Generative AI - Basics 1

Alasdair Newson
anewson@isir.upmc.fr

Gen AI School

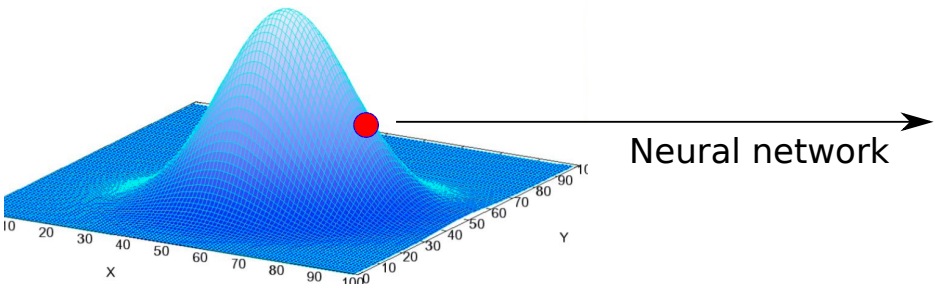
Introduction

About me

- Alasdair Newson, Lecturer, MLIA team, ISIR lab (Sorbonne Université);
- Research subjects: **image restoration and editing**, deep generative models;

Introduction

- Deep generative models are functions/neural networks (NNs) which can **synthesise random examples of complex data**
  - Input : random vector
  - Output : random image (or data)



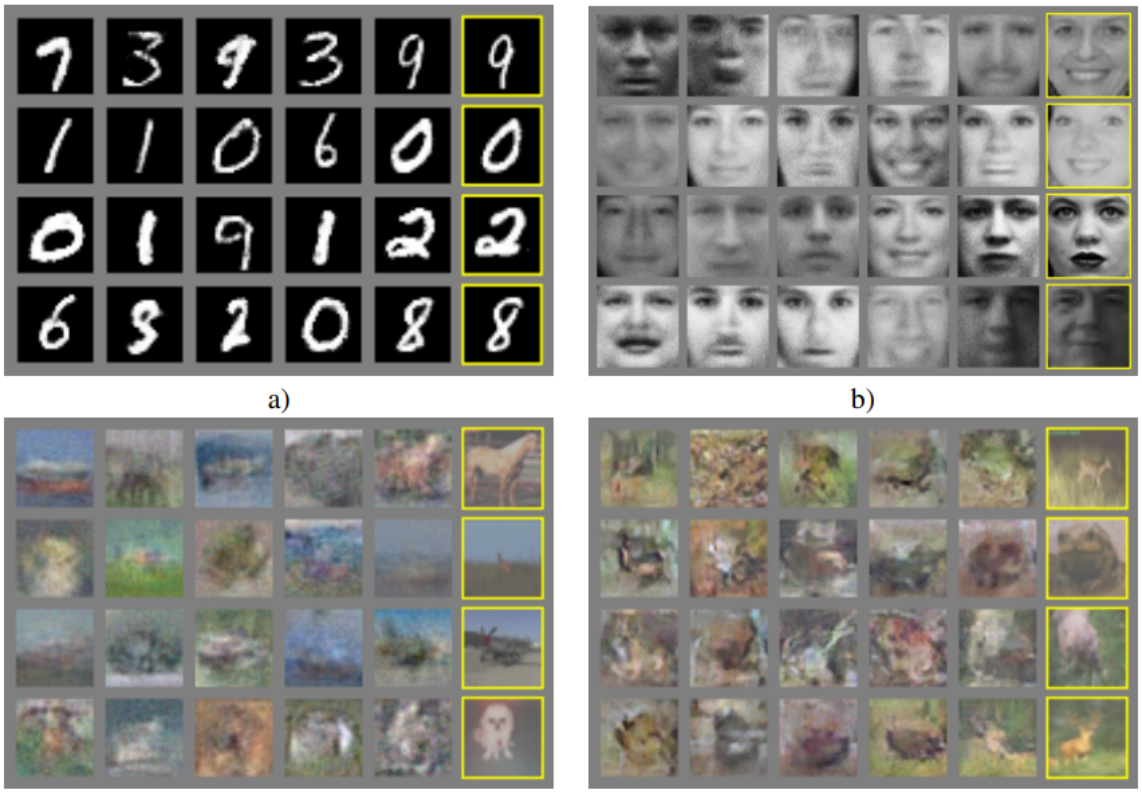
Random vector

Synthesis of random image

- Main questions we look at today:
  - Three main architectures : Variational autoencoders, GANs, Diffusion Models;
  - **Architectures? Training**?

A very brief history

- Generative models took off in 2015;
- Since then, they have gone from this*:



* ***Generative Adversarial Nets***, *Goodfellow et al, NIPS 2014*

Introduction

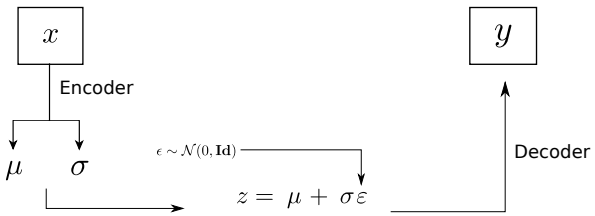
- To this* !

* ***High-resolution image synthesis with latent diffusion models***, *Rombach et al, CVPR 2022*

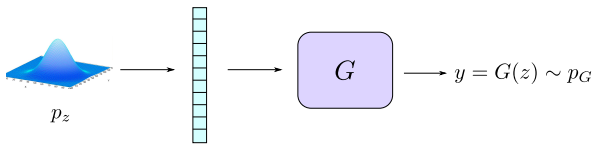
Introduction

- Currently, three main popular deep generative models :
  - Variational Autoencoder;
  - Generative Adversarial Network;
  - Diffusion models;

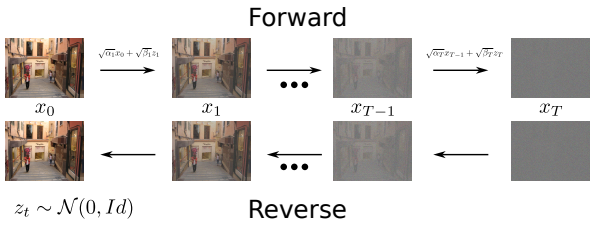
Variational autoencoder



Generative Adversarial Network



Diffusion Model



- The first two (and the third sometimes) rely on the concept of a **latent space**;
  - We will look at this shortly;

Introduction

Mathematical/statistical tools required

- Neural networks, standard architectures;
- Probability distributions, conditional probability;
- Bayes's theorem;

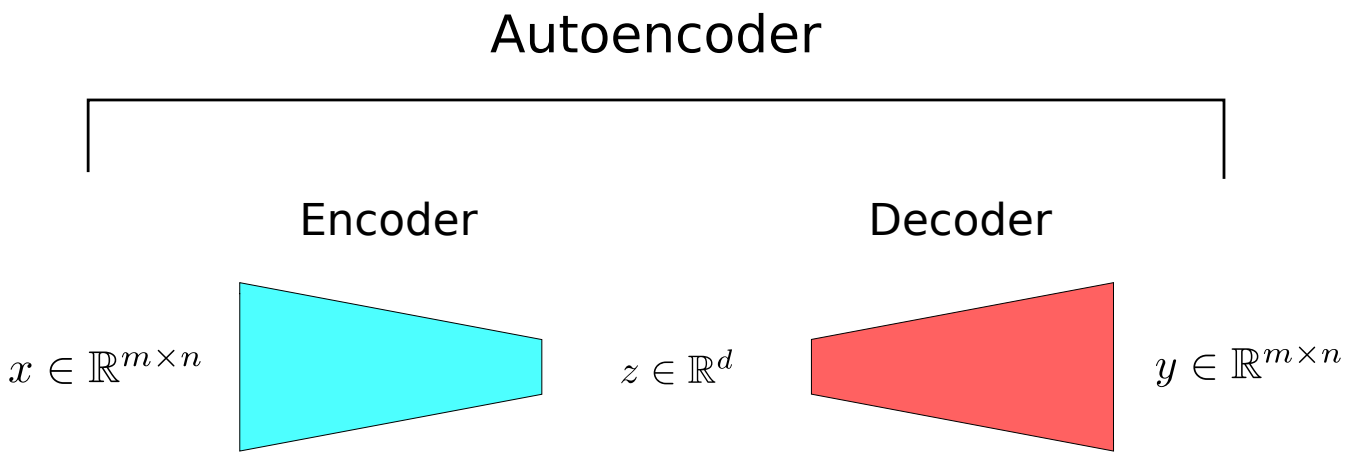
- Variational Bayes, Evidence of Lower Bound;
- Kullback-Leibler divergence;

- We consider the subjects in blue to be known;
- Those in red will be explained in the lesson;

Autoencoders and the latent space

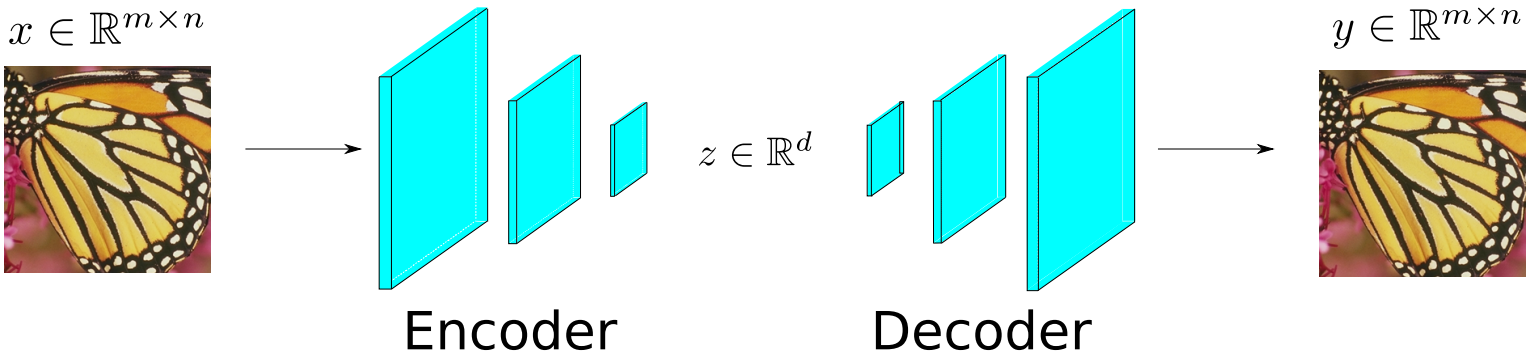
Autoencoders

- Autoencoders consist of two networks : an **encoder** and a **decoder**
  - Encoder : map data x to a smaller **latent space**
  - Decoder : map point z back from latent space to original data space
- Main idea : the latent space is a space where it is **easier to manipulate/understand data**
- **Information bottleneck** : more powerful and compact representation of data



Autoencoders

- The autoencoder is trained to minimise some norm between the input x and the output y of the decoder
- In almost all cases, we have $d << mn$
- This forces the autoencoder to learn a compact and powerful latent space



Autoencoders - some notation

- An AE is a neural network consisting of two sub-networks
  - The encoder f,

$$f : \mathbb{R}^{mn} \to \mathbb{R}^d$$
$$x \mapsto f(x) = z$$

  - The decoder g,

$$g : \mathbb{R}^d \to \mathbb{R}^{mn}$$
$$z \mapsto g(z) = y$$

Autoencoders - some notation

- An AE is a neural network consisting of two sub-networks
  - The encoder f,

$$f : \mathbb{R}^{mn} \to \mathbb{R}^d$$
$$x \mapsto f(x) = z$$

  - The decoder g,

$$g : \mathbb{R}^d \to \mathbb{R}^{mn}$$
$$z \mapsto g(z) = y$$

- As in other neural networks, the main components of AEs are **mlp's/convolutions**, **biases** and **non-linearities**;
  - Do whatever you want to create an information bottleneck

Autoencoders

- Autoencoder is trained to reproduce the input x as an output y, having gone through the bottleneck of the network;

Autoencoding training minimisation problem

$$\begin{aligned}\mathcal{L}(x) &= \|y - x\|_2^2 \\ &= \|g \circ f(x) - x\|_2^2 \\ &= \sum_i^m \sum_j^n \Big((g \circ f(x))_{i,j} - x_{i,j} \Big)^2\end{aligned}$$

- Put simply : **output should look like input !**

Making an Information Bottleneck

- Architecture: encoder should **reduce** tensor size, decoder should **increase** tensor size. How is this done ?

Making an Information Bottleneck

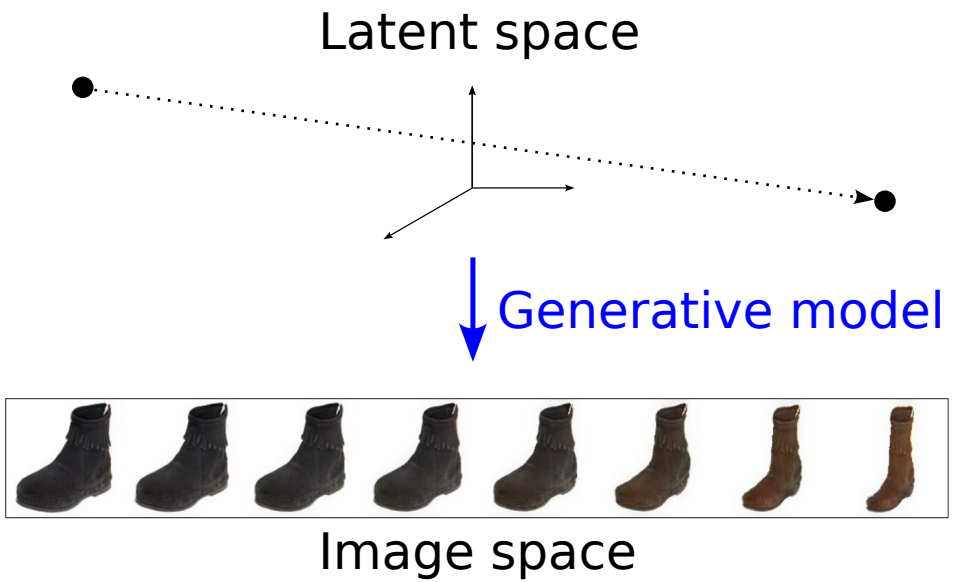
- Architecture: encoder should **reduce** tensor size, decoder should **increase** tensor size. How is this done ?

1. MLPs do this naturally : simply choose the output vector size;
2. Convolutions : stride, deconvolution (convolution + upsampling);
3. Upsampling (bilinear, bicubic etc);
4. Max pool, average pool, unpool;

- We do not explain these here;

Autoencoders

- Important idea: latent space should correspond to **semantic or visually meaningful attributes** of image content



Interpolation in the latent space of a generative model[†]

[†] ***Generative Visual Manipulation on the Natural Image Manifold***, *J-Y. Zhu, P. Krähenbühl, E. Schechtman, A. Efros, CVPR 2016*

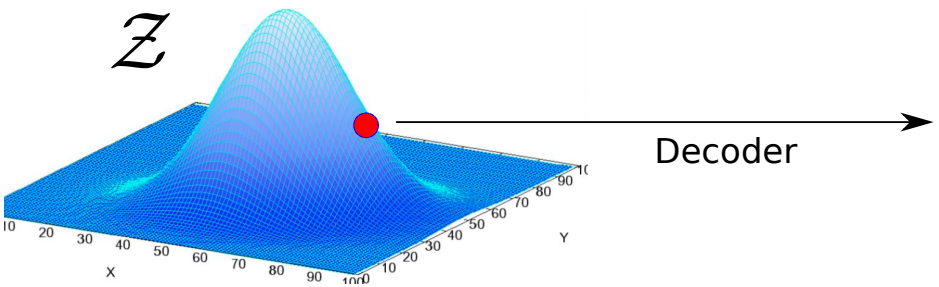
Variational autoencoders

Variational autoencoder

- Suppose we want to use an autoencoder to produce **random examples of data**, how would we go about this ?

Variational autoencoder

- Suppose we want to use an autoencoder to produce **random examples of data**, how would we go about this ?
- We can encourage the data in an Autoencoder's **latent space** to follow a **probability distribution**;
- Synthesis will then consist of :
  1. Sampling in the latent space
  2. Decoding to produce the random image



Probabilistic model in latent space

Synthesis of random image

Variational autoencoder - variational Bayesian approach

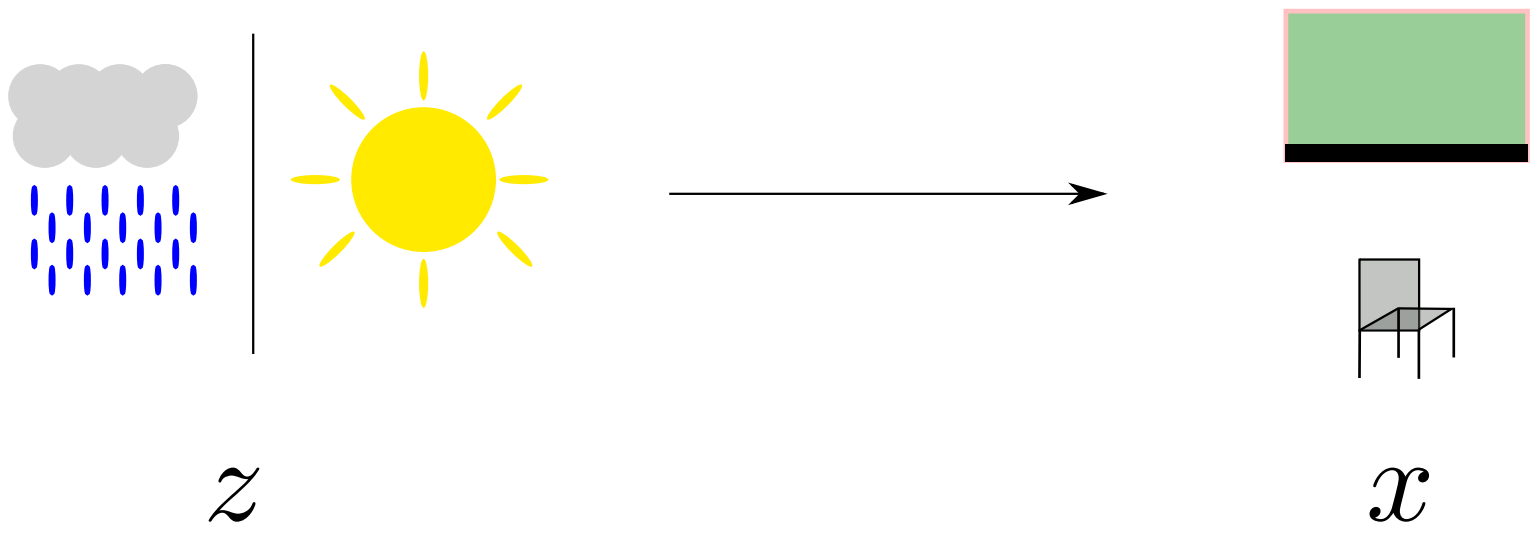
- The Variational Autoencoder (VAE) encourages the latent code z to follow a certain distribution, via the loss function
- This is in turn achieved by using a **Variational Bayesian** approach

Variational autoencoder - variational Bayesian approach

- The Variational Autoencoder (VAE) encourages the latent code z to follow a certain distribution, via the loss function
- This is in turn achieved by using a **Variational Bayesian** approach
- The Variational Bayesian approach is a methodology to approximate the **posterior distribution** of unobserved variables in graphical models
- We will need some tools from statistics : conditional probability, Bayes theorem, marginal distributions, distribution divergences ...
- Keep in mind that the main goal here is to **choose a loss function which will encourage the data in the latent space to follow a probability distribution**

Variational autoencoder - variational Bayesian approach

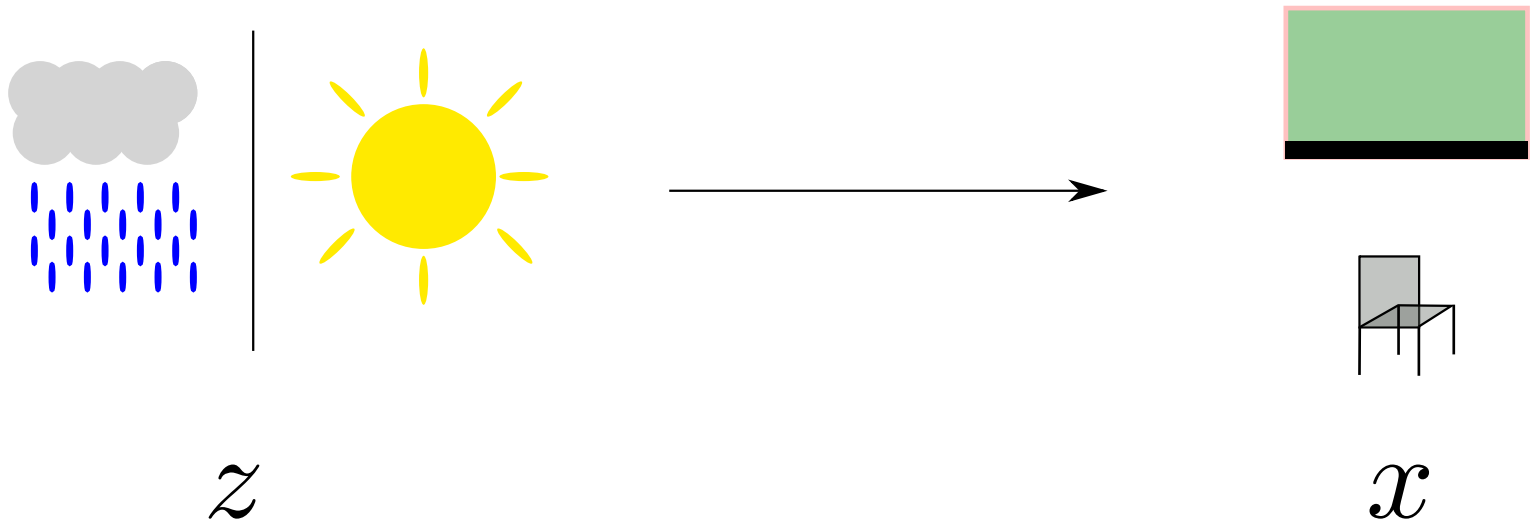
- Let's take an example. Suppose we have a student A, who comes to lessons or not. The event of A's presence in the lesson is x. This is the **observed variable**
- We also know that sometimes it rains, and that A's presence in the class depends on whether it rains or not. Let us denote the event of it raining with z. This is the **latent variable**
- Suppose that we do not directly know whether it is raining (we cannot look out of the window), but can only observe whether the student A is in the lesson



z x

Variational autoencoder - variational Bayesian approach

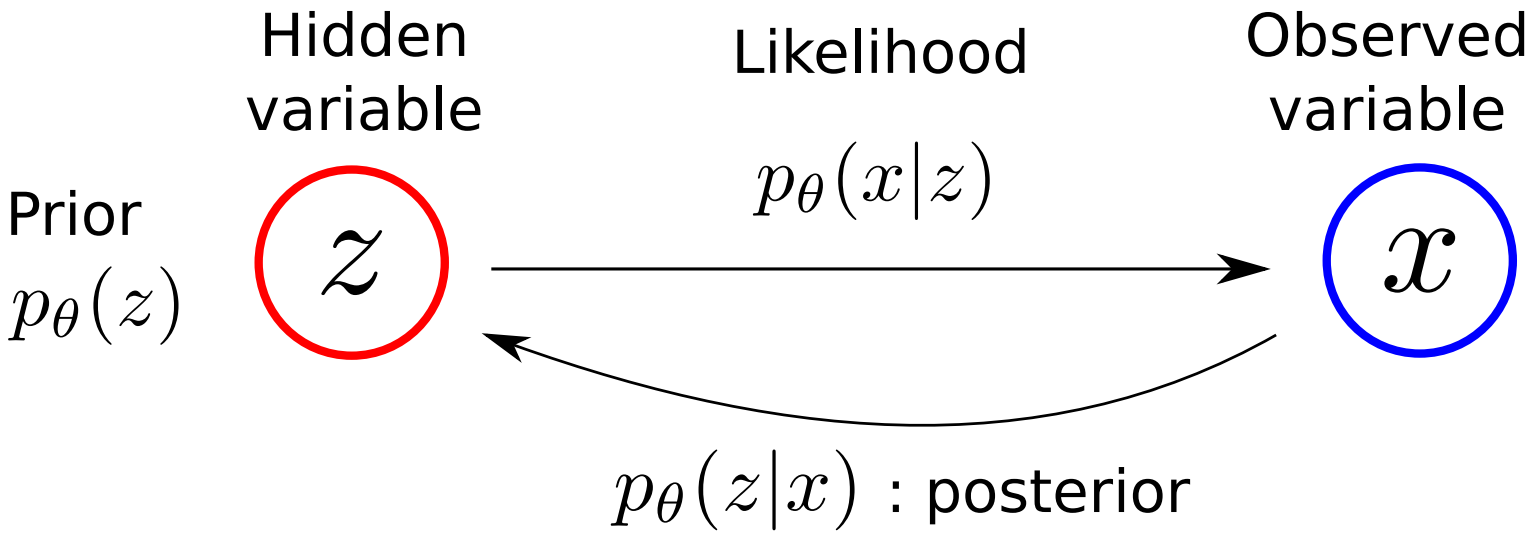
- The probability of A being in the lesson, $\mathbb{P}(x)$, is the **marginal probability**
- The probability of it raining $\mathbb{P}(z)$ is the **prior probability**
- The probability of A being in the lesson given z, $\mathbb{P}(x|z)$ is the **likelihood**
- The probability of it raining, given x, $\mathbb{P}(z|x)$ is the **posterior probability**



z x

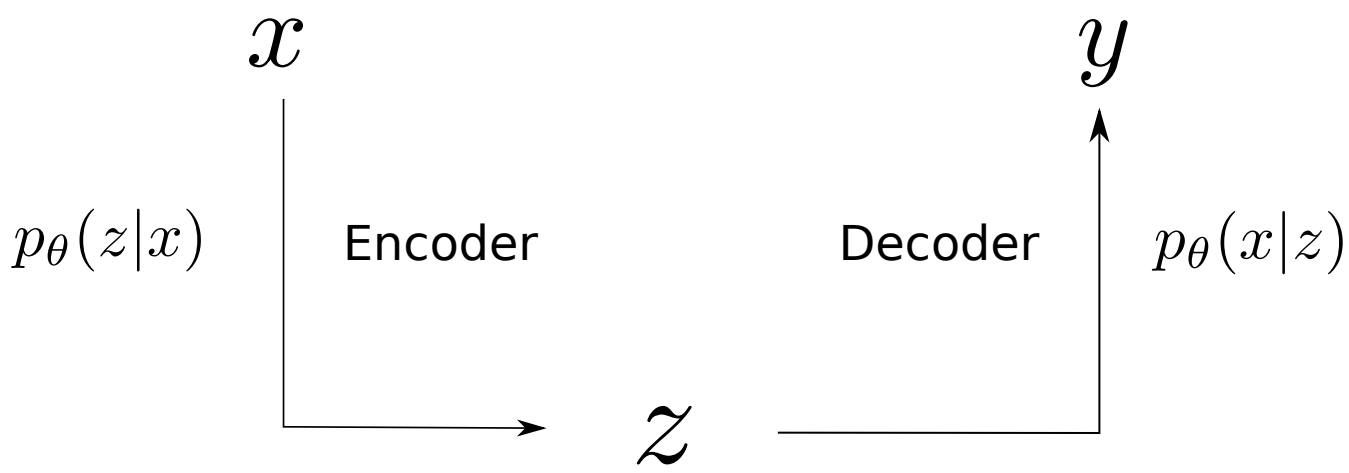
Variational autoencoder - variational Bayesian approach

- At this point, we assume that all the probabilities discussed have a **probability density function**
- We suppose that the distributions come from families of distrbutions parameterised by the parameters θ ($p_\theta(x)$, $p_\theta(x|z)$ etc)



Variational autoencoder - variational Bayesian approach

- There are some clear analogies with the autoencoder
- Encoder : posterior $p_\theta(z|x)$
- Decoder : likelihood $p_\theta(x|z)$



- We want to **impose the prior** $p_\theta(z)$!!

Variational autoencoder - variational Bayesian approach

- Often, we know (or we can at least estimate) the likelihood $p_\theta(x|z)$
- However, often the posterior distribution $p_\theta(z|x)$ is difficult to calculate, or intractable. Why is this the case ?

Variational autoencoder - variational Bayesian approach

- Often, we know (or we can at least estimate) the likelihood $p_\theta(x|z)$
- However, often the posterior distribution $p_\theta(z|x)$ is difficult to calculate, or intractable. Why is this the case ?

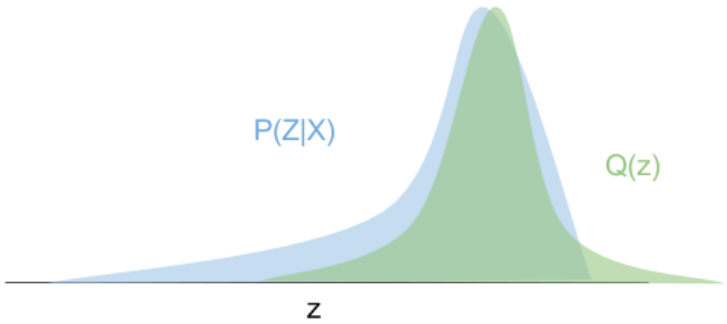
$$p_\theta(z|x) = \frac{p_\theta(x|z)\ \ p_\theta(z)}{p_\theta(x)} \qquad \text{Bayes's rule}$$

$$= \frac{p_\theta(x|z)\ \ p_\theta(z)}{\int p_\theta(x,z)dz} \qquad \text{Marginal distribution}$$

- $\int p_\theta(x,z)dz$ can be a very high-dimensional integral
- Calculating the posterior probability is known as the **inference problem**

Variational autoencoder - variational Bayesian approach

- So, how do we approach the problem of inference ?
- The variational Bayesian approach consists in using an **approximate distribution** $q_\phi(z|x) \approx p_\theta(z|x)$, which is easier to manipulate



- What does it mean for two probability distributions to be **similar** ?
  - Often, this is defined using the **Kullback-Leibler divergence**

Variational autoencoder

Kullback-Leibler divergence

Let p and q be two probability distributions defined over the same domain. The Kullback-Leibler divergence is defined as

$$KL(p \,||\, \ q) = \int p(x) \log \frac{p(x)}{q(x)} dx \tag{1}$$

Variational autoencoder

Kullback-Leibler divergence

Let p and q be two probability distributions defined over the same domain. The Kullback-Leibler divergence is defined as

$$KL(p \mid\mid q) = \int p(x) \log \frac{p(x)}{q(x)} dx \tag{1}$$

The Kullback-Leibler divergence has some interesting properties :

- Non-negative : $KL(p \mid\mid q) \geq 0$
- $KL(p \mid\mid q) = 0 \iff p = q$ almost everywhere
- Non-symmetric : $KL(p \mid\mid q) \neq KL(q \mid\mid p)$

The second point is quite important. Why ? Because we know that **by minimising the KL divergence we are necessarily forcing p and q closer together**.

Variational autoencoder

- Let's come back to variational Bayesian methods now. We wish to approximate $p_\theta(z|x)$ with $q_\phi(z|x)$. To do this, we will find a $q_\phi*$:

$$q_\phi^* = \arg\min_{q_\phi} KL\left(q_\phi(z|x) \,||\, p_\theta(z|x)\right) \,^\dagger \tag{2}$$

- Unfortunately, this does not help us much. Why ? **Because we don't know** $p_\theta(z|x)$!

- We will have to minimise $KL\left(q_\phi(z|x) \,||\, p_\theta(z|x)\right)$ some other way

† *You might notice that the q_ϕ before the p_θ. We do not go into the technical reasons here ...*

Variational autoencoders - The evidence lower bound

Evidence of Lower BOund (ELBO, lower bound of $\log p_\theta(x)$)

$$\text{ELBO}(q_\phi) = \mathbb{E}_{q_\phi}\left[\log(p_\theta(x,z))\right] - \mathbb{E}_{q_\phi}\left[\log q_\phi(z|x)\right]$$

- This is known as the "Evidence of Lower BOund" because :

$$\log p_\theta(x) = \text{ELBO}(q_\phi) + KL(q_\phi(z|x) \;\; || \, p_\theta(z|x)))$$

- KL divergence is positive : the ELBO is a lower bound for $\log p_\theta(x)$ (the "evidence")
- **By maximising the ELBO, we minimise the KL divergence between $q_\phi(z|x)$ and $p_\theta(z|x)$**

Variational autoencoders - Variational Autoencoder loss function

- We can rewrite the ELBO in the following manner*

$$\text{ELBO}(q_\phi) = \mathbb{E}_{q_\phi}\left[\log(p_\theta(x|z))\right] - KL(q_\phi(z|x) \,||\, p_\theta(z))$$

- **We know all of these terms!** : we can use it as a VAE loss function!

Variational Autoencoder loss function

$$\mathcal{L}(x;\theta,\phi) = \overbrace{\mathbb{E}_{q_\phi}\left[\log(p_\theta(x|\ z))\right]}^{\text{Reconstruction error}} - \underbrace{KL(q_\phi(z|x) \,||\, p_\theta(z))}_{\text{Enforce the prior distribution}}$$

- **Important note !** : we want to **maximise** this loss function !

* *This is shown at the end of the slides*

Variational Autoencoder summary

1. We modelled the autoencoding process using a probabilistic (Bayesian) framework, with an observed and a hidden variable
2. We wanted to calculate the posterior distribution $p_\theta(z|x)$, but this is complicated
3. We used an approximation $q_\phi(z|x)$ to $p_\theta(z|x)$
4. We used the ELBO as a loss function to minimise $KL(q_\phi(z|x)||p_\theta(z))$
   - Ensures a good reconstruction
   - Encourages the latent space to follow our chosen distribution (the prior $p_\theta(z)$)

Variational Autoencoder

- There remains one more important detail : how to backpropagate through samples of z ? **Random variable, not differentiable**
- Solution : **"reparametrisation trick"**, make the random element an network input

Variational Autoencoder

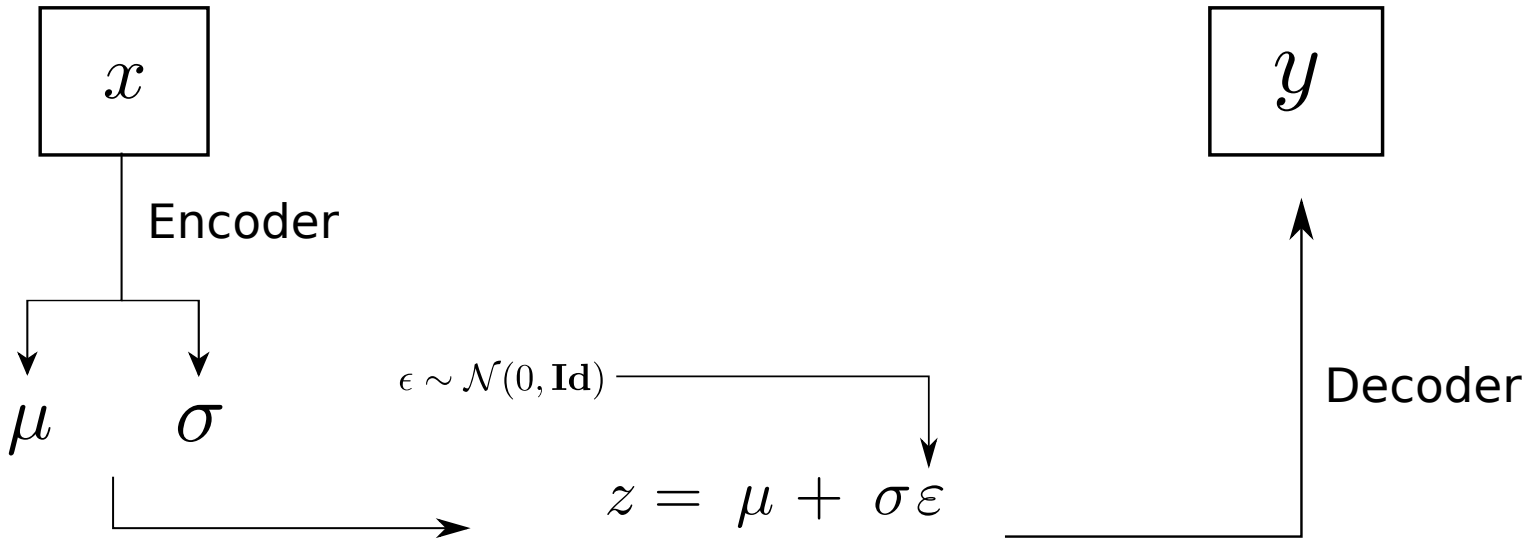
- There remains one more important detail : how to backpropagate through samples of z ? **Random variable, not differentiable**
- Solution : **"reparametrisation trick"**, make the random element an network input
- In the Gaussian case, where q_ϕ is a multivariate Gaussian vector, with mean μ and diagonal covariance matrix $\sigma\mathrm{Id}$, this gives

$$z = \mu + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathrm{Id})$$

- μ and σ are produced by the encoder
- Thus, backpropagation can be carried out w.r.t to the parameters ϕ and θ

Variational autoencoder **in practice**

- The variational autoencoder is actually quite simple to implement
- Take the case of Gaussian $q_\phi(z|x)$

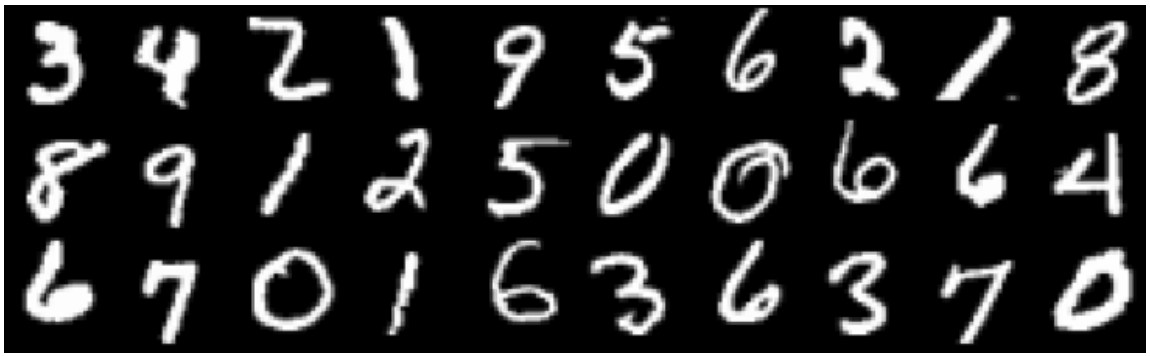


- Encoder and decoder can be MLPs, CNNs ...
- What is the loss in practice ?

Variational autoencoder **in practice**

- Let us take the following case, well-adapted to the **mnist dataset** :
  - Prior : $p_\theta(z) \sim \mathcal{N}(0, \mathrm{Id})$
  - Variational approximation : $q_\phi(z|x) \sim \mathcal{N}(\mu, \sigma \mathrm{Id})$, where $(\mu, \sigma) = \Phi_e(x)$
  - Likelihood : $p_\theta(x|z) \sim Ber(y)$, where $y = \Phi_d(z)$

$$\mathcal{L} = \overbrace{\sum_{i=1}^{mn} x_i \log y_i + (1 - x_i) \log(1 - y_i)}^{\text{Reconstruction error}} - \underbrace{\frac{1}{2} \sum_{j=1}^{d} \left(\mu_j^2 + \sigma_j^2 - 1 - \log\left(\sigma_j^2\right) \right)}_{\text{KL divergence}}$$



Variational autoencoder results

- Some results of variational autoencoders on mnist data : random samples

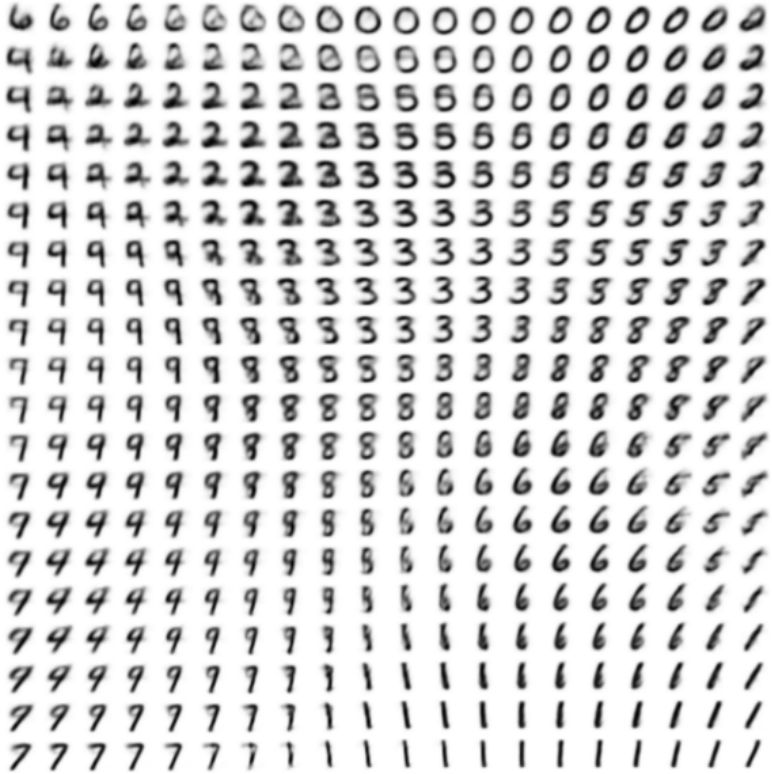
(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

Auto-Encoding Variational Bayes, *D. P. Kingma, M. Welling, arXiv preprint arXiv:1312.6114, 2013*

Variational autoencoder results

- Some results of VAEs on mnist, face data : uniform samples

(a) Learned Frey Face manifold



(b) Learned MNIST manifold

Auto-Encoding Variational Bayes, *D. P. Kingma, M. Welling, arXiv preprint arXiv:1312.6114, 2013*

Variational autoencoder results

- Some results of VAEs on more complex digits data

Variational autoencoder results

- Very recent results of VAEs on more complex digits data

NVAE: A Deep Hierarchical Variational Autoencoder, *A. Vahdat, J. Kautz, arXiv preprint arXiv:2007.03898, 2020*

Variational Autoencoders : summary

- Rigourous framework to autoencode data onto a probabilisitcally modelled latent space
- Advantages
  - Theoretically-motivated, loss function meaningful
  - Learn to and from mapping (encoder and decoder)
- Drawbacks
  - Have to re-write loss function for each different model, not always easy
  - In practice, do not produce as complex examples as **Generative Adversarial Networks**, which we will see next week

Generative Adversarial Networks

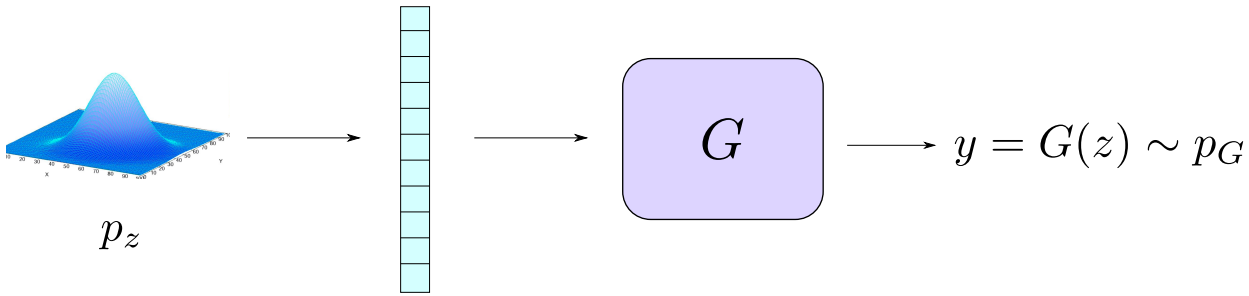
Generative Adversarial networks

- We saw that variational autoencoders were not straightforward to adapt to new situations
- **Generative adversarial networks** (GANs)* are another generative model that generate random examples of high-dimensional data

* ***Generative Adversarial Nets***, *Goodfellow et al, NIPS 2014*

Generative Adversarial networks

- The GAN contains only the **decoder** part of an autoencoder
  - The code z is **explicitly sampled** from a chosen distribution p_z (contrary to the VAE)
- The decoder is referred to here as the "Generator"



- We suppose that the data in the databse follows a distribution p_{data}
- We want to make the distribution of $y = G(z)$, p_G, similar to p_{data}*

* *Why can we not do this via the KL divergence as before ? Too high dimensionality (previously, we worked in the latent space)*

Generative Adversarial networks

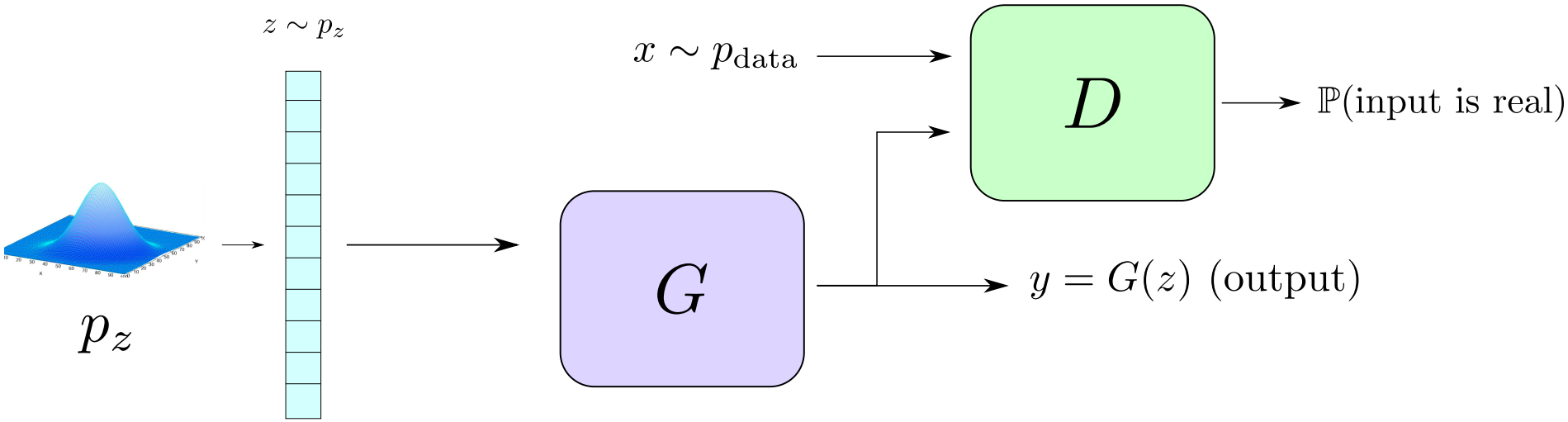
- However, with no reconstruction error, how do we make x look like the data ?
- Answer : Train another network : a **Discriminator** D (or "Adversarial Network")

Generative Adversarial networks

- However, with no reconstruction error, how do we make x look like the data ?
- Answer : Train another network : a **Discriminator** D (or "Adversarial Network")
- $D : \mathbb{R}^{mn} \rightarrow [0, 1]$ is trained to **identify "good" (or "true")** examples of the data
- $G : \mathbb{R}^{z} \rightarrow \mathbb{R}^{mn}$ is trained to **produce realistic data examples**
- The two networks are trained at the same time, and **each try to fool the other !**

Generative Adversarial networks

- The full GAN architecture looks like this



Generative Adversarial networks

- The discriminator is a really interesting idea, why ?
- Reliable and powerful image/data models are difficult to establish
- It is difficult to say whether an image is "good" or not
  - The discriminator acts as a **learned image norm** !
  - It can be used for other purposes also ...
- How is this is achieved ? Via a well-designed loss function

Generative Adversarial networks

GAN loss

- Train generator G and the discriminator D in a minimax optimisation problem

$$\min_G \max_D \overbrace{\mathbb{E}_{x\sim p_{data}}\left[\log D(x)\right]}^{D \text{ is trying to recognize true data}} + \underbrace{\mathbb{E}_{z\sim p_z}\left[\log\left(1-D(G(z))\right)\right]}_{\substack{G \text{ is trying to fool } D, \\ \text{but } D \text{ is trying not to be fooled}}}$$

Generative Adversarial networks

GAN loss

- Train generator G and the discriminator D in a minimax optimisation problem

$$\min_G \max_D \overbrace{\mathbb{E}_{x\sim p_{data}}\left[\log D(x)\right]}^{D \text{ is trying to recognize true data}} + \underbrace{\mathbb{E}_{z\sim p_z}\left[\log\left(1-D(G(z))\right)\right]}_{\substack{G \text{ is trying to fool } D, \\ \text{but } D \text{ is trying not to be fooled}}}$$

- Minimisation w.r.t G
  - Second term is low, $\implies$ $1 - D(G(z))$ is close to 0 $\implies$ D is recognising $G(z)$ as a true data example : G **has fooled** D

GAN loss

- Train generator G and the discriminator D in a minimax optimisation problem

$$\min_G \max_D \overbrace{\mathbb{E}_{x\sim p_{data}}[\log D(x)]}^{D \text{ is trying to recognize true data}} + \underbrace{\mathbb{E}_{z\sim p_z}[\log(1-D(G(z)))]}_{\substack{G \text{ is trying to fool } D, \\ \text{but } D \text{ is trying not to be fooled}}}$$

- Minimisation w.r.t G
  - Second term is low, $\implies 1-D(G(z))$ is close to 0 $\implies$ D is recognising $G(z)$ as a true data example : G **has fooled** D
- Maximisation w.r.t D
  - First term is high $\implies$ $D(x)$ is close to 1 : D **is learning to recognize true data**
  - Second term is high $\implies 1-D(G(z))$ is close to 1 : D **is not getting fooled by** G

Generative Adversarial networks

- At the beginning of the training, the examples from G are not very good : D can spot them easily
- At the end of training, **the discriminator should not be able to tell the true data from the generated data** : $p_G = p_{data}$
- Optimisation alternates between minimisation and maximisation steps

Generative Adversarial networks

- At the beginning of the training, the examples from G are not very good : D can spot them easily
- At the end of training, **the discriminator should not be able to tell the true data from the generated data** : $p_G = p_{data}$
- Optimisation alternates between minimisation and maximisation steps
- Are we sure that this loss is well-designed for this purpose ?
- In fact, we can show that this is the case

Generative Adversarial networks

- First, we establish the following lemma :

Optimal GAN discriminator D^*

For a fixed G, the optimal D is $D^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_G(x)}$

Generative Adversarial networks

- First, we establish the following lemma :

Optimal GAN discriminator D^*

For a fixed G, the optimal D is $D^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_G(x)}$

$$\begin{aligned}\mathcal{L}(G, D) &= \mathbb{E}_{x\ \sim p_{data}}\left[\log D(x)\right] + \mathbb{E}_{z\ \sim p_z}\left[\log(1 - D(G(z)))\right] \\ &= \int_{\mathcal{X}} p_{data}(x)\log(D(x))dx + \int_{\mathcal{Z}} p_z(z)\log(1 - D(G(z)))dz \\ &= \int_{\mathcal{X}} p_{data}(x)\log(D(x)) + p_G(x)\log(1 - D(x))dx.\end{aligned}$$

Generative Adversarial networks

- First, we establish the following lemma :

Optimal GAN discriminator D^*

For a fixed G, the optimal D is $D^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_G(x)}$

$$\begin{aligned}\mathcal{L}(G, D) &= \mathbb{E}_{x\ \sim p_{data}}[\log D(x)] + \mathbb{E}_{z\ \sim p_z}[\log(1 - D(G(z)))] \\ &= \int_{\mathcal{X}} p_{data}(x) \log(D(x)) dx + \int_{\mathcal{Z}} p_z(z) \log(1 - D(G(z))) dz \\ &= \int_{\mathcal{X}} p_{data}(x) \log(D(x)) + p_G(x) \log(1 - D(x)) dx.\end{aligned}$$

- For every x, the maximum of the previous equation w.r.t $D(x)$ is

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$$

Generative Adversarial networks

- Given this lemma, we can now state the main optimality theorem

Global optimum of the GAN loss function

The global optimum of the GAN loss function is achieved if and only if $p_G = p_{data}$. At this point $\mathcal{L}(G, D) = -\log 4$.

- Given this lemma, we can now state the main optimality theorem

Global optimum of the GAN loss function

The global optimum of the GAN loss function is achieved if and only if $p_G = p_{data}$. At this point $\mathcal{L}(G, D) = -\log 4$.

- First, our previous lemma allows us to rewrite the loss function

$$\max_D \mathcal{L}(G, D) = \mathbb{E}_{x\ \sim p_{data}} [\log D^*(x)] + \mathbb{E}_{z\ \sim p_z} [\log(1 - D^*(G(z)))] \tag{3}$$

$$= \mathbb{E}_{x\ \sim p_{data}} [\log D^*(x)] + \mathbb{E}_{x\ \sim p_G} [\log(1 - D^*(x))]$$

$$= \mathbb{E}_{x\ \sim p_{data}} \left[\log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}\right] + \mathbb{E}_{x\ \sim p_G} \left[\log \left(\frac{p_G(x)}{p_{data}(x) + p_G(x)}\right)\right]. \tag{4}$$

Generative Adversarial networks

- Given this lemma, we can now state the main optimality theorem

Global optimum of the GAN loss function

The global optimum of the GAN loss function is achieved if and only if $p_G = p_{data}$. At this point $\mathcal{L}(G, D) = -\log 4$.

- First, our previous lemma allows us to rewrite the loss function

$$\max_D \mathcal{L}(G, D) = \mathbb{E}_{x\ \sim p_{data}} [\log D^*(x)] + \mathbb{E}_{z\ \sim p_z} [\log(1 - D^*(G(z)))] \tag{3}$$

$$= \mathbb{E}_{x\ \sim p_{data}} [\log D^*(x)] + \mathbb{E}_{x\ \sim p_G} [\log(1 - D^*(x))]$$

$$= \mathbb{E}_{x\ \sim p_{data}} \left[\log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}\right] + \mathbb{E}_{x\ \sim p_G} \left[\log \left(\frac{p_G(x)}{p_{data}(x) + p_G(x)}\right)\right]. \tag{4}$$

- Therefore, if $p_G = p_{data}$, then $\mathcal{L}(G, D) = \log \frac{1}{2} + \log \frac{1}{2} = -\log(4)$

Generative Adversarial networks

- Now, we are going to show that $-\log(4)$ is the optimal value of the loss function

Generative Adversarial networks

- Now, we are going to show that $-\log(4)$ is the optimal value of the loss function
- First, we remark that :

$$-\log(4) = \mathbb{E}_{x\ \sim p_{data}}\left[-\log(2)\right] + \mathbb{E}_{x\ \sim p_G}\left[-\log(2)\right]. \tag{5}$$

Generative Adversarial networks

- Now, we are going to show that $-\log(4)$ is the optimal value of the loss function
- First, we remark that :

$$-\log(4) = \mathbb{E}_{x\ \sim p_{data}}\left[-\log(2)\right] + \mathbb{E}_{x\ \sim p_G}\left[-\log(2)\right]. \tag{5}$$

- Therefore, by subtracting Equation 5 from Equation 4, we have

$$\begin{aligned}\mathcal{L}(G, D^*) &= -\log(4) + \int p_{data}(x) \log \frac{p_{data}(x)}{\frac{1}{2}\left(p_{data}(x) + p_G(x)\right)} dx\ + \\ &\qquad \int p_G(x) \log \frac{p_G(x)}{\frac{1}{2}\left(p_{data}(x) + p_G(x)\right)} dx \\ &= -\log(4) + KL\left(p_{data}\ ||\ \frac{p_{data} + p_G}{2}\right) + KL\left(p_G\ ||\ \frac{p_{data} + p_G}{2}\right).\end{aligned}$$

Generative Adversarial networks

- This can also be rewritten as

$$\mathcal{L}(G, D^*) = -\log(4) + 2\,\mathbf{JSD}\left(\mathbf{p_{data}} \,||\, \mathbf{p_G}\right).$$

- The **JSD** is the **Jensen-Shannon divergence**
- This is another distance between distributions. For p and q, we have :

$$JSD(p, q) = \frac{1}{2} KL\left(p \,||\, \frac{1}{2}(p+q)\right) + \frac{1}{2} KL\left(q \,||\, \frac{1}{2}(p+q)\right)$$

Generative Adversarial networks

- This can also be rewritten as

$$\mathcal{L}(G, D^*) = -\log(4) + 2\,\mathbf{JSD}\left(\mathbf{p_{data}} \,||\, \mathbf{p_G}\right).$$

- The **JSD** is the **Jensen-Shannon divergence**
- This is another distance between distributions. For p and q, we have :

$$JSD(p, q) = \frac{1}{2} KL\left(p \,||\, \frac{1}{2}(p+q)\right) + \frac{1}{2} KL\left(q \,||\, \frac{1}{2}(p+q)\right)$$

- The JSD is **non-negative and equal to zero if and only if** $p_{data} = p_G$
- Therefore $-\log(4)$ is the optimal value, and only reached when $p_{data} = p_G$

Generative Adversarial networks

- To summarise, we know that **by minimising the GAN loss, we are sure to encourage p_G to approximate p_{data}**
- In spite of this theoretical result (and others), training GANs is very difficult and is a current hot topic of research

Generative Adversarial networks

- Here are some results of the original GAN paper*

- These original results have been vastly improved on
- There are many, many GAN variants. We present a few now

* ***Generative Adversarial Nets***, *Goodfellow et al, NIPS 2014*

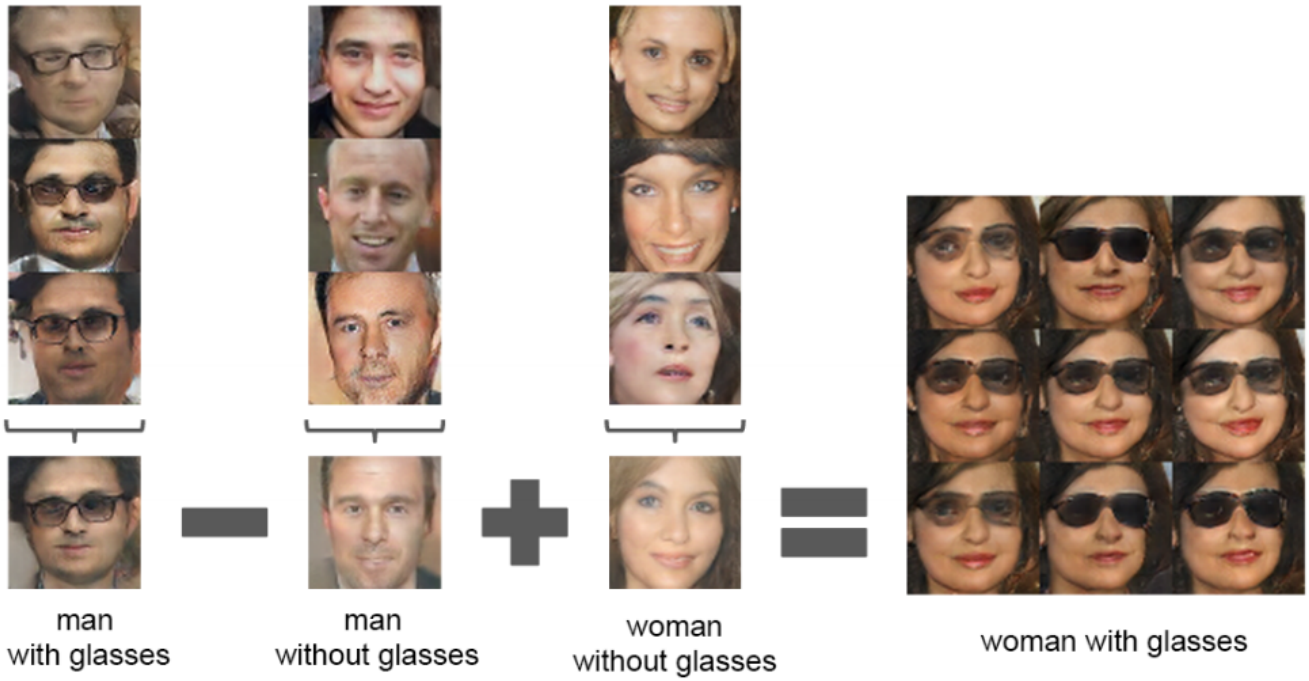
Generative Adversarial networks

- Deep Convolutional Generative Adversarial Networks (DCGAN)
- More complex data, sharper generation

* ***Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks***, *Radford, A. and Chintala, S., arXiv:1511.06434, 2015*

Generative Adversarial networks

- Deep Convolutional Generative Adversarial Networks (DCGAN)
- **Linear algebra in the latent space**



* ***Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks***, *Radford, A. and Chintala, S., arXiv:1511.06434, 2015*

Generative Adversarial networks

- Around 2017, "intermediate"-resolution results were achieved by different research teams†

† ***Progressive growing of gans for improved quality, stability, and variation***, *Karras, T., Aila, T., Laine, S., and Lehtine, J., arXiv preprint arXiv:1710.10196, 2017*

Generative Adversarial networks

- Around 2017, "intermediate"-resolution results were achieved by different research teams†

† ***Progressive growing of gans for improved quality, stability, and variation***, *Karras, T., Aila, T., Laine, S., and Lehtine, J., arXiv preprint arXiv:1710.10196, 2017*

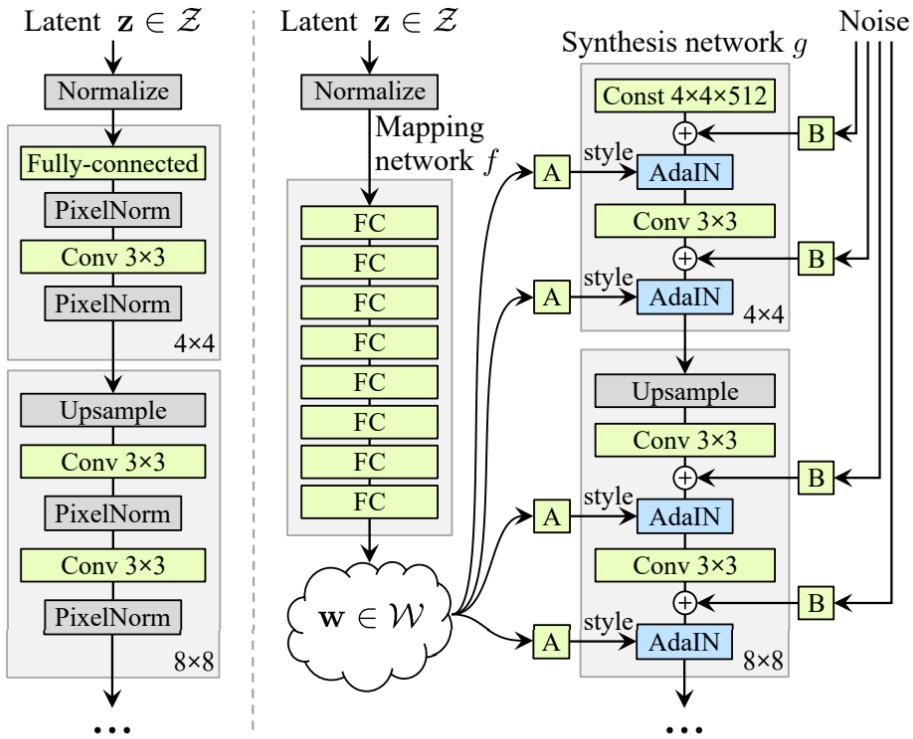
Generative Adversarial networks

- The most recent examples of GANs show realistic results on images of up to $1024 \sim 1024$ pixel resolution
- One of the most impressive is StyleGAN[†] (there is also another, more recent version, StyleGAN2)
- The main idea of StyleGAN is to transform the **probabilistic** latent space into another, **less entangled** and more **linear**
  - "Entanglement" means mixing up several visual attributes in the latent space
  - If this happens, it is difficult to control attributes separately

[†] ***A Style-Based Generator Architecture for Generative Adversarial Networks***, *Karras, T., Laine, S., and Aila, T. CVPR, 2019*

Generative Adversarial networks

- StyleGAN achieves these goals by **inserting the latent code at several resolutions**
- This replaces the purely sequential architecture of classic GANs



(a) Traditional (b) Style-based generator

Generative Adversarial networks

- Some high-resolution examples (from StyleGAN2)

Generative Adversarial networks

- Some high-resolution examples (from StyleGAN2)

Generative Adversarial networks

- Some of the research goals concerning GANs :
  1. Achieving high-resolution results, **removing visual artefacts**
  2. How to **navigate** correctly in the latent space ? What is the **geometry** of the latent space
  3. How to achieve **disentangled respresentations** in the latent space

Summary on GANs

- GANs are a powerful and generic way of producing **random examples of complex images**
- However, they are **notoriously difficult to train**;
  - Diffusion models better in this respect;
- A significant disadvantage with respect to variational autoencoders is that **GANs do not train an encoder** : it is a one-way transformation
  - Often, for restoration or other inverse problems, it is useful to have an "inverse" transformation

Diffusion models

Diffusion Models

- Diffusion models*,† become image synthesis state-of-the-art;
- Produce incredible results‡ :

* ***Deep Unsupervised Learning using Nonequilibrium Thermodynamics***, *J. Sohl-Dickstein, ICML 2015*
† ***Denoising Diffusion Probabilistic Models***, *Ho et al., NIPS 2020*
‡ ***High-Resolution Image Synthesis with Latent Diffusion Models***, *R. Rombach et al, CVPR 2022*

Diffusion Models

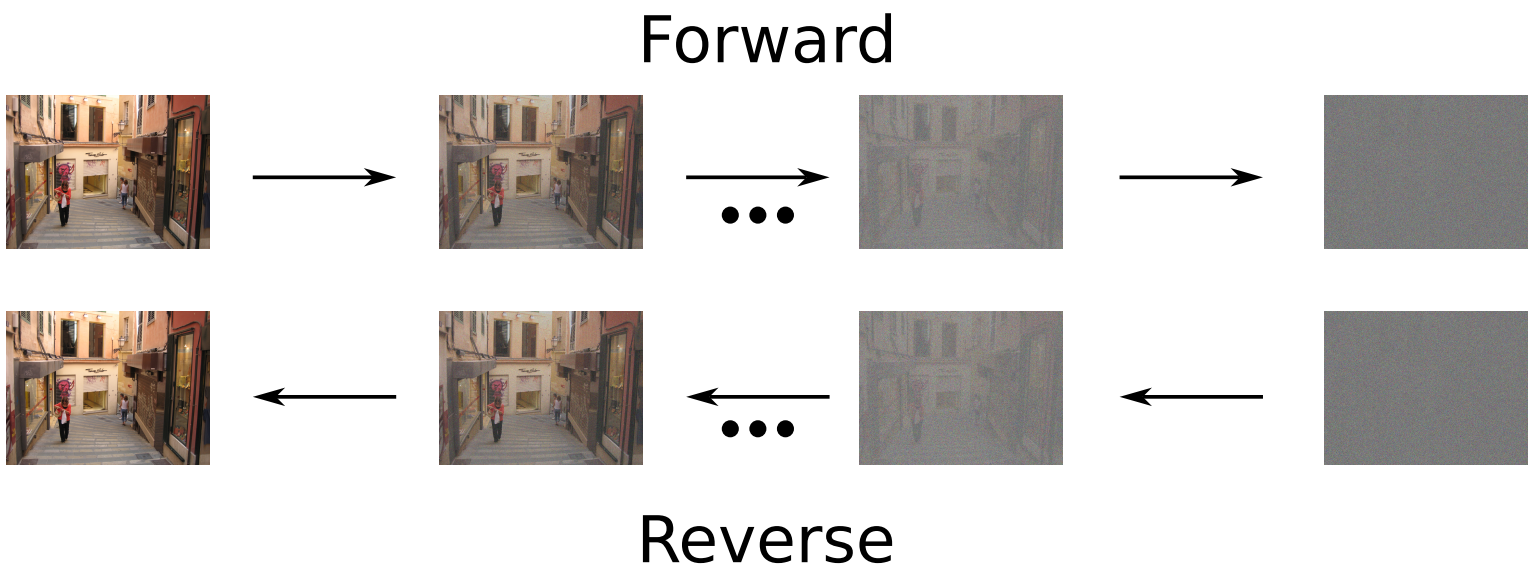
- There are several formulations of diffusion models, with different technical tools;
  - All lead to similar algorithms;
- In this lesson, we present the **Denoising Diffusion Probabilistic Models (Ho et al) version**;
  - Less technical tools, but slightly more complicated;
  - As far as possible, we maintain same notation;
- Other versions : Score-based, Denoising Diffusion Implicit Models;

Diffusion Models

- Main difference between diffusion models and other generative models:
  - Neural network **applied iteratively**;

Diffusion Models

- Main difference between diffusion models and other generative models:
  - Neural network **applied iteratively**;
- Core idea: we know how to add noise to an image; if we know how to remove it, we can **synthesise images from noise**
  - "Forward/reverse*" random processes;



* *We use "reverse" to not confuse with "backward" in backpropagation*

Diffusion Models

- Diffusion model algorithm :
  1. Train a neural network to denoise images;
  2. Sample a random Gaussian noise;
  3. Iteratively denoise to produce random synthesised image;
- Sounds simple !
- Well, let us look at the mathematical formulation;

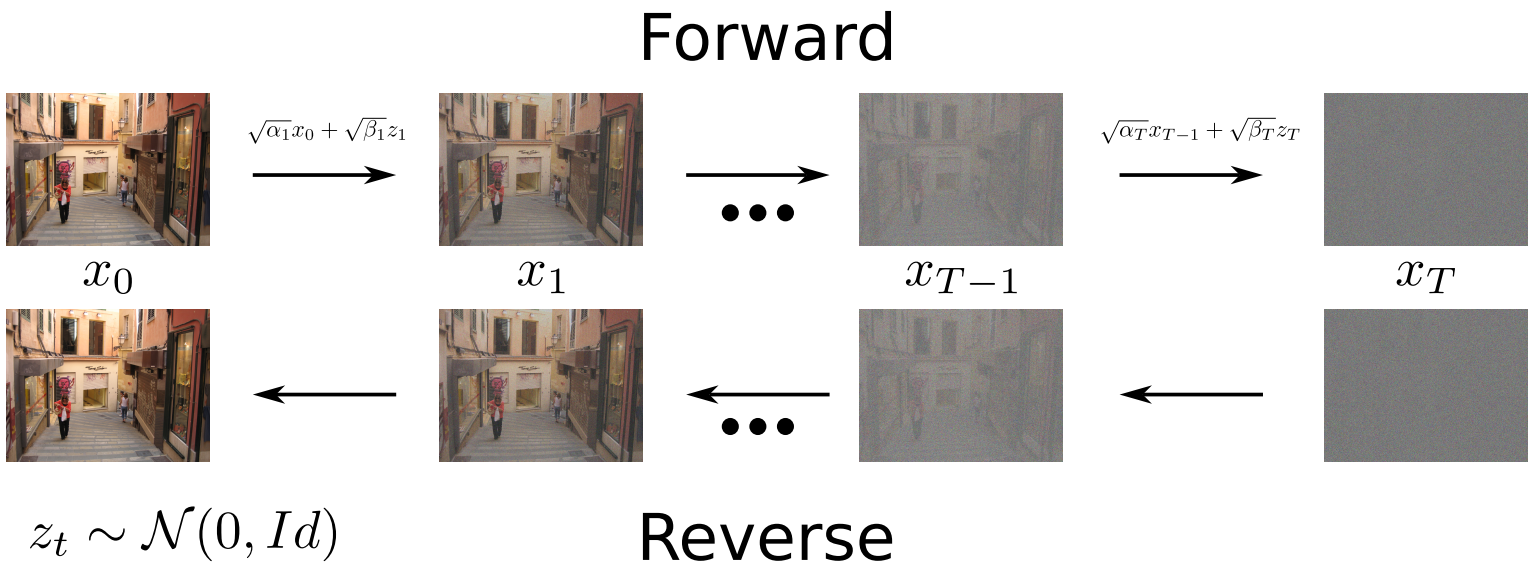
Thanks to Arthur Leclaire and Bruno Galerne for their exceedingly enlightening explanations on this subject !

Forward process

- Diffusion models first set up a **forward** process: $(x_0, x_1, x_2, \ldots, x_T)$
  - x_t's are images with an increasing amount of noise;

Forward process

- Diffusion models first set up a **forward** process: $(x_0, x_1, x_2, \ldots, x_T)$
  - x_t's are images with an increasing amount of noise;
- We define $x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{\beta_t} z_t$
  - z_t's are independent Gaussian noises: $z_t \sim \mathcal{N}(0, \beta_t)$;
  - (α_t, β_t) are scalars;



Diffusion Models

- Let q_0 be the probability distribution of x_0 (noiseless image);
- Formally, we wish to draw a sample from q_0 (same goal as any generative model);

Diffusion Models

- Let q_0 be the probability distribution of x_0 (noiseless image);
- Formally, we wish to draw a sample from q_0 (same goal as any generative model);
- First, note that the forward process $(x_0, x_1, x_2, \ldots, x_T)$ forms a **Markov Chain** (x_t only depends on x_{t-1}):

$$q(x_1, \ldots, x_T|x_0) = q(x_T|x_{T-1})\ q(x_{T-1}|x_{T-2})\ \ldots\ q(x_1|x_0) \tag{6}$$

Diffusion Models

- Let q_0 be the probability distribution of x_0 (noiseless image);
- Formally, we wish to draw a sample from q_0 (same goal as any generative model);
- First, note that the forward process $(x_0, x_1, x_2, \ldots, x_T)$ forms a **Markov Chain** (x_t only depends on x_{t-1}):

$$q(x_1, \ldots, x_T|x_0) = q(x_T|x_{T-1})\, q(x_{T-1}|x_{T-2})\, \ldots\, q(x_1|x_0) \tag{6}$$

- Also, by the definition of $q(x_t|x_{t-1})$:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha}x_{t-1}, \sqrt{\beta_t}Id) \tag{7}$$

- Finally, Ho et al set $\alpha_t = \sqrt{1-\beta_t}$;

Diffusion Models

- We have, recursively:

$$
\begin{aligned}
x_t &= \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}z_t \\
&= \sqrt{1-\beta_t}\left(\underbrace{\sqrt{1-\beta_{t-1}}x_{t-2} + \sqrt{\beta_{t-1}}z_{t-1}}_{x_{t-1}}\right) + \sqrt{\beta_t}z_t \\
&= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{(1-\beta_t)\beta_{t-1}}z_{t-1} + \sqrt{\beta_t}z_t \\
&= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{(\alpha_t)(1-\alpha_{t-1})}z_{t-1} + \sqrt{1-\alpha_t}z_t
\end{aligned}
\tag{8}
$$

- Reminder: z_{t-1} and z_t are i.i.d normal variables;
- NB: we have converted β_t to α_t for conveniance of formulae;

Diffusion Models

- Recall: sum of two independent Gaussian r.v. is also a Gaussian r.v.
  - Mean and **variance** are both summed;
- Thus, we have $\sqrt{(\alpha_t)(1-\alpha_{t-1})}z_{t-1} + \sqrt{1-\alpha_t}z_t \sim \mathcal{N}(0, 1-\alpha_t\alpha_{t-1})$, and we can write:

$$x_t = \sqrt{\alpha_t\alpha_{t-1}}x_{t-1} + \sqrt{1-\alpha_t\alpha_{t-1}}\epsilon_t, \tag{9}$$

- where $\epsilon_t \sim \mathcal{N}(0, Id)$

Diffusion Models

- Recall: sum of two independent Gaussian r.v. is also a Gaussian r.v.
  - Mean and **variance** are both summed;
- Thus, we have $\sqrt{(\alpha_t)(1-\alpha_{t-1})}z_{t-1} + \sqrt{1-\alpha_t}z_t \sim \mathcal{N}(0, 1-\alpha_t\alpha_{t-1})$, and we can write:

$$x_t = \sqrt{\alpha_t\alpha_{t-1}}x_{t-1} + \sqrt{1-\alpha_t\alpha_{t-1}}\epsilon_t, \tag{9}$$

- where $\epsilon_t \sim \mathcal{N}(0, Id)$
- More generally, wrt x_0, we can write:

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_t, \tag{10}$$

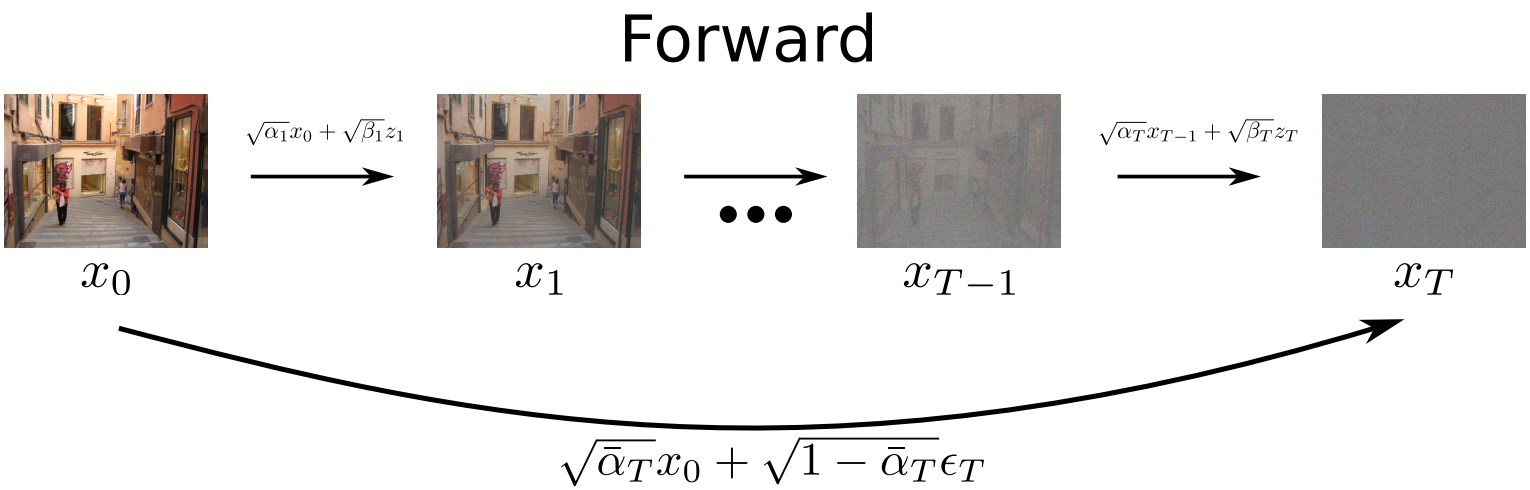
- where $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$

Diffusion Models

- Intuitively, ϵ_t is the noise which produces x_t from the initial image x_0

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon_t, \tag{11}$$

- Thus, we can skip to the end of the Markov chain !



Reverse process

- Now, **how do we go in the reverse direction ?**
  - Recall: we want to do this to randomly synthesise images;
- This is in fact not easy: we only want to remove noise from x_t to x_{t-1}
  - Most denoisers estimate x_0 directly;
  - Furthermore, they are not trained on very high noise levels;
- To solve this, let us look at the mathematical formulation more carefully;

Diffusion Models

- Formally, reverse process is again a Markov chain $(x_T, x_{T-1}, \ldots, x_0)$;
  - This is also chosen to be a series of Gaussian r.v.'s*;
  - We note p_θ the prob. distribution of the backward process;

* *DDPM authors state that this is accurate for small variances β_t, but this is quite a vague point in the literature*

Diffusion Models

- Formally, reverse process is again a Markov chain $(x_T, x_{T-1}, \ldots, x_0)$;
  - This is also chosen to be a series of Gaussian r.v.'s*;
  - We note p_θ the prob. distribution of the backward process;
- We have $p_\theta(x_T, \ldots, x_0) = p(x_T) \prod_{t=T}^{1} p_\theta(x_{t-1}|x_t)$;

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{12}$$

- **How to calculate the mean μ_θ and covariance Σ_θ of this reverse process, for each t ?**

* *DDPM authors state that this is accurate for small variances β_t, but this is quite a vague point in the literature*

Diffusion Models

- We first note a special property of the Markov chain when the x_t's are Gaussian r.v.'s:
  - If (x_0, x_t) are known, x_{t-1} is also Gaussian !

Diffusion Models

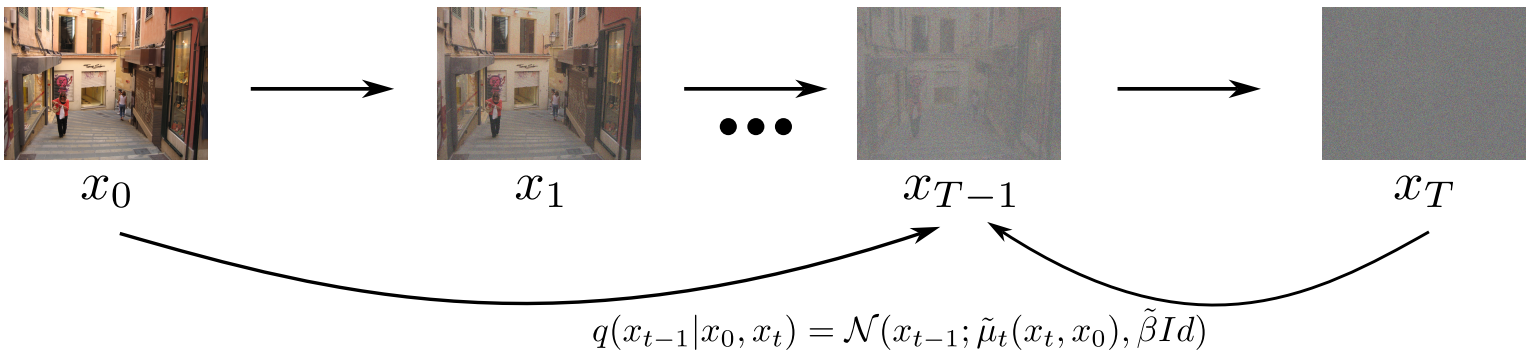
- We first note a special property of the Markov chain when the x_t's are Gaussian r.v.'s:
  - If (x_0, x_t) are known, x_{t-1} is also Gaussian !

- $q(x_{t-1}|x_0, x_t) = \mathcal{N}(x_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t Id)$, with

$$\tilde{\mu}(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} x_t \tag{13}$$

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \tag{14}$$

Reverse (partial)



Diffusion Models

- Why is this useful ? We don't know x_0 in practice: indeed, x_0 is what we are trying to produce!;

Diffusion Models

- Why is this useful ? We don't know x_0 in practice: indeed, x_0 is what we are trying to produce!;
- In fact, we can **train a NN to estimate** x_0, and then use this to sample $q(x_{t-1}|x_0, x_t)$;
  - Indeed, estimating x_0 is the common goal of denoisers;
  - Easier than estimating x_{t-1} directly from x_t;
- Thus, knowing x_t and an estimation of x_0, we can sample x_{t-1};

Diffusion Models

- Why is this useful ? We don't know x_0 in practice: indeed, x_0 is what we are trying to produce!;
- In fact, we can **train a NN to estimate x_0**, and then use this to sample $q(x_{t-1}|x_0, x_t)$;
  - Indeed, estimating x_0 is the common goal of denoisers;
  - Easier than estimating x_{t-1} directly from x_t;
- Thus, knowing x_t and an estimation of x_0, we can sample x_{t-1};
- So, our algorithm is now :
  1. Train denoiser;
  2. Sample Gaussian noise;
  3. Iterate:
     - Estimate x_0 using denoiser(x_t);
     - Sample x_{t-1}, using x_T and estimate of x_0
- So, how is the denoiser trained ?

Diffusion Models

- We train a network to maximise $p_\theta(x_0)$, using the ELBO, as in the VAE;
- Slightly different form due to the Markov chain setting:

$$\log(p_\theta(x_0)) \geq \mathbb{E}\left[KL\left(q(x_T|x_0)||p_\theta(x_T)\right)\right] + \mathbb{E}\left[\log p_\theta(x_0|x_1)\right] \tag{15}$$

$$+ \sum_{t>1} KL\left(q(x_{t-1}|x_0, x_t)||p_\theta(x_{t-1}|x_t)\right) \tag{16}$$

- The terms in blue are known, and do not intervene in the optimisation;
- So, what is $KL\left(q(x_{t-1}|x_0, x_t)||p_\theta(x_{t-1}|x_t)\right)$?

Diffusion Models

- Recall that $p_\theta(x_{t-1}|x_t)$ was chosen to be Gaussian
  - We just don't know the mean and variance yet;

- Thus, since $p_\theta(x_{t-1}|x_t)$ and $q(x_{t-1}|x_0, x_t)$ are both Gaussian, we have a closed form solution for $KL\left(q(x_{t-1}|x_0, x_t)||p_\theta(x_{t-1}|x_t)\right)$

$$KL\left(q(x_{t-1}|x_0, x_t)||p_\theta(x_{t-1}|x_t)\right) = \mathbb{E}_q\left[\frac{1}{\beta_t}\|\tilde{\mu}(x_t, x_0) - \mu_\theta(x_t, t)\|_2^2\right] \tag{17}$$

Diffusion Models

- Recall that $p_\theta(x_{t-1}|x_t)$ was chosen to be Gaussian
  - We just don't know the mean and variance yet;
- Thus, since $p_\theta(x_{t-1}|x_t)$ and $q(x_{t-1}|x_0, x_t)$ are both Gaussian, we have a closed form solution for $KL\left(q(x_{t-1}|x_0, x_t)||p_\theta(x_{t-1}|x_t)\right)$

$$KL\left(q(x_{t-1}|x_0, x_t)||p_\theta(x_{t-1}|x_t)\right) = \mathbb{E}_q\left[\frac{1}{\beta_t}\|\tilde{\mu}(x_t, x_0) - \mu_\theta(x_t, t)\|_2^2\right] \tag{17}$$

- Basically: the training loss makes the **mean of $p_\theta(x_{t-1}|x_t)$ as close as possible to $q(x_{t-1}|x_0, x_t)$**;
- Role of the network: estimate x_0 from x_t;

Diffusion Models

- Final note: the authors of DDPM reformulate the loss such that the **network estimate the noise** ϵ_t, rather than x_0
  - Why do they do this ? Because they report that this gives better results (see Section 3.2) ...
- Since $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon_t$ (Equation (11)), it is equivalent to estimate x_0 or ϵ_t from x_t;
  - Knowing x_0 gives us ϵ_t directly, and vice cersa

Diffusion Models

- Final note: the authors of DDPM reformulate the loss such that the **network estimate the noise** ϵ_t, rather than x_0
  - Why do they do this ? Because they report that this gives better results (see Section 3.2) ...
- Since $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon_t$ (Equation (11)), it is equivalent to estimate x_0 or ϵ_t from x_t;
  - Knowing x_0 gives us ϵ_t directly, and vice cersa
- Final training loss (after much simplification):

$$\mathcal{L} = \mathbb{E}_{t, x_0, \epsilon} \left[\left\| \epsilon - f_\theta \left(\underbrace{\sqrt{\bar{\alpha}} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon}_{x_t} \right) \right\|_2^2 \right] \tag{18}$$

- ϵ : noise to estimate with NN denoiser f_θ;

Diffusion Models

- Once the training is carried out, we can sample $p_\theta(x_{t-1}|x_t)$ using the following formula:

$$x_{t-1} = \frac{1}{\sqrt{\alpha}_t}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} f_\theta(x_t, t)\right) + \tilde{\beta} z_t, \tag{19}$$

- where $z_t \sim \mathcal{N}(0, Id)$ is a Gaussian noise;

Diffusion model summary

Algorithm 1 Diffusion model training

- Repeat following until converged:
  1. $x_0 \sim q(x_0)$ (take example x_0 from database);
  2. $t \sim \text{Uniform}(1, \ldots, T)$;
  3. $\epsilon \sim \mathcal{N}(0, Id)$;
  4. One optimiser step on $\nabla_\theta \left\| \epsilon - f_\theta \left(\sqrt{\bar{\alpha}} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon \right) \right\|_2^2$

Algorithm 2 Diffusion model testing (synthesis)

- $x_T \sim \mathcal{N}(0, Id)$;
- For $t = T, \ldots, 1$, do
  1. $z \sim \mathcal{N}(0, Id)$, if $t > 1$, else $z = 0$;
  2. $x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} f_\theta(x_t, t) \right) + \tilde{\beta} z_t$

Diffusion models summary

- Idea: if we can reverse a noise process, then we can synthesise random images;
- Forward process $x_{t-1} \rightarrow x_t$ is easy to sample: we just add Gaussian noise a certain number of timesteps;
- Reverse process $x_t \rightarrow x_{t-1}$ is more difficult to sample;

Diffusion models summary

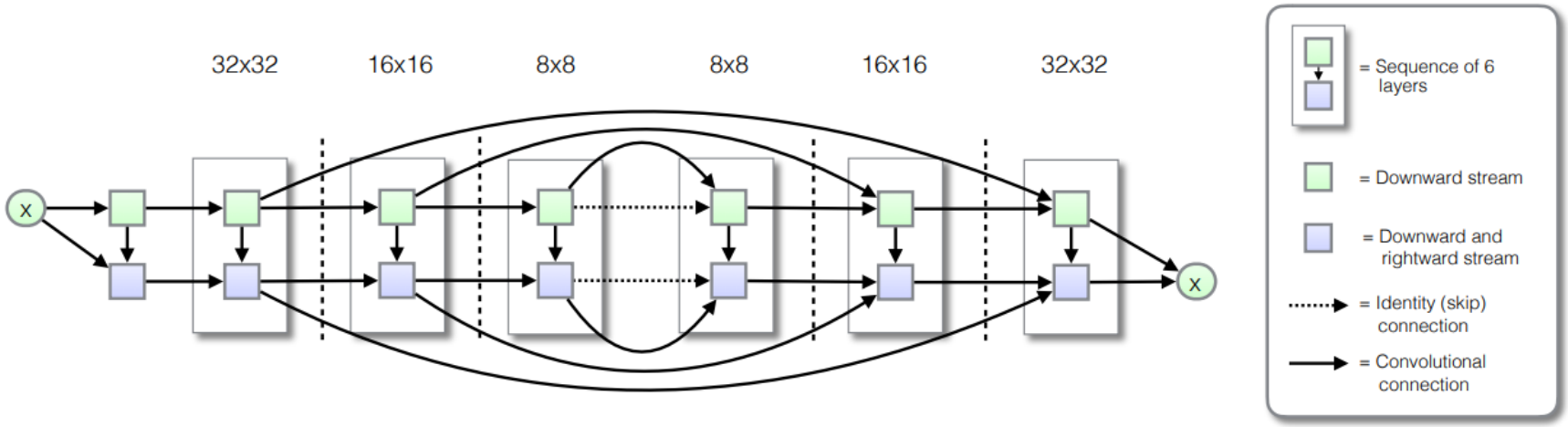
- Idea: if we can reverse a noise process, then we can synthesise random images;
- Forward process $x_{t-1} \to x_t$ is easy to sample: we just add Gaussian noise a certain number of timesteps;
- Reverse process $x_t \to x_{t-1}$ is more difficult to sample;
- We first note that, if we know (x_0, x_t), then $p_\theta(x_{t-1}|x_0, x_t)$ is Gaussian;
  - Thus, we need to estimate x_0 first: job of a denoiser;
- We formulate the ELBO of $\log p_\theta(x_0)$ such that $KL\left(q(x_{t-1}|x_0, x_t)||p_\theta(x_{t-1}|x_t)\right)$ appears.
  - Meaning : the Gaussian reverse process should be as close as possible to $q(x_{t-1}|x_0, x_t)$;

Diffusion Models

- A network f_θ is trained to predict x_0 (or, equivalently, ϵ_t) from any x_t;
- The synthesis starts with a noise image, and these two steps are iterated:
  1. Use f_θ to estimate x_0 (or ϵ_t);
  2. Use x_t and the estimation of x_0 to sample x_{t-1}

Diffusion Models

- Architecture of f_θ (in DDPM) is a U-Net, more precisely "PixelCNN++"*
  - This is a common type of architecture for denoising;



* ***Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications***, *T. Salimans et al, arXiv:1701.05517, 2017*

Some legitimate questions !

- Why can't we just train a network to predict/sample x_{t-1} directly from x_t ?
  - Why do we have to sample it *indirectly* via x_0 ?

Some legitimate questions !

- Why can't we just train a network to predict/sample x_{t-1} directly from x_t ?
  - Why do we have to sample it *indirectly* via x_0 ?
  - Answer: it is **difficult for the network to predict the same image with slightly less noise** ($x_t \rightarrow x_{t-1}$);
  - If you could do this, diffusion models would be much simpler (less maths);

Some legitimate questions !

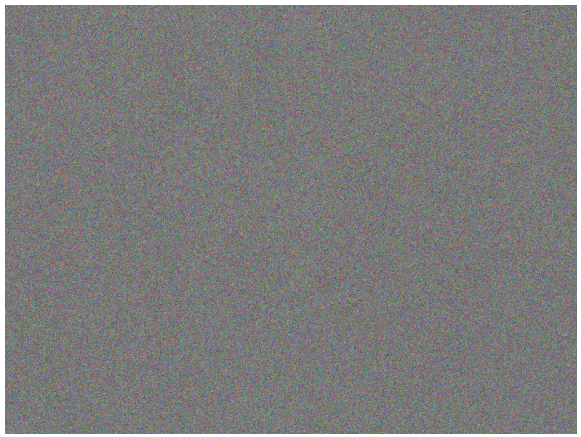
- Why can't we just train a network to predict/sample x_{t-1} directly from x_t ?
  - Why do we have to sample it *indirectly* via x_0 ?
  - Answer: it is **difficult for the network to predict the same image with slightly less noise** ($x_t \rightarrow x_{t-1}$);
  - If you could do this, diffusion models would be much simpler (less maths);
- **Why is it better to carry out an iterative diffusion process**, rather than just one step (as in VAEs/GANs) ?

Some legitimate questions !

- Why can't we just train a network to predict/sample x_{t-1} directly from x_t ?
  - Why do we have to sample it *indirectly* via x_0 ?
  - Answer: it is **difficult for the network to predict the same image with slightly less noise** ($x_t \rightarrow x_{t-1}$);
  - If you could do this, diffusion models would be much simpler (less maths);
- **Why is it better to carry out an iterative diffusion process**, rather than just one step (as in VAEs/GANs) ?
  - This is currently a subject of research;

Some legitimate questions !

- At high noise levels (large T), f_θ is not really denoising (too much noise, can't see x_0);



How to denoise this image ?

- If not denoising, what is f_θ doing?

Original DDPM results

DDPM results

Diffusion Models

Stable diffusion

- Diffusion models have many different variants;
- One of the most famous is "Stable Diffusion"$*$
- Carries out diffusion in a pretrained latent space;
- Conditional on a textual input (we do not explain this here)

* ***High-Resolution Image Synthesis with Latent Diffusion Models***, *R. Rombach et al, CVPR 2022*

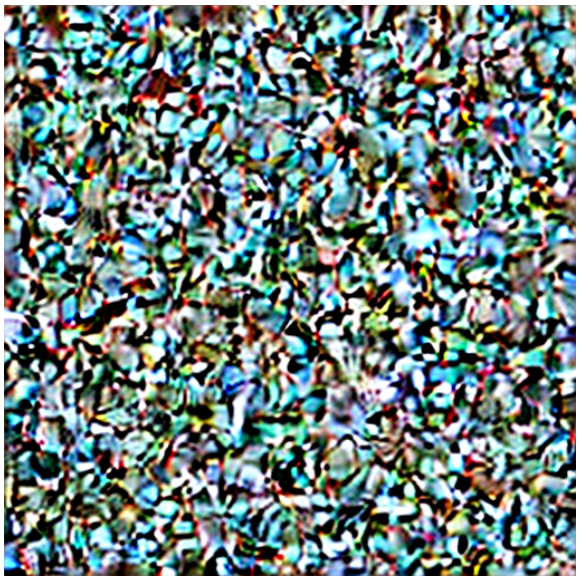
Stable diffusion

- Produces incredible results:

Diffusion Models

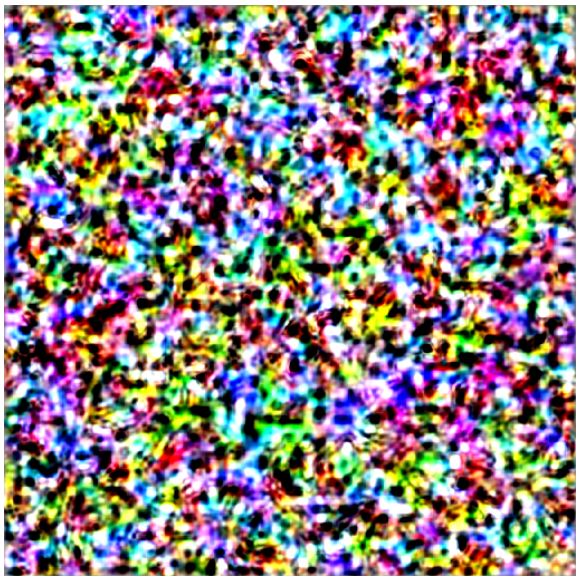
- Example of iterations of stable diffusion;
- Text input: "Link fighting with Ganon"

Noise



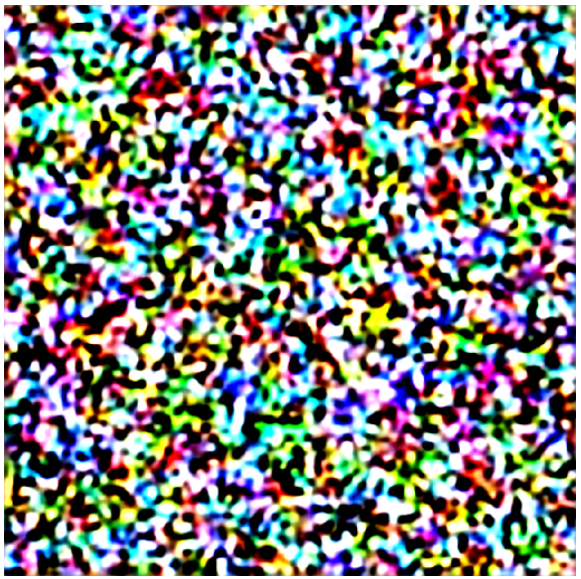
ϵ_{50}

Estimation* of x_0



$x_{50} - \varepsilon_{50}$

x_0



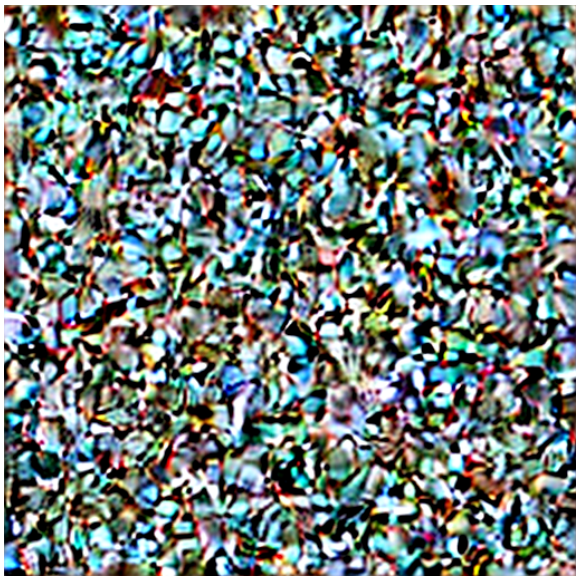
x_{50}

Stable Diffusion is in a latent space, so this is not exactly correct here

Diffusion Models

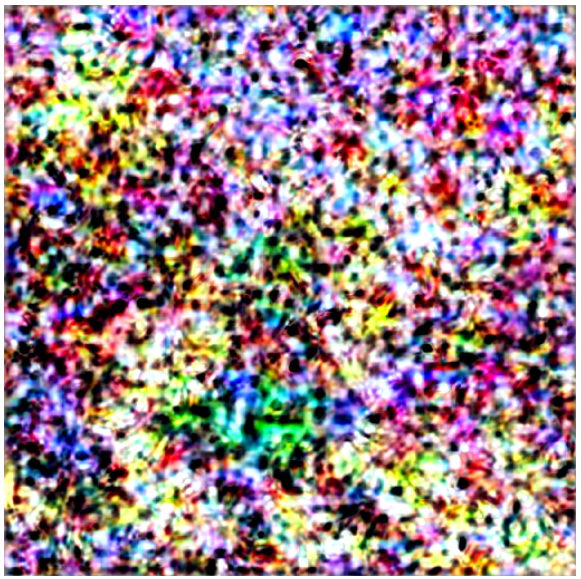
- Example of iterations of stable diffusion;
- Text input: "Link fighting with Ganon"

Noise



ϵ_{40}

Estimation* of x_0



$x_{40} - \varepsilon_{40}$

x_0

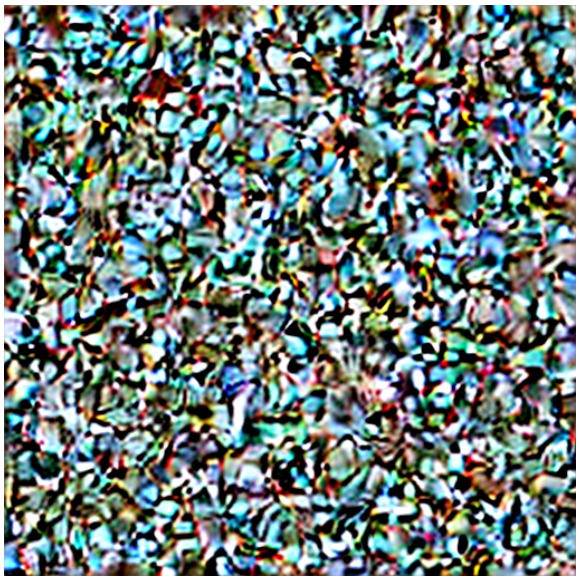
x_{40}

Stable Diffusion is in a latent space, so this is not exactly correct here

Diffusion Models

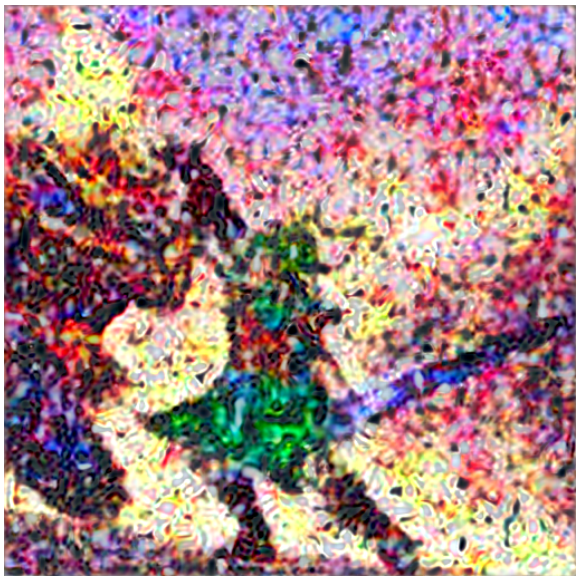
- Example of iterations of stable diffusion;
- Text input: "Link fighting with Ganon"

Noise



ϵ_{30}

Estimation* of x_0



$x_{30} - \varepsilon_{30}$

x_0

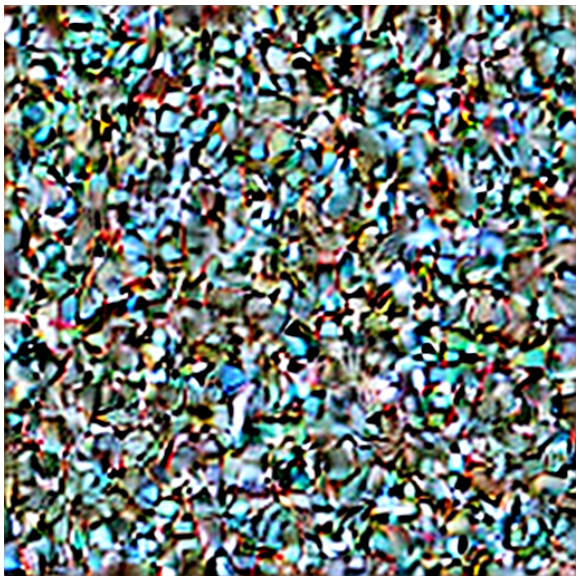
x_{30}

Stable Diffusion is in a latent space, so this is not exactly correct here

Diffusion Models

- Example of iterations of stable diffusion;
- Text input: "Link fighting with Ganon"

Noise

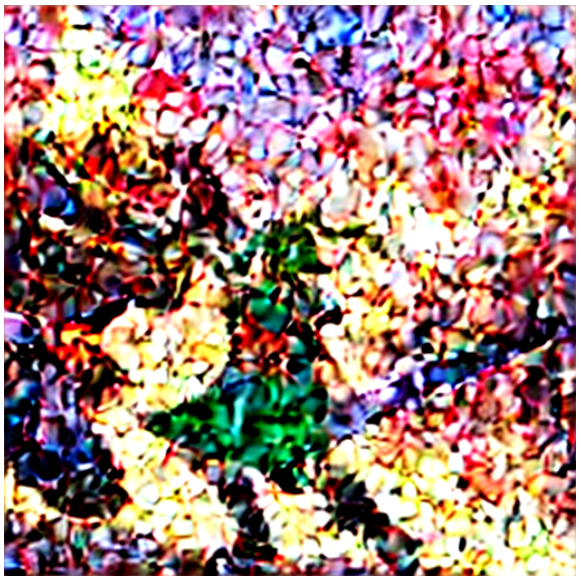


ϵ_{20}

Estimation* of x_0

$x_{20} - \varepsilon_{20}$

x_0



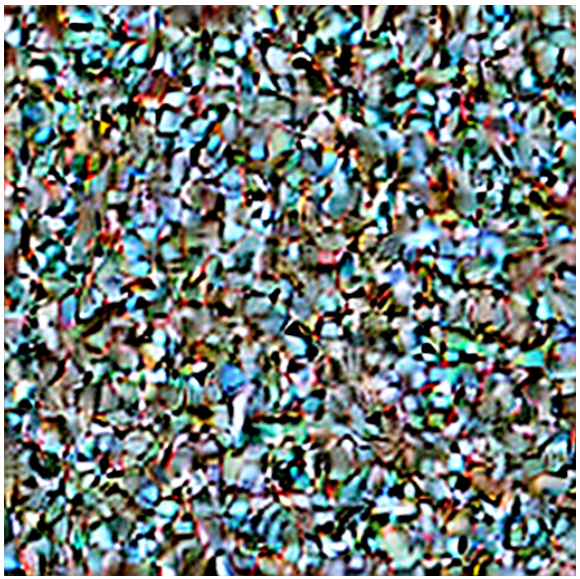
x_{20}

Stable Diffusion is in a latent space, so this is not exactly correct here

Diffusion Models

- Example of iterations of stable diffusion;
- Text input: "Link fighting with Ganon"

Noise



ϵ_{10}

Estimation* of x_0

$x_{10} - \varepsilon_{10}$

x_0

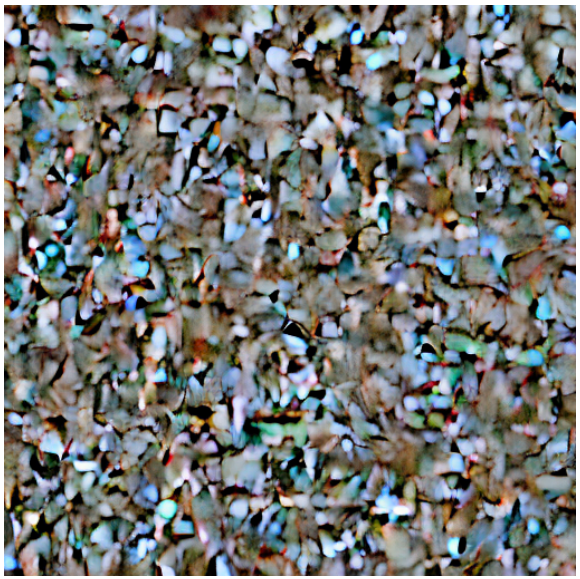
x_{10}

Stable Diffusion is in a latent space, so this is not exactly correct here

Diffusion Models

- Example of iterations of stable diffusion;
- Text input: "Link fighting with Ganon"

Noise



ϵ_1

Estimation* of x_0

$x_1 - \varepsilon_1$

x_0

x_1

Stable Diffusion is in a latent space, so this is not exactly correct here

Advantages of diffusion models

- More stable training wrt to GANs, which require a discriminator;
- Due to the sampling at each time step t, one initial noise x_T can produce many different outputs:

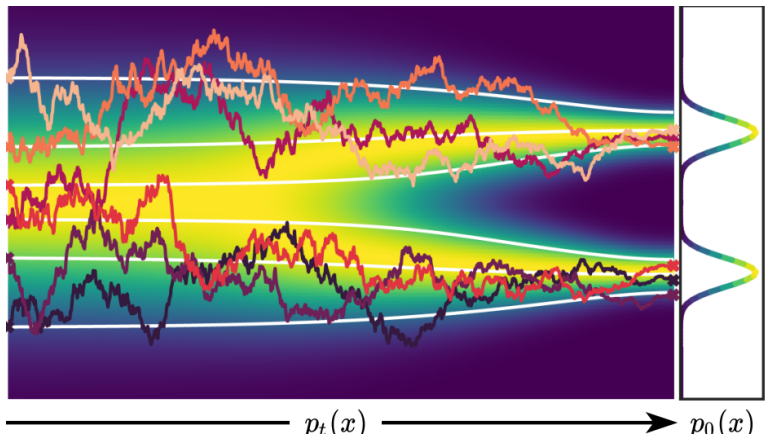


Illustration from Song et al 2021*

- VAEs and GANs produce (mostly) the same image for each initial noise

* *Score-Based Generative Modeling Through Stochastic Differential Equations, Song et al, ICLR 2021*

Disadvantage of diffusion models

- Networks f_θ tend to be **huge** !
  - Why is this ? Because f_θ has to denoise at a very wide range of noise levels (even when there is only noise)
- Theory can be quite complicated;
  - Not always explained or implemented clearly (various practical techniques)
  - Theory and practice are often not aligned;

Summary

Conclusion

We have seen three types of generative models

1. Variational autoencoders
   - An Autoencoder whose latent space is encouraged to follow a certain distribution
   - Variational Bayesian formulation
2. Generative Adversarial Networks
   - A generator trained to create realistic data
   - A discriminator trained to identify true and false data examples
3. Diffusion models
   - Invert iterative denoising process for synthesis;
     - Two Markov chains: forward and reverse;
   - Reverse process difficult to sample: achieve it via estimation of x_0

Summary of advantages and weaknesses of Generative Models

Model/method	Advantages	Disadvantages
VAE	• Rigourous formulation • Encoder and decoder trained	• Loss must be rewritten for different probability distributions (not easy) • Less impresive performance than GANs/diffusion models
GAN	• Highly flexible • Applied to complex data, impressive results	• Difficult to train • No reverse transformation (encoder)
Diffusion models	• Versatile algorithm • Easier to train than GANs • Produces best results	• Involved theory • Slow iterative procedure

References

A few references

- Kingma, Diederik, and Welling, **Auto-encoding Variational Bayes**, arXiv:1312.6114, 2013
- Goodfellow et al, **Generative Adversarial Nets**, NIPS 2014
- Gatys, L. A., Ecker, A. S, and Bethge, M., **Texture synthesis using convolutional neural networks**, NIPS, 2015
- Gatys, L. A., Ecker, A. S, and Bethge, M., **A Neural Algorithm of Artistic Style**, arXiv:1508.06576, 2015
- Radford, A. and Chintala, S., **Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks**, arXiv:1511.06434, 2015
- Zhu, Krähenbühl, Schechtman, Efros, **Generative Visual Manipulation on the Natural Image Manifold**, ECCV,2016
- Karras, T., Aila, T., Laine, S., and Lehtine, J.,**Progressive growing of gans for improved quality, stability, and variation**, arXiv preprint arXiv:1710.10196, 2017

Variational autoencoders - Variational Autoencoder loss function

- We show that the ELBO can be rewritten in the following manner:

$$\text{ELBO}(q_\phi) = \mathbb{E}_{q_\phi}\left[\log(p_\theta(x|z))\right] - KL(q_\phi(z|x) \,||\, p_\theta(z))$$

- Start out with the initial formulation:

$$\begin{aligned}
\text{ELBO}(q_\phi) &= \mathbb{E}_{q_\phi}\left[\log(p_\theta(x,z))\right] - \mathbb{E}_{q_\phi}\left[\log q_\phi(z|x)\right] \\
&= \int_{q_\phi} \left[\log\left(p_\theta(x|z).p_\theta(z)\right) - \log q_\phi(z|x)\right]\ q_\phi(z|x)dz \qquad \text{Conditional prob.} \\
&= \int_{q_\phi} \log(p_\theta(x|z)).q_\phi(z|x)dz - \int_{q_\phi} \left(\log q_\phi(z|x) - \log p_\theta(z)\right) q_\phi(z|x)dz \\
&= \mathbb{E}_{q_\phi}\left[\log p_\theta(x|z)\right] - KL\left(q_\phi(z|x)||p_\theta(z)\right)
\end{aligned}$$