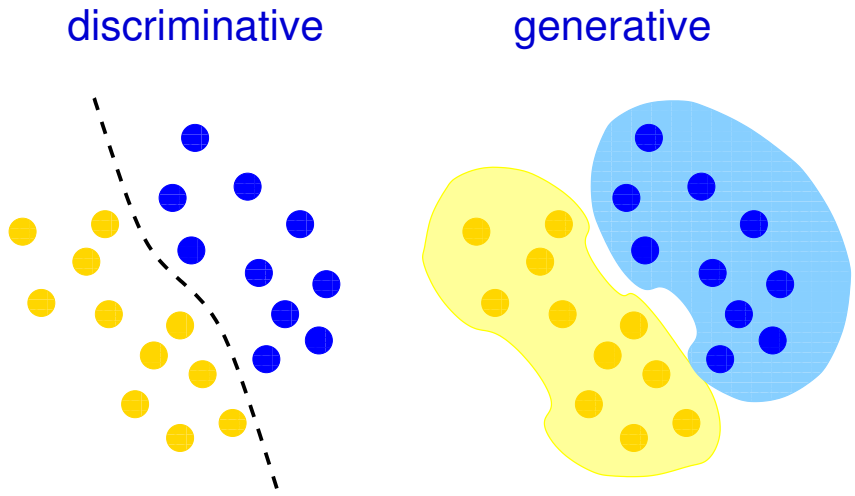
Generative Models



Generative Models

Generative Model: a model that tries 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
- build a generator able to sample according to p_{model}, possibly providing estimations of the likelihood

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
- tipical 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
- ...

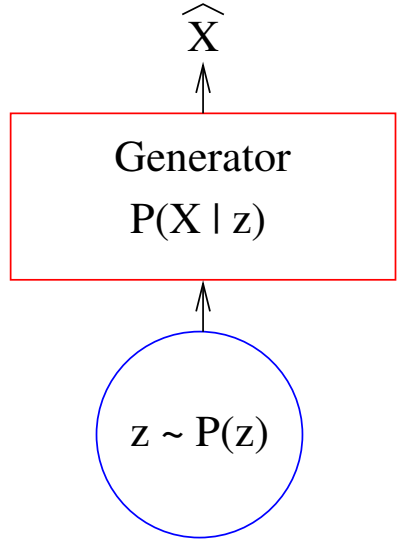
When the output is intrinsically multi-modal (and we do not want to give up to the possibility to produce multiple outputs) we need to rely on generative modeling.

Latent variables models

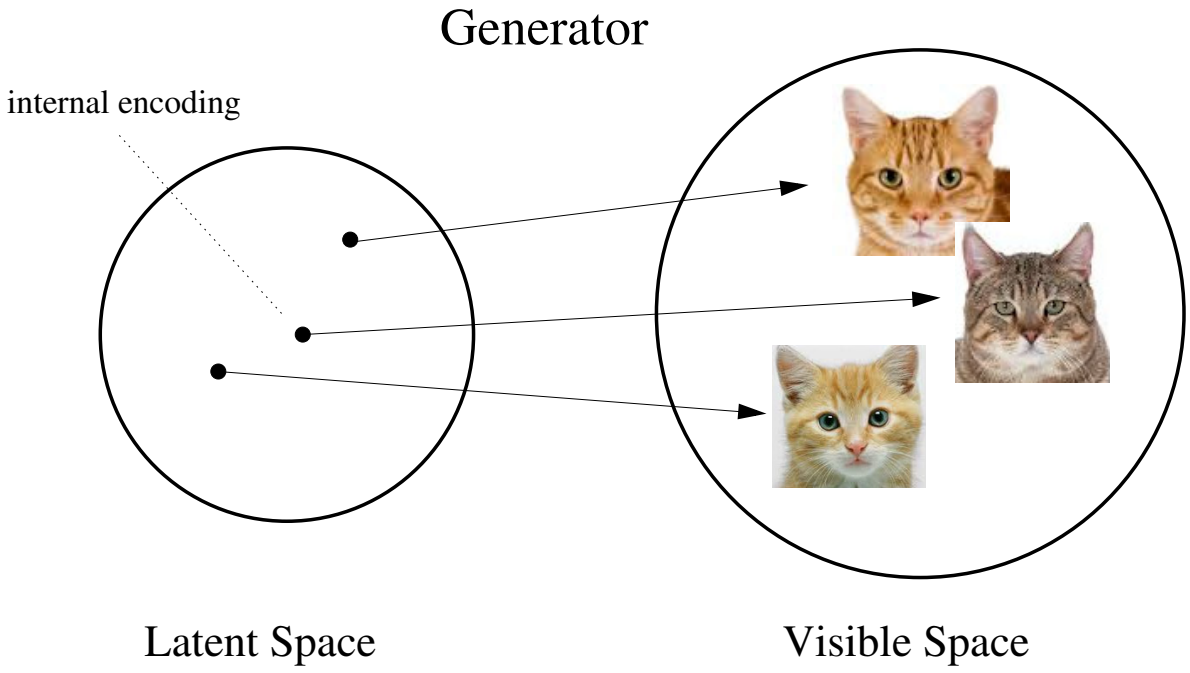
In latent variable models we express the probability of a data point X through **marginalization** over a vector of latent variables:

$$P(X) = \int P(X|z)P(z)dz \approx \mathop{\mathbb{E}}_{z \sim P(z)} P(X|z) \tag{1}$$

This simply means that we try to learn a way to sample X starting from a vector of values z (this is $P(X|z)$), where z is distributed with a **know prior** distribution $P(z)$. z is the **latent encoding** of X.



Latent space and visible space



Suggested reading:

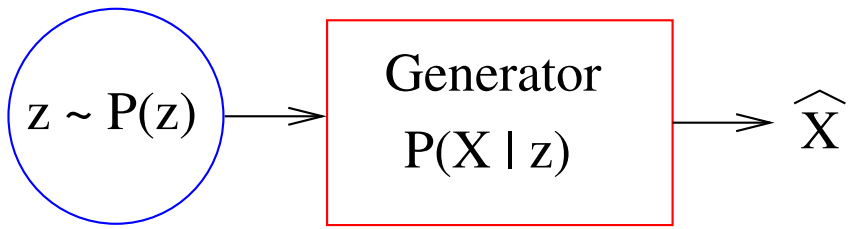
Comparing the latent space of generative models

Generative models

There are four main classes of generative models:

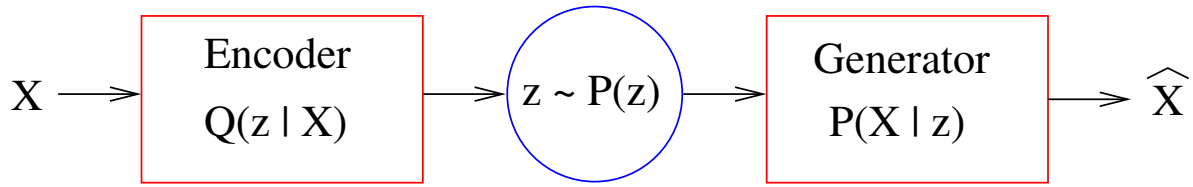
- compressive models
  - Variational Autoencoders (VAEs)
  - Generative Adversarial Networks (GANs)
- dimension preserving models
  - Normalizing Flows
  - Denoising Diffusion Models

The models differ in the way the generator is trained



VAE

In a Variational Autoencoder the generator is coupled with an **encoder** producing a latent encoding z given X. This will be distributed according to an inference distribution $Q(z|X)$.

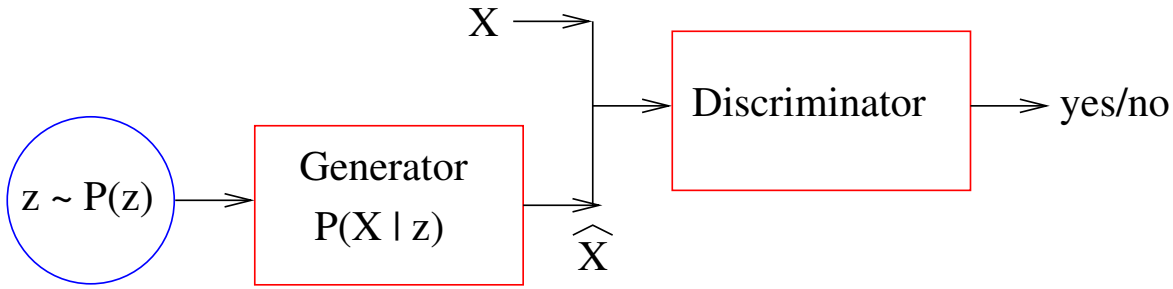


The loss function aims to:

- minimize the reconstruction error between X and $\hat{X}$
- bring the marginal inference distribution $Q(z)$ close to the prior $P(z)$

GAN

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.



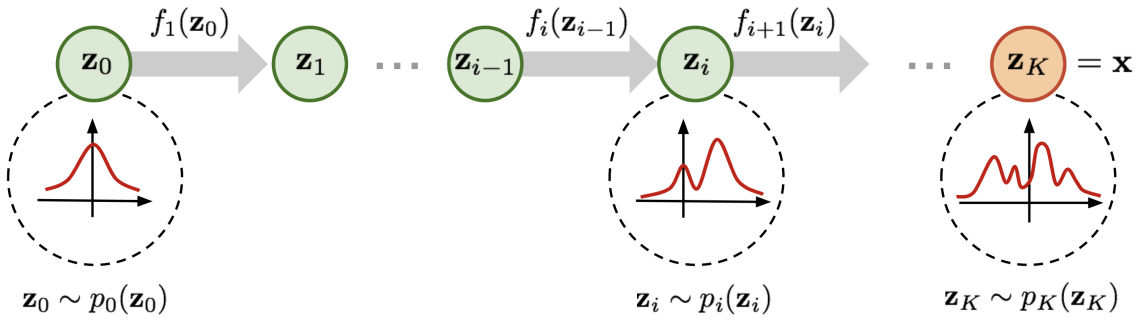
Detector and Generator are trained together.

The loss function aims to:

- instruct the detector to spot the generator
- instruct the generator to fool the detector

Normalizing Flows

In Normalizing Flows the generator is split into a long chain of *invertible* transformations.



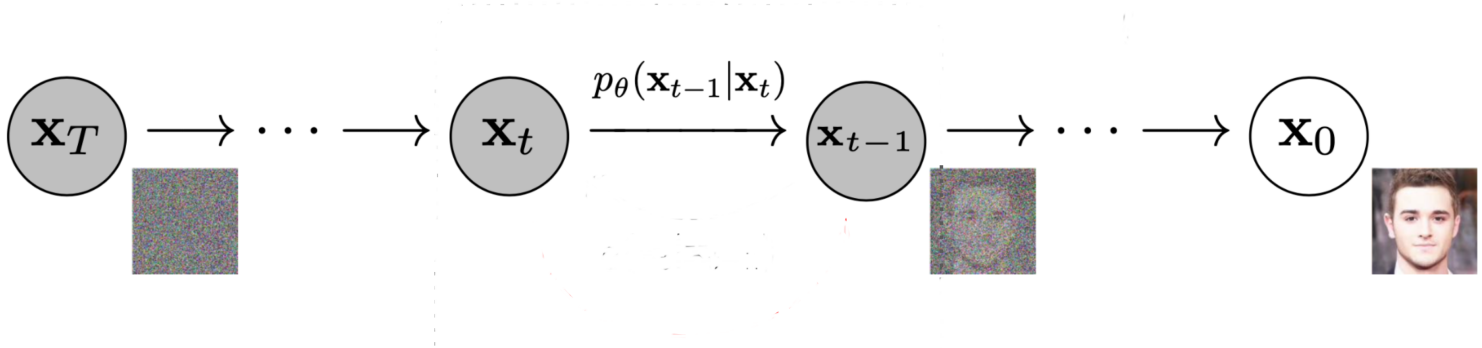
The network is trained by maximizing loglikelihood.

- **Pros**: it allows a precise computation of the resulting loglikelihood
- **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.

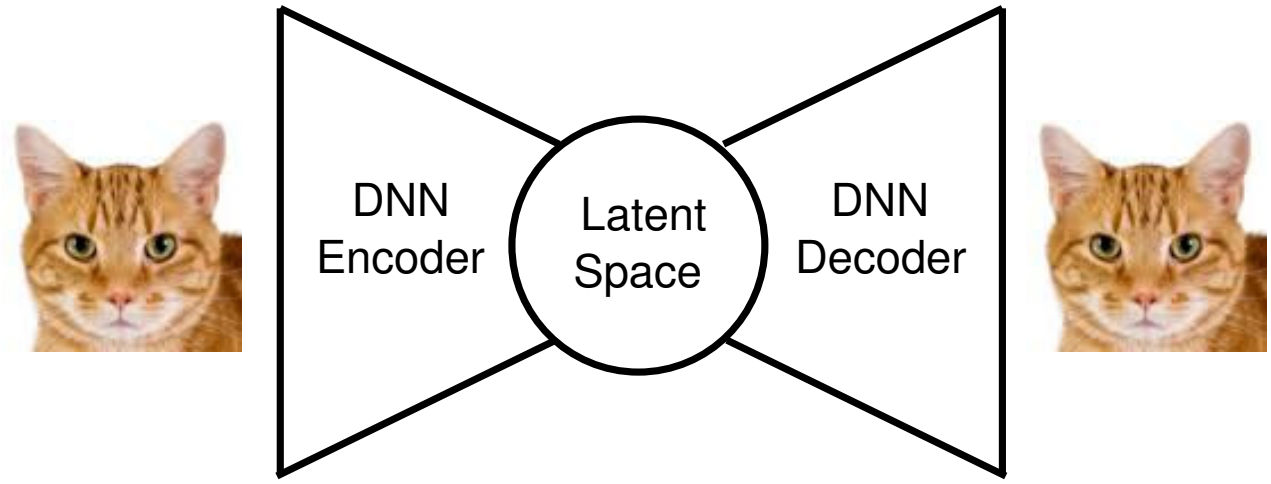
Variational Autoencoders

Suggested reading:

A survey on Variational Autoencoders from a GreenAI Perspective

The problem with the deterministic autoencoder

An autoencoder is a net trained to reconstruct input data out of a learned internal representation (e.g. minimizing quadratic distance)

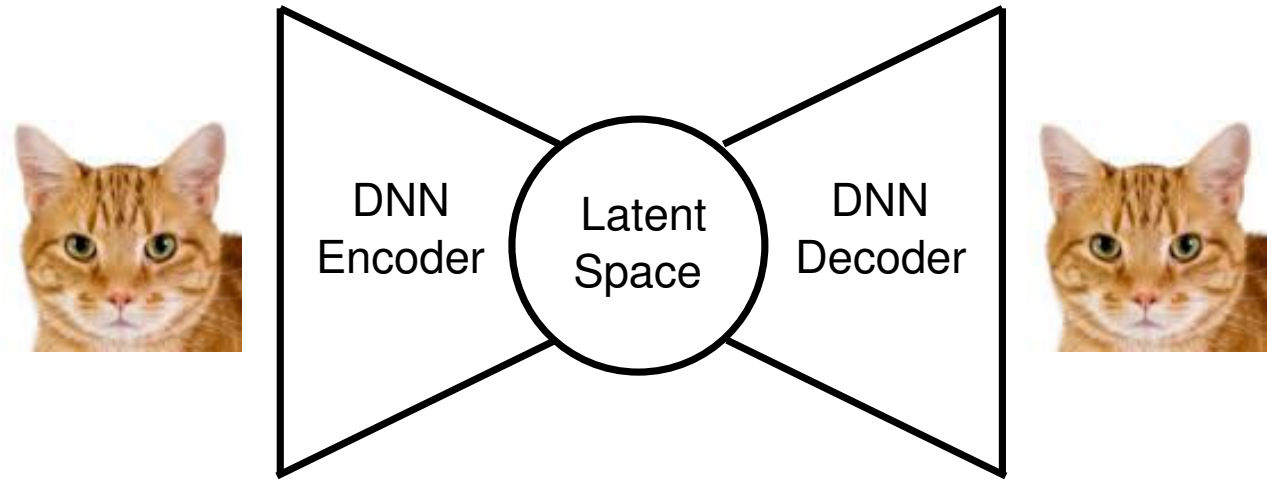


Can we use the decoder to **generate** data by **sampling** in the latent space?

No, since we do not know the distribution of latent variables.

The problem with the deterministic autoencoder

An autoencoder is a net trained to reconstruct input data out of a learned internal representation (e.g. minimizing quadratic distance)

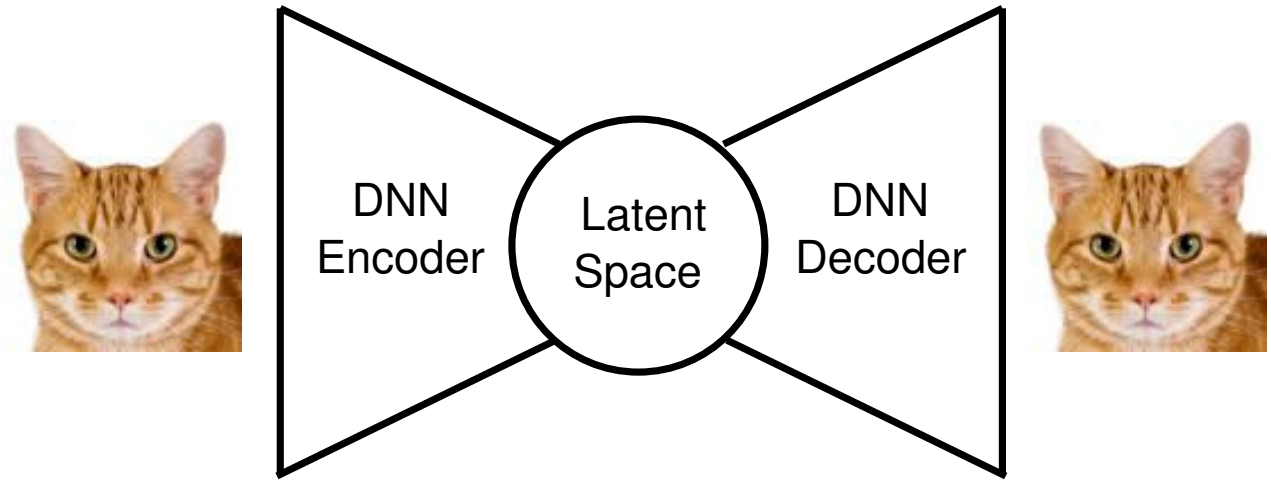


Can we use the decoder to **generate** data by **sampling** in the latent space?

No, since we do not know the distribution of latent variables.

The problem with the deterministic autoencoder

An autoencoder is a net trained to reconstruct input data out of a learned internal representation (e.g. minimizing quadratic distance)

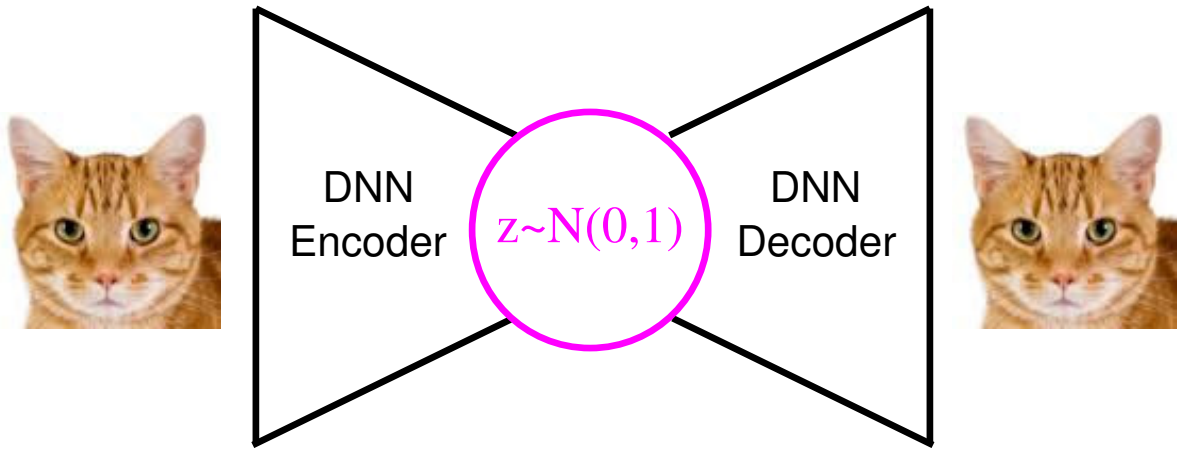


Can we use the decoder to **generate** data by **sampling** in the latent space?

No, since we do not know the distribution of latent variables.

Variational autoencoder

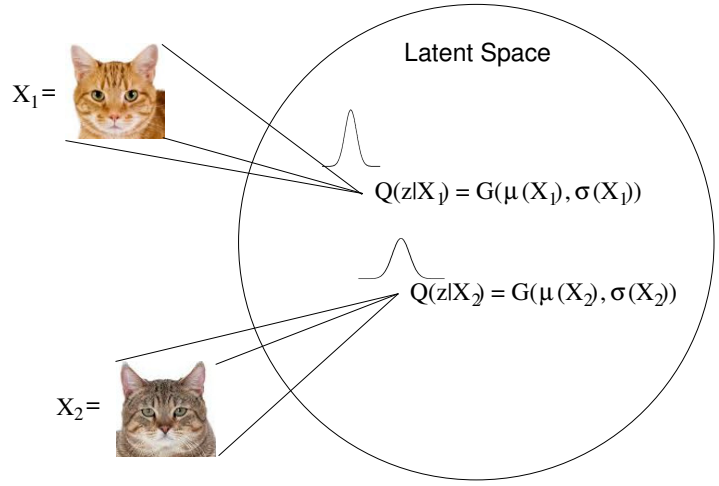
In a Variational Autoencoder (VAE) we try **to force** latent variables to have a known prior distribution $P(z)$ (e.g. a Normal distribution)



If the distribution computed by the generator is $Q(z|X)$ we try to force the marginal distribution $Q(z) = \mathbb{E}_{X \sim P_{data}} Q(z|X)$ to look like a normal distribution.

Different moments for each point

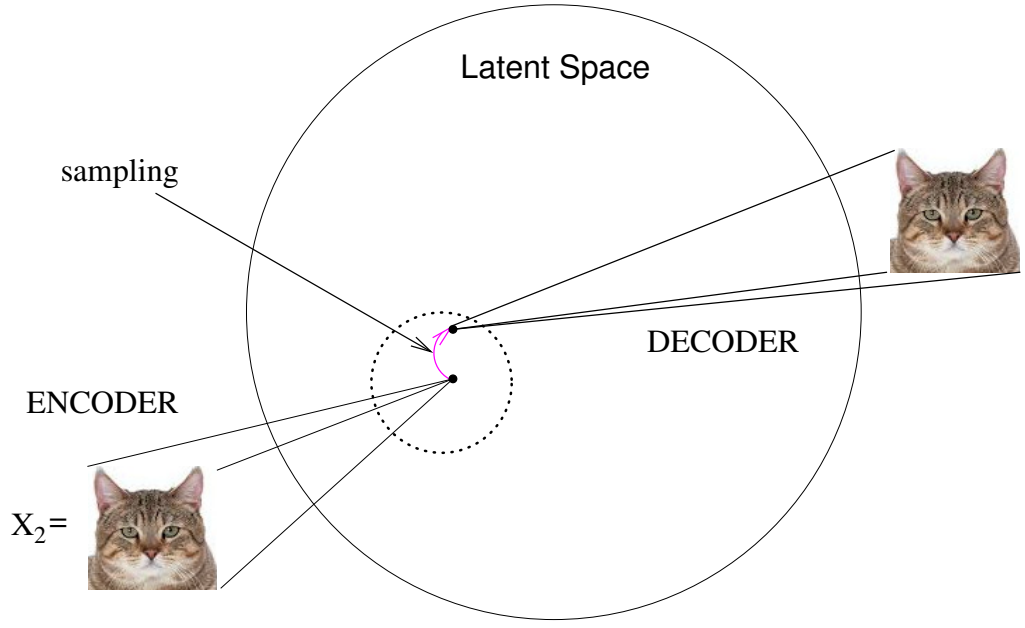
We assume $Q(z|X)$ has a Gaussian distribution $G(\mu(X), \sigma(X))$ with different moments for each different input X.



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

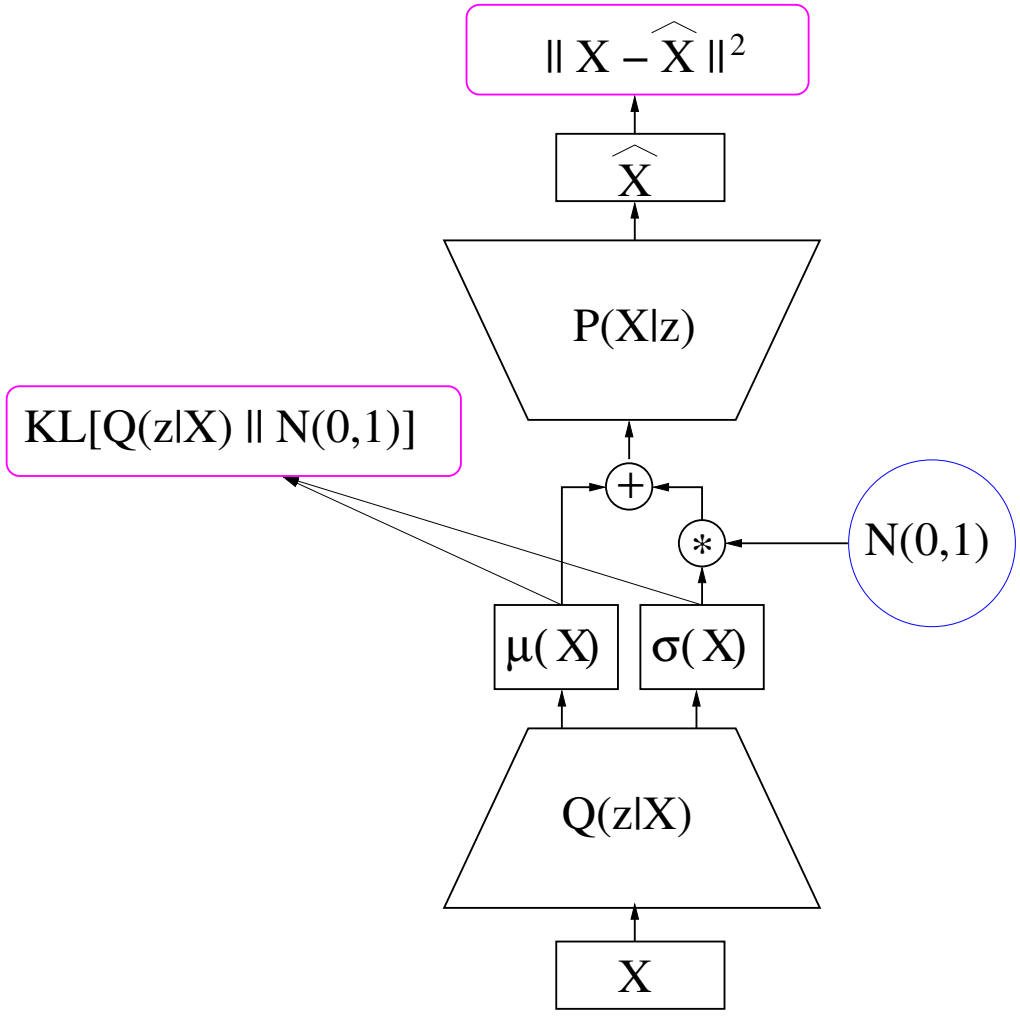
Sampling in the latent space

During training, we sample around $\mu(X)$ with the computed $\sigma(X)$ before passing the value to the decoder



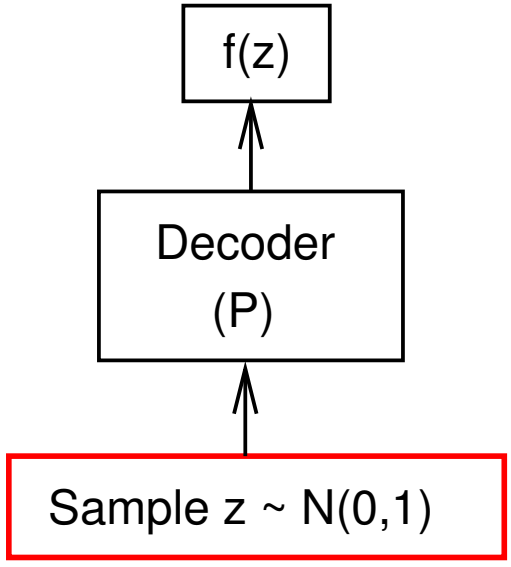
Among other things, sampling add noise to the encoding, improving its robustness.

The full picture

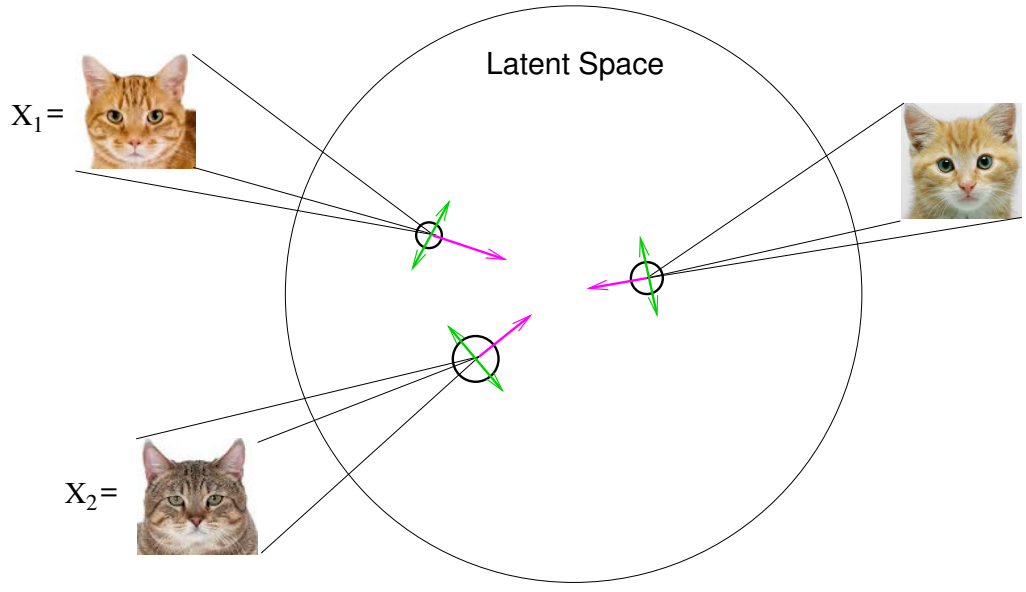


Generation of new samples

$\mu(X)$ and $\Sigma(X)$ are not used to generate new samples from the input domain (we have no X)



The effect of KL-divergence



The effect of the KL-divergence on latent variables consist in

- pushing $\mu_z(X)$ towards 0, so as to center the latent space around the origin
- push $\sigma_z(X)$ towards 1, augmenting the "coverage" of the latent space, essential for generative purposes.

Problems with VAE

- balancing loglikelihood and KL regularizer in the loss function
- variable collapse phenomenon
- marginal inference vs prior mismatch
- blurriness (aka variance loss)

VAE Demo

DEMO!

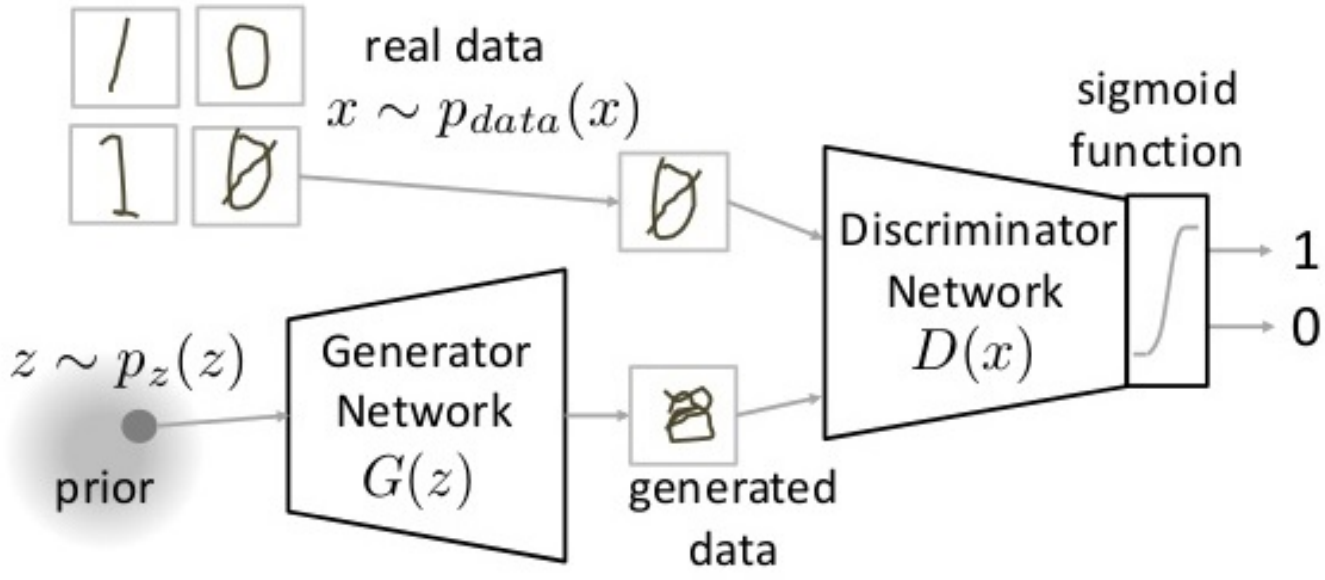
Generative Adversarial Networks

Suggested reading:

NIPS 2016 Tutorial: Generative Adversarial Networks

The GAN approach: a two player game

A game between the generator and the discriminator



Generative Adversarial Networks I.J.Goodfellow et al., 2014

A Min Max game

$$Min_G Max_D V(D, G)$$

$$V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[log\ D(x)] + \mathbb{E}_{z \sim p_z(z)}[log\ (1 - D(G(z)))]$$

- $\mathbb{E}_{x \sim p_{data}(x)}[log\ D(x)]$ = negative cross entropy of the discriminator w.r.t the true data distribution
- $\mathbb{E}_{z \sim p_z(z)}[log\ (1 - D(G(z)))]$ = negative cross entropy of the "false" discriminator w.r.t the fake generator

Training

Alternately train the discriminator, freezing the generator, and the generator freezing the discriminator:

---

for number of training iterations **do**

  for k steps **do**

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

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

  end for

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

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

end for

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

An example

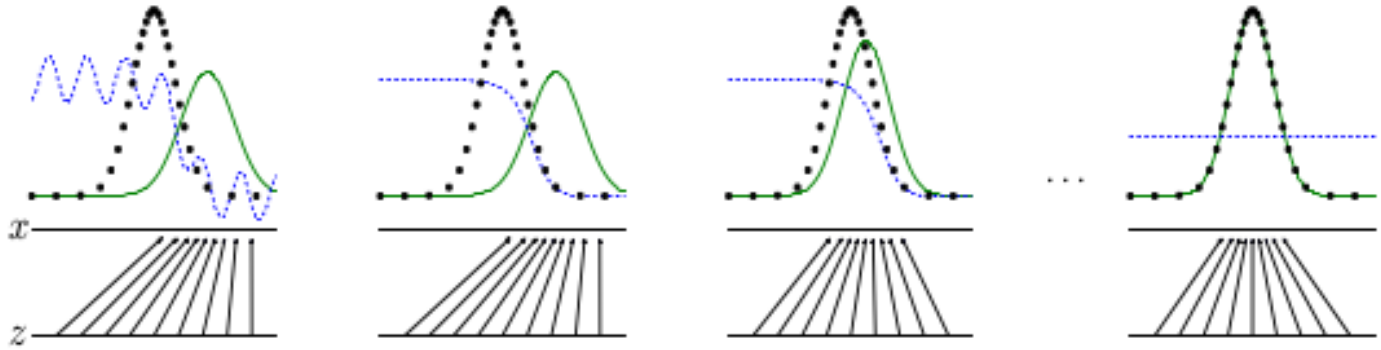


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

A simple demo

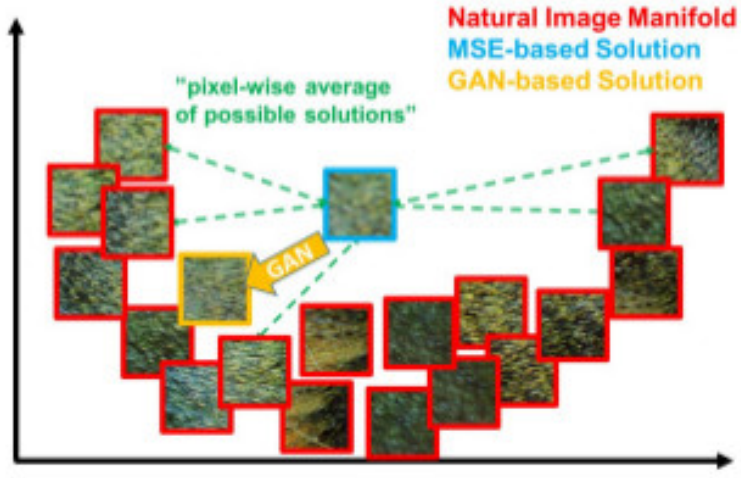
Demo!

Stay inside the data manifold

Patches from the natural image manifold (red) and super-resolved patches obtained with MSE (blue) and GAN (orange).

Pixel-wise average of possible solutions could produce images outside the actual data manifold.

GAN drives the reconstruction towards the natural image manifold producing perceptually more convincing solutions.



picture from Photo-Realistic Single Image Super-Resolution. C.Ledig et al., 2016.

An application: super-resolution

Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

Forcing to operate a choice - instead of mediating - could result in sharper images

Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. C.Ledig et al., 2016.

An application: face generation

Goal: Generation of plausible realistic photographs of human faces.

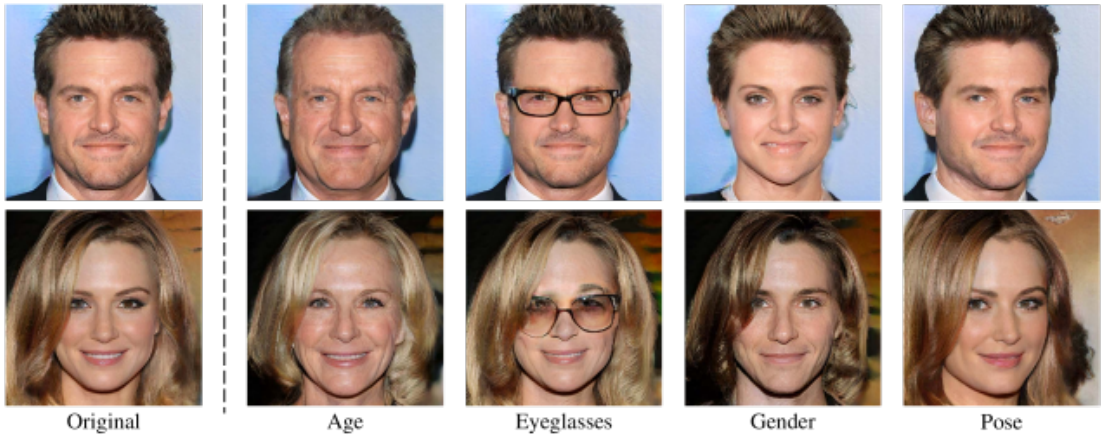
Face generation video by Nvidia

Problems with Gans

- the fact that the discriminator get fooled does not mean the fake is good (neural networks are easily fooled)
- problems with counting, perspective, global structure, ...
- **mode collapse**: generative specialization on a good, fixed sample

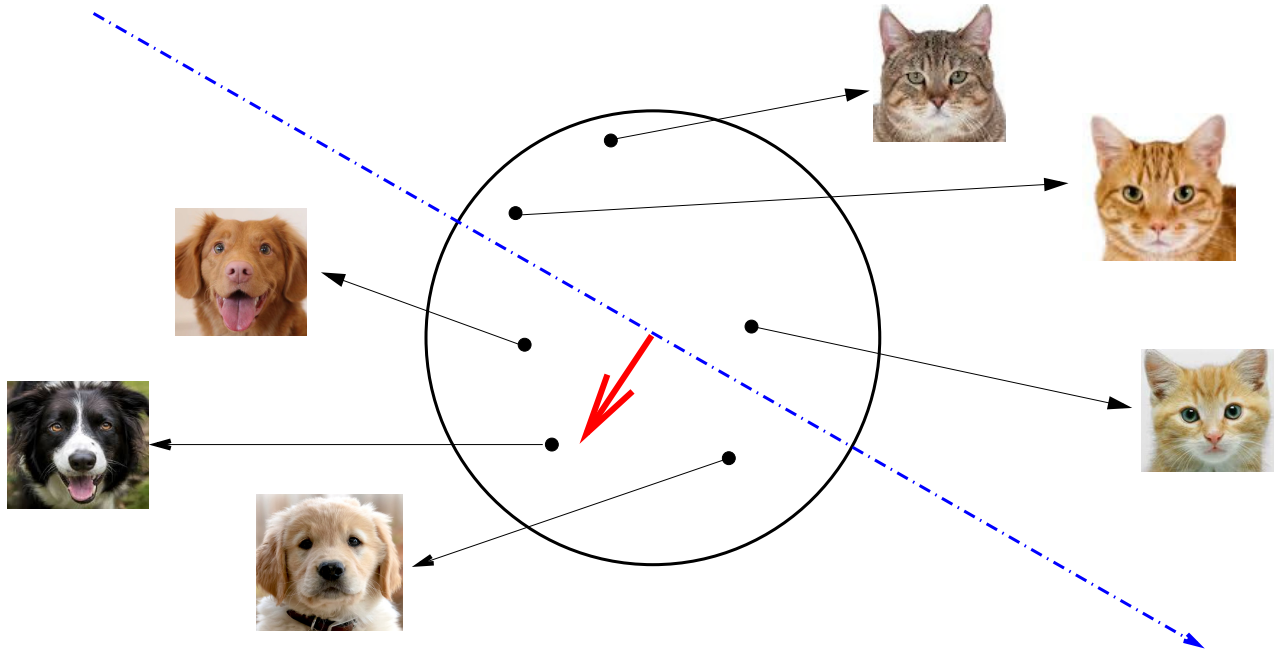
See Ian Goodfellows' slides

Latent space exploration



Interpreting the Latent Space of GANs for Semantic Face Editing

Attribute editing



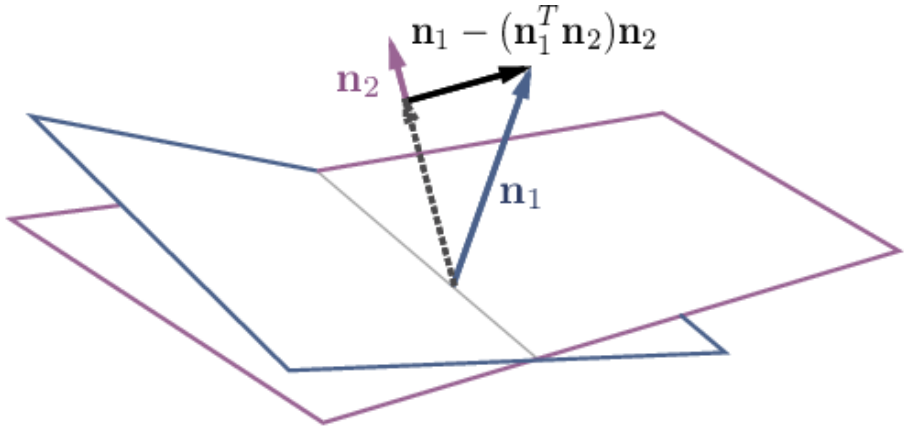
Key ideas behind Representation Learning

The generative process is **continuous**: a small deplacement in the latent space produces a small modification in the visible space.

Real-world data depends on a relatively **small number of explanatory factors of variation** (latent features) providing compressed internal representations.

Understanding these features we may define **trajectories** producing desired alterations of data in the visible space.

Entanglement and disentanglement

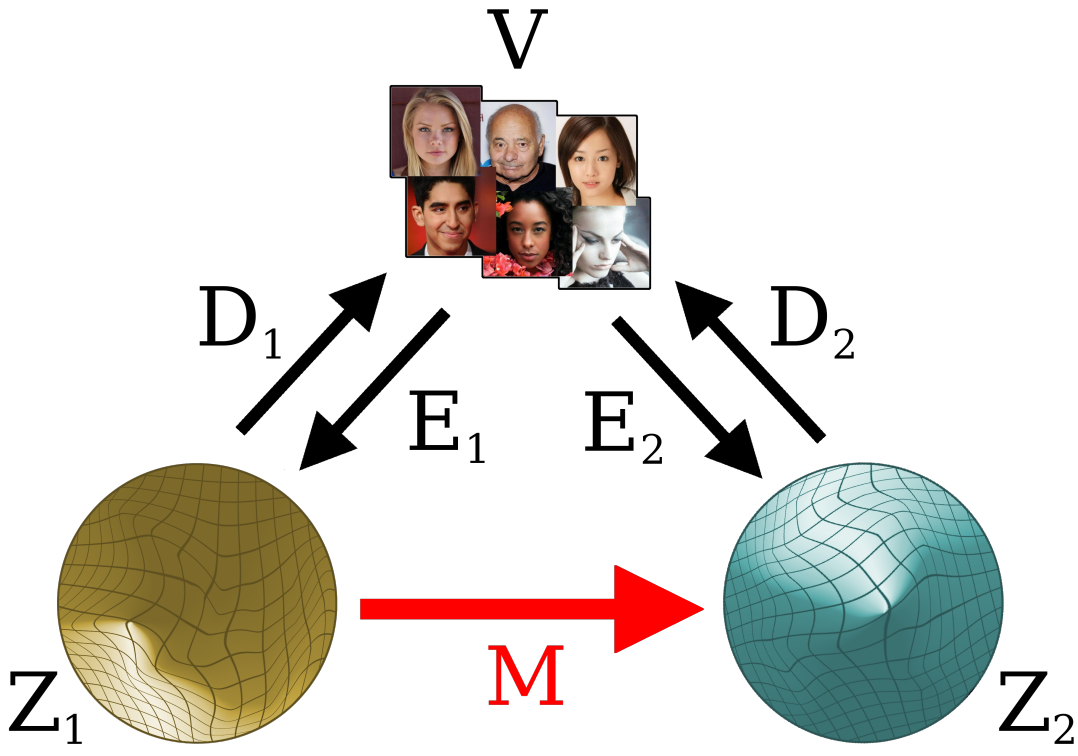


When there is more than one attribute, editing one may affect another since some semantics can be coupled with each other (entanglement).

To achieve more precise control (disentanglement), we can use projections to force the different directions of variation to be orthogonal to each other.

Comparing spaces

Learn a direct map between spaces



Comparing the latent space of generative models

Achievements

We can pass from a latent space to another by means of a simple **linear map** preserving most of the content.

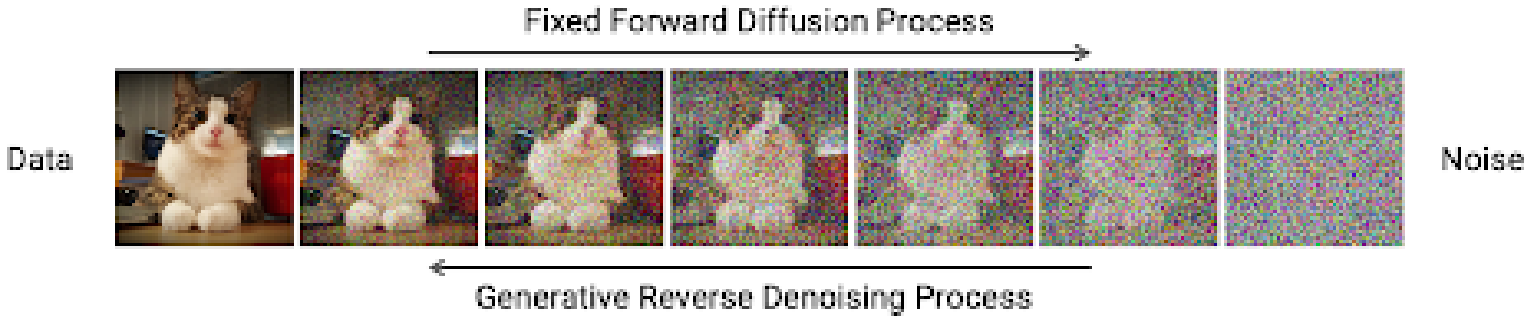
The organization of the latent space seems to be independent from

- the training process
- the network architecture
- the learning objective: GAN and VAE share the same space!

The map can be defined by a small set of points common to the two spaces: the **support set**. Locating these points in the two spaces is enough to define the map.

See also my blog for a discussion.

Diffusion models



Suggested reading: What are diffusion models

The denoising network

The denoising network implements the inverse of the operation of adding a given amount of noise to an image (direct diffusion).

The denoising network takes in input:

1. a noisy image x_t
2. a signal rate α_t expressing the amount of the original signal remaining in the noisy image

and try to predict the noise in it:

$$\epsilon_\theta(x_t, \alpha_t)$$

The predicted image would be:

$$\hat{x_0} = (x_t - \sqrt{1-\alpha_t} \cdot \epsilon_\theta(x_t, \alpha_t))/\sqrt{\alpha_t}$$

Training step

- take an input image x_0 in the training set and normalize it
- consider a **signal ratio** α_t
- generate a random noise $\epsilon \sim N(0, 1)$
- generate a noisy version x_t of x_0 defined as

$$x_t = \sqrt{\alpha_t} \cdot x_0 + \sqrt{1 - \alpha_t} \cdot \epsilon$$

- let the network predict the noise $\epsilon_\theta(x_t, \alpha_t)$ from the noisy image x_t and the signal ratio α_t
- train the network to minimize the prediction error, namely

$$\|\epsilon - \epsilon_\theta(x_t, \alpha_t)\|$$

Sampling procedure

- fix a sheduling $\alpha_T > \alpha_{T-1} > \ldots\ldots > \alpha_1$
- start with a random noisy image $x_T \sim N(0,1)$
- for t in T...1 do:
- compute the predicted error $\epsilon_\theta(x_t, \alpha_t)$
- compute $\hat{x_0} = (x_t - \sqrt{1-\alpha_t} \cdot \epsilon_\theta(x_t, \alpha_t))/\sqrt{\alpha_t}$
- obtain x_{t-1} reinjecting noise at rate α_{t-1}, namely

$$x_{t-1} = \sqrt{\alpha_{t-1}} \cdot \hat{x_0} + \sqrt{1-\alpha_{t-1}} \cdot \epsilon$$

Network architecture

Use a (conditional) Unet!