Generative Models

When we are using *discriminative models* we're interested in doing a *classification*, so we are just interested in discriminating objects belonging to different classes.

Generative models, on the other hand, are models that try to learn the actual *distribution* p_{data} of real data from *available samples* (training set).

Goal: ==build probability distribution p_{model} close to p_{data}.== We can either try to - explicitly estimate the distribution (i.e. with a math formula) - build a *generator* able to sample according to p_{model}, possibly providing estimations of the likelihood - i.e., we have a dataset of faces, and we try to generate faces which are similar to the dataset of faces.

Obviously, we can also compute a likelihood and other quality indicators that would help us try to measure the quality of our generated samples.

Why studying Generative Models?

- Improve our knowledge on data and their distribution in the visible feature space
- Improve our knowledge on the latent representation of data and the encoding of complex high-dimensional distributions
- Typical approach in many problems involving multi-modal outputs
- Find a way to produce realistic samples from a given probability distribution
- Generative models can be incorporated into reinforcement learning, e.g. to predict possible futures

Multi-modal output In many interesting cases there is *no unique intended solution* to a given problem: - add colors to a gray-scale image - guess the next word in a sentence - fill a missing information - predict the next state/position in a game - . . . When the output is intrinsically multi-modal (and we do not want to give up to the possibility to produce multiple outputs) we need to rely on *generative modeling*.

Let's take for example the idea of coloring an image. In this case, there are many different solutions to this problem, and interpolating or mixing all the possible solution would not give us an especially good result. So, we want to generate a possible solution in some conditioned and stochastic way.

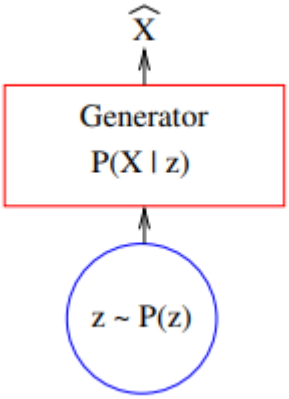
Latent variables models

In **latent variable models** we express the probability of a *data point* X through

$$P(X) = \int \underbrace{P(X|z)}_{\text{Generator}} P(z)dz \approx \mathop{\mathbb{E}}_{z \sim P(z)} P$$

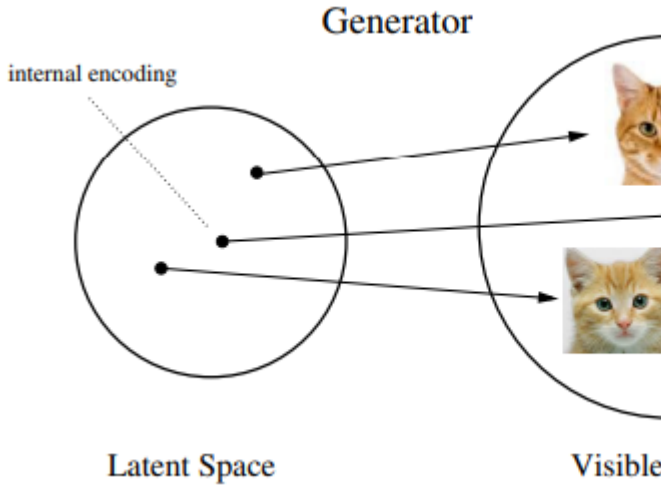
marginalization over a vector of *latent variables*:

The generator computes the probability conditioned on a *vector of latent variables* z (which could be, for example, the internal representation of a face). These latent variables capture important but *hidden information* or patterns that are not explicitly measured or recorded. Essentially, z could be seen as an internal representation of X. z is *distributed with a known prior distribution* $P(z)$.



As we can see, $P(z)$ is passed to the generator, which in turn generates a possible sample $\hat{X}$ that belongs in the distribution. z is also called latent representation (or latent encoding), while $\hat{X}$ is called also visible representation. Typically, the generator is a *deterministic component*.

> [!As said in the slides] This simply means that we try to learn a way to sample X starting from a vector of values z (this happens in the generator, which is $P(X|z)$), where z is *distributed with a known prior distribution* $P(z)$. z is the latent encoding of X.



Dimensions of latent space and visible space

The Visible space is a huge space (i.e. for an image of dimensions 64x64, the dimension of the space 64x64x3 (since we have to consider the colors also)). The dimension of the *latent space*, instead, could be *much smaller*, since the [[2023-04-04 - The data manifold & autoencoders#Data Manifold|manifold) that we want to reach is very small compared to the whole visible space dimension.

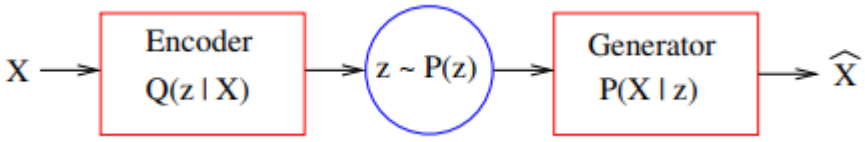
Classes of generative models

There are four main classes of generative models: - **compressive models** - they compress the latent space according to what we said earlier.
- Variational Autoencoders (VAEs) - Generative Adversarial Networks (GANs)
- **dimension preserving models** - the latent space has the same dimension of the visible space. - Normalizing Flows - Denoising Diffusion Models

Each model also differs in the way the *generator* is trained.

VAE - Variational AutoEncoders

To train a generator in a VAE, you couple a generator with an **encoder**, thus producing a *latent encoding* z *given* X. This will be distributed according to an *inference distribution* (a statistical deduction) $Q(z|X)$.



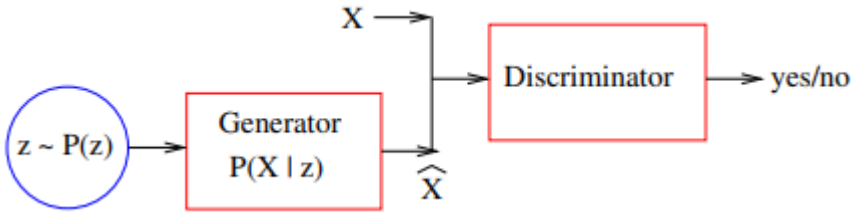
It basically tries to "reconstruct" the input image, so in that way it kinda works like an [[2023-04-04 - The data manifold & autoencoders#What is an autoencoder?|autoencoder). The different wrt to autoencoders is that we cannot

use them for generators, since we do not know *how z is distributed inside the latent space* (using only autoencoders). So, for example we may pick a point of the latent space which is outside the manifold, and thus we may sample a bad image.

Variational Autoencoders force this problem by trying to *enforce the distribution* of *z to be a Normal/Gaussian distribution*, and this can be seen in the loss function specifically, in which we try to force the z to assume a Gaussian distribution. The loss function aims to: - minimize the reconstruction error between X and $\hat{X}$ - bring the ==marginal inference distribution $Q(z)$ close to the prior distribution $P(z)$== (this is usually a Gaussian distribution) [this was the part I was talking about] And is the sum of these 2 components.

GAN - Generative Adversarial Network

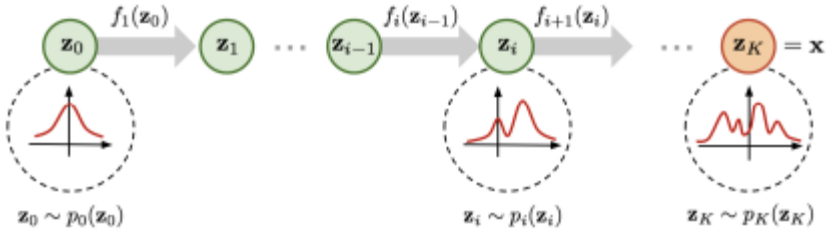
The GAN approach is completely different. In a Generative Adversarial Network, the generator is coupled with a *discriminator*, trying to ==tell apart real data from fake data== produced by the generator. Detector and Generator are trained together *alternatively* (when you train the generator you freeze the discriminator and viceversa).



The loss function aims to: - instruct the *detector* to *spot* the *generator*. - instruct the *generator* to *fool* the *detector*. At this point, one could see that if one is working bad so is the other, thus training is tricky. At the end of this process, the generator is supposed to win of course. Usually, GAN produces better results than VAE.

Normalizing Flows

In Normalizing Flows, the *generator* is *split* into a ==long chain of *invertible* transformations==, so you can essentially go back and invert the transformation.

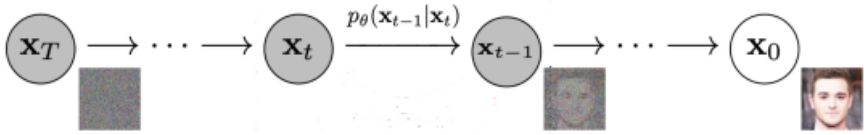


As you may remember, the Normalizing Flow is a *dimension preserving model*, so the latent space is the same dimension as the visible space. What we're doing is trying to apply some transformations to the latent space to obtain the visible space.

The network is trained by *maximizing log-likelihood*. - *Pros*: it allows a precise computation of the resulting log-likelihood. - **Cons**: the fact of restricting to invertible transformation limit the expressiveness of the model.

Diffusion Models

In Diffusion Models, the *latent space* is understood as *a strongly noised version of the image* to be generated. The generator is split into a long chain of denoising steps, where each step t *attempts to remove Gaussian noise* with a given variance

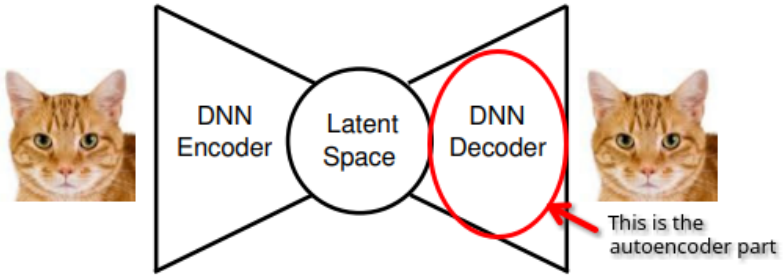


σ_t .
We train a single network implementing the denoising operation, parametric in σ_t.

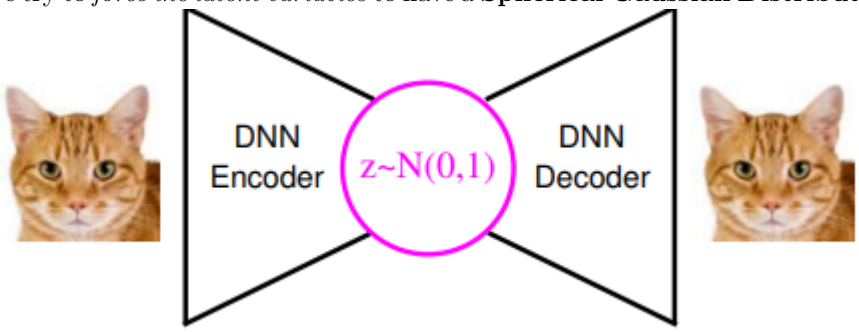
This method uses *Reverse Diffusion*, where Diffusion is a technique which essentially aims to add and distribute noise to an image. One of the famous implementation of this method, Stable Diffusion, uses 1000 steps of denoising. Usually, these models start from a very small image (i.e. 64x64 resolution) and they then add a hyper-resolution step with a totally unrelated network.

VAE in depth

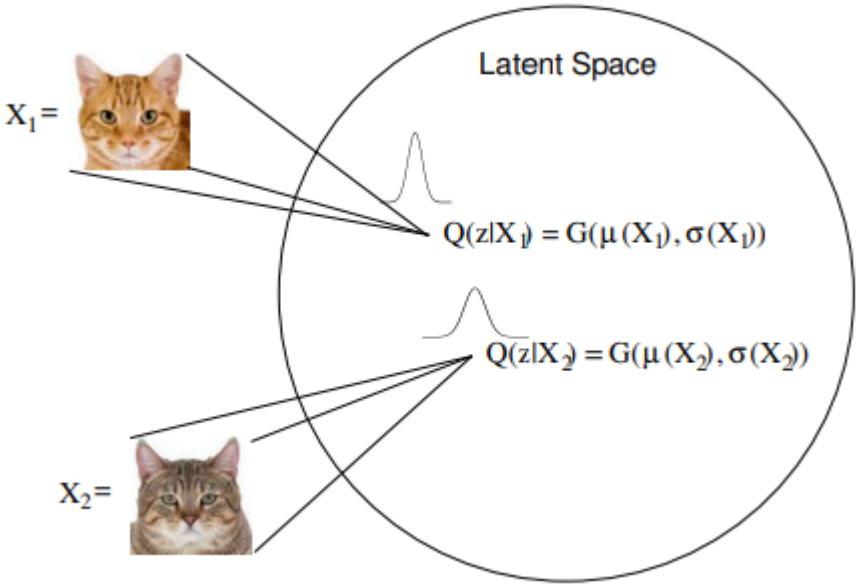
An autoencoder is a net trained to reconstruct input data out of a learned internal representation (e.g. minimizing quadratic distance) The idea is to use the encoder like a traditional generator in a way.



Can we use the decoder (aka the single autoencoder) to generate data by sampling in the latent space? No, since we do not know the distribution of latent variables. So, in a way, we need some control inside of the latent space, and thus we try to *force the latent variables* to have a **Spherical Gaussian Distribution**.



We assume $Q(z|X)$ has a Gaussian distribution $G((X), \sigma(X))$ with different mo-



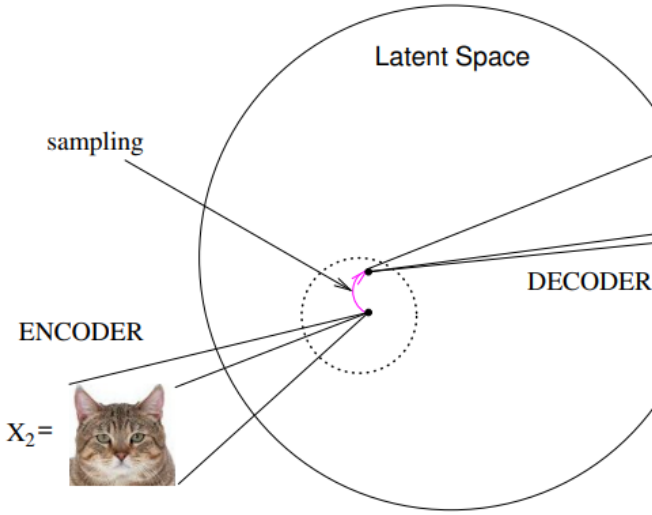
ments for each different input X.

The values $(X), \sigma(X)$ are both *computed by the generator*, that is hence *returning an encoding* $z = (X)$ (the enconding corresponds to the mean) and a *variance* $\sigma(X)$ around it, expressing the portion of the latent space that is essentially encoding ==an information similar to X==.

Sampling in the latent space

==During training==, we sample around (X) with the computed $\sigma(X)$ before passing the value to the decoder. Here essentially what that means: - Suppose

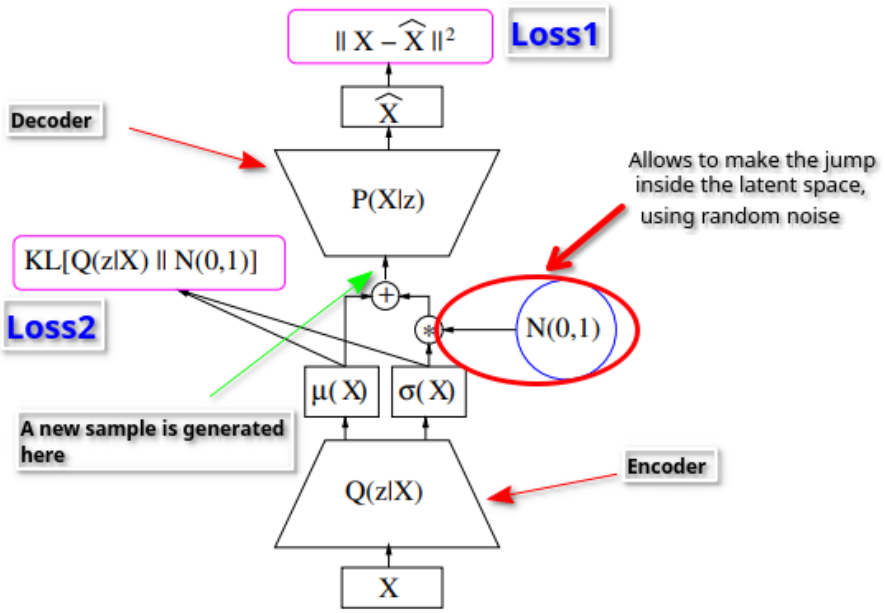
that for the data X_2, we know its mean and its variance σ. Using the variance, *we sample* around it [the dotted circle in the picture] and we find in this neighbourhood an image that is very much similar to X_2.



- This is done before *passing* the value *to the decoder*.
After we pass the value to the decoder, we expect to be able to decode the same image that we received as input.

This process is important for 2 reasons: 1. If we can sample the input image from the whole "variance area", we *cover a larger area of the latent space*. Thus, it increases the probability for the generator of deriving an output that makes sense from the latent space. 2. Among other things, the sampling that we do using the varians *adds noise to the encoding*, and so it improves the robustness of the model.

The full picture of VAE



We also have 2 losses: 1. The **reconstruction error** of the input from the output. 2. The **Kullback-Leiber component**, which is just the distance between the 2 gaussian distribution $N(0,1)$ and $Q(z|X)$ - Allows $Q(z|X)$ to be *as close as possible to a Gaussian* distribution.

The effect of the Kullback-Leiber divergence on latent variables has 2 effects: 1. pushing $_z(X)$ towards 0, so as to *center the latent space around the origin* 2. push $\sigma_z(X)$ towards 1, augmenting the *"coverage" of the latent space*, essential for generative purposes.

VAE problems

- *balancing* log-likelihood and KL regularizer in the loss function
  - Also, if the value of the variance is too small, we may make the areas in the latent space too big, and thus the areas may overlap.
- variable collapse phenomenon
  - i.e., you have 50 variables but only 10 are used.
- marginal inference vs prior distribution mismatch
  - Q(z) vs. Q(z|X), we try to make them as similar as possible but we might fail.
- blurriness (aka variance loss)