1 - D^*(x))) dx$$

$$L(G^*, D^*) = \log\left(\frac{1}{2}\right) \int_x (p_{real}(x) + p_{fake}(x)) dx = -2 \log 2$$

The link with JS divergence

Computing the divergence

- According to the formulation of JS divergence defined earlier we have for $p_{real}(x)$ and $p_{fake}(x)$

$$\begin{aligned} JS(p_{real}||p_{fake}) &= \frac{1}{2}KL(p_{real}||\frac{p_{real}+p_{fake}}{2}) + \frac{1}{2}KL(p_{fake}||\frac{p_{real}+p_{fake}}{2}) \\ &= \frac{1}{2} \int_x \left(p_{real}(x) \log \frac{p_{real}(x)}{\frac{p_{real}(x)+p_{fake}(x)}{2}} + p_{fake}(x) \log \frac{p_{fake}(x)}{\frac{p_{real}(x)+p_{fake}(x)}{2}} \right) dx \\ &= \frac{1}{2} (2\log 2 + \int_x p_{real}(x) \log D^*(x) + p_{fake}(x) \log(1 - D^*(x)) dx) \\ &= \frac{1}{2} (2\log 2 + L(G, D^*)) \end{aligned}$$

- Which can also be written as

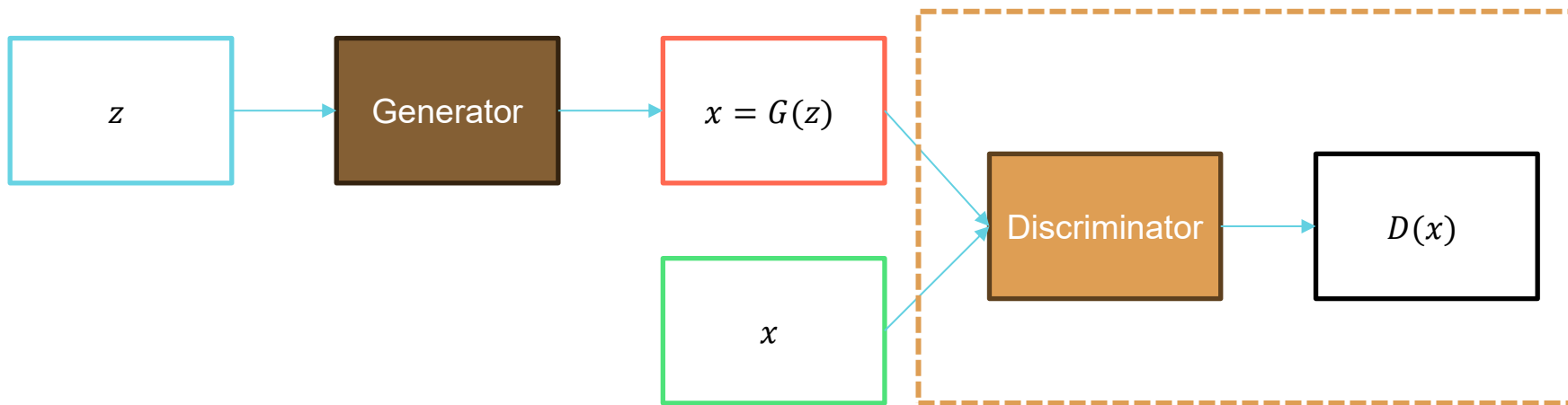
$$L(G, D^*) = 2JS(p_{real}||p_{fake}) - 2\log 2$$

- Meaning that minimum JS coincides perfectly to the case where both G and D are optimal

$$L(G^*, D^*) = -2\log 2$$

Training process

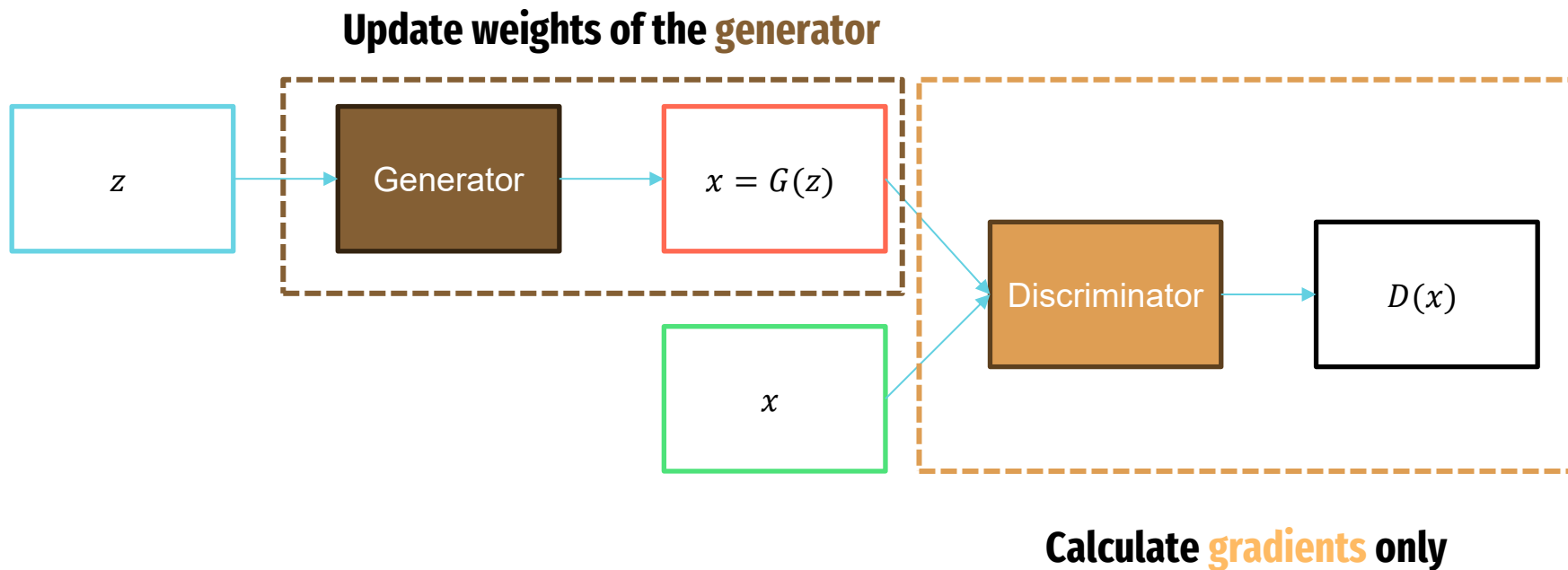
GAN



Update the **discriminator** weights

Training process

GAN



A comment on the loss function

Minimax cost

- A saturation problem can arise when the discriminator is too effective

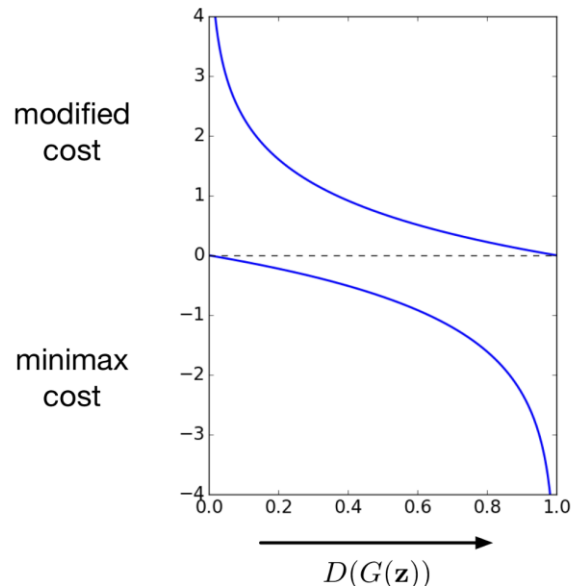
$$L_G \approx cst + \mathbb{E}_{x \sim p_{fake}} [\log(1 - D(x))]$$

Modified cost

- A modified cost can be introduced

$$L'_G \approx cst + \mathbb{E}_{x \sim p_{fake}} [-\log(D(x))]$$

- It fixes the saturation problem shown on the chart



Final note on game theory

GAN

Nash equilibrium

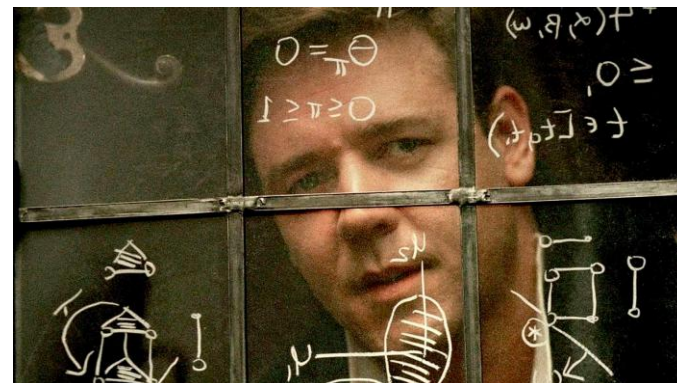
- With GANs, two models are trained simultaneously to find a Nash equilibrium in a two-player non-cooperative game.

Problem

- Each model updates its cost independently with no consideration for the other player in the game

Conclusion

- Updating the gradient of both models concurrently cannot guarantee convergence in GANs



Other famous GANs

- **Conditional GANs**

- Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784.

- **Wasserstein GANs**

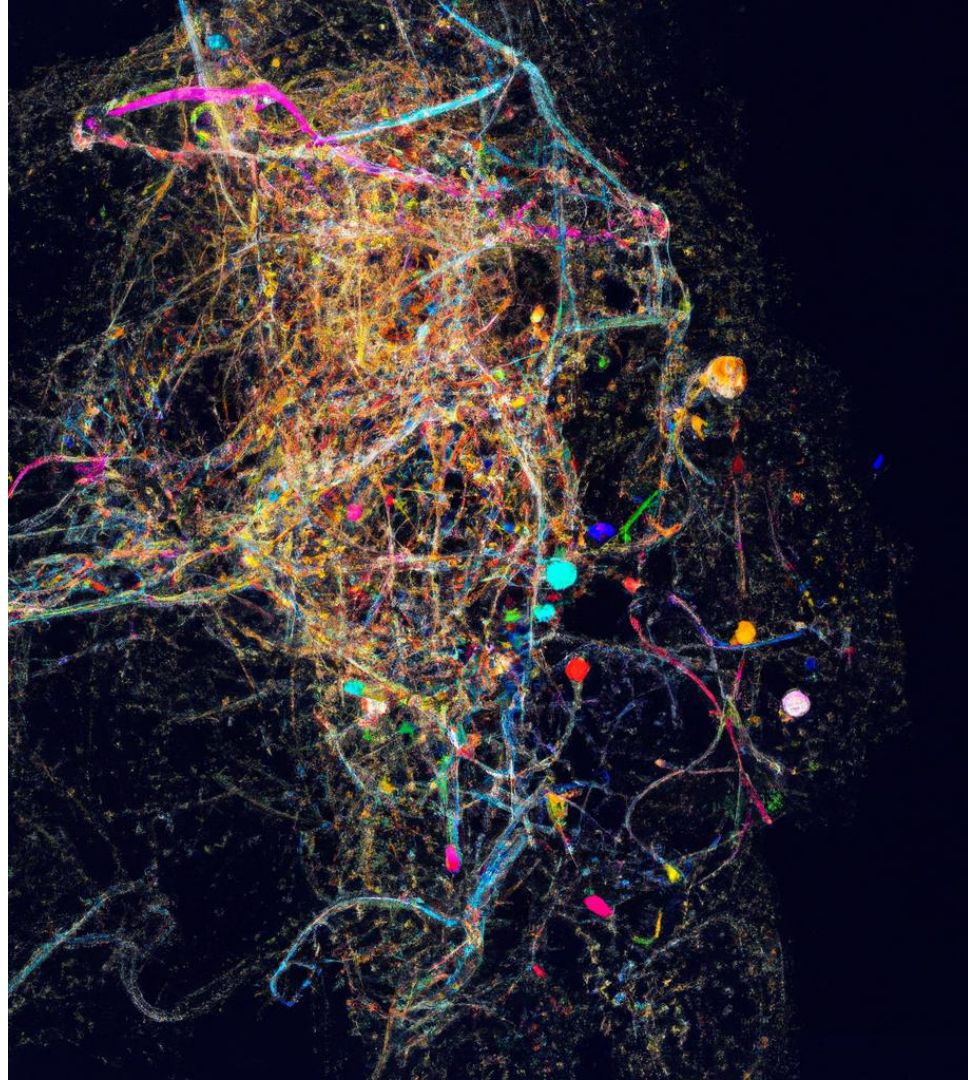
Arjovsky, M., Chintala, S., & Bottou, L. (2017, July). Wasserstein generative adversarial networks. In International conference on machine learning (pp. 214-223). PMLR.

- **Cycle GANs**

Zhu, J. Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In Proceedings of the IEEE international conference on computer vision (pp. 2223-2232).

Diffusion Models

State of the art



General principle

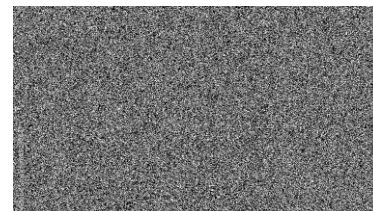
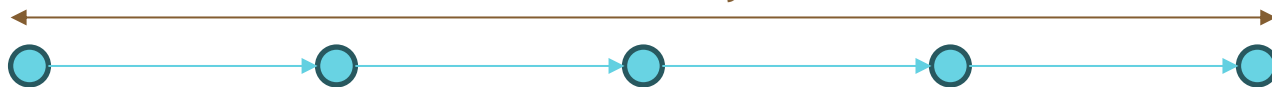
Inspired from **non-equilibrium thermodynamics**

- In physics, diffusion processes are non-reversible
- As the diffusion process progresses, we lose information which can be interpreted as noise being added along the way
- In Deep Learning we attempt to revert this diffusion process by learning how to reconstruct the images in spite of the noise being carried along the way
- Noise is applied to the images using a Markov Chain process

Adding noise along the Markov Chain

Diffusion Models

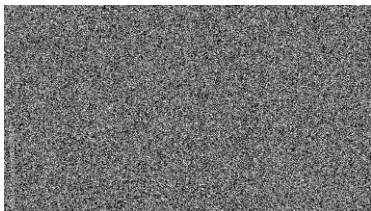
> 100 or 1000 layers



The goal is to learn the **inverse transform** of **image** to noise

After training

Noise can generate **high resolution images**



Which noise to use?

Diffusion Models

Gaussian noise (as usual)

Unet

