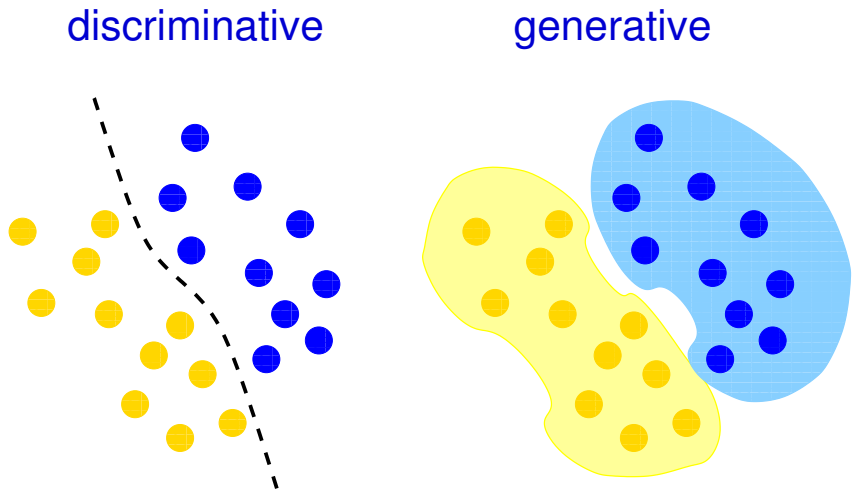
Generative Models



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}

Generative models mostly focus on the second approach.

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
- ...

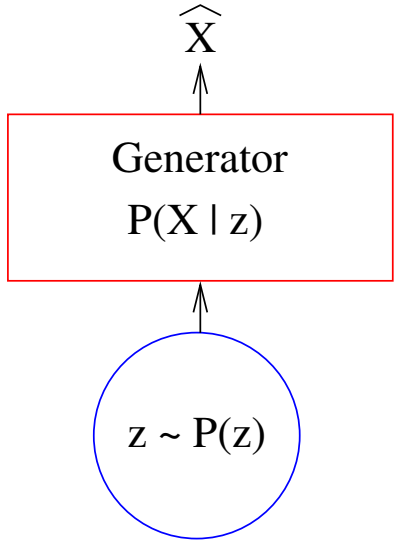
When the output is intrinsically multi-modal (and we do not want to give up to the possibility to produce multiple outputs) we need to rely on generative modeling.

Latent variables models

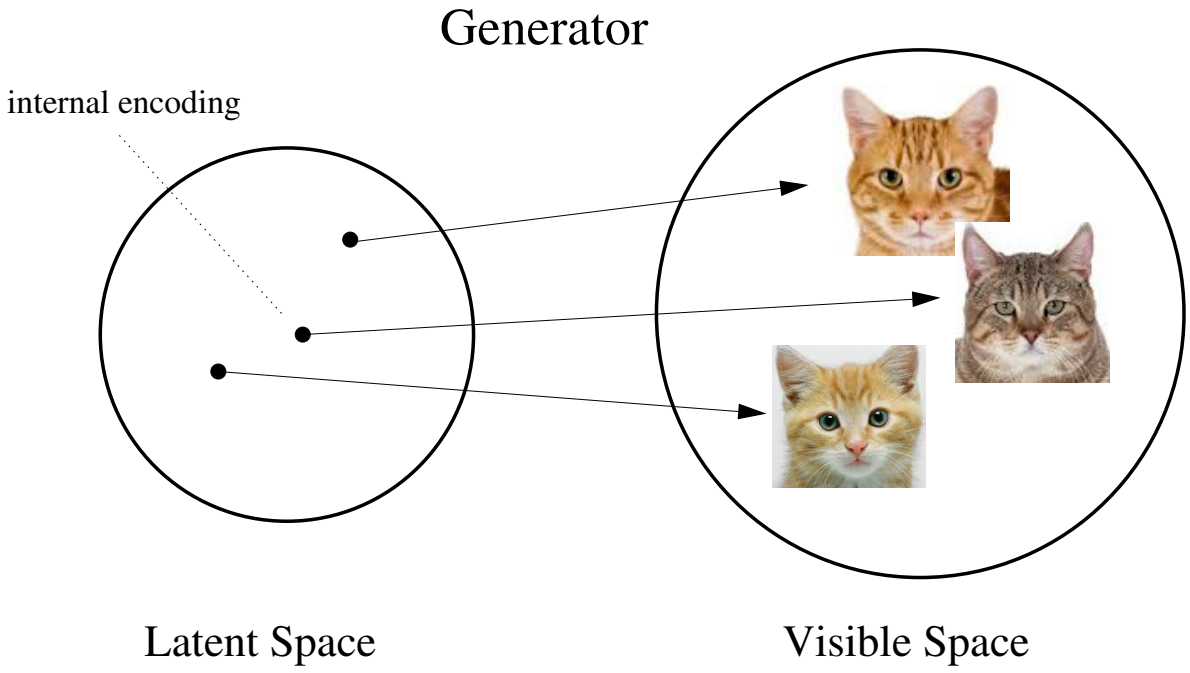
In latent variable models we express the probability of a data point X through **marginalization** over a vector of latent variables:

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

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



Suggested reading:

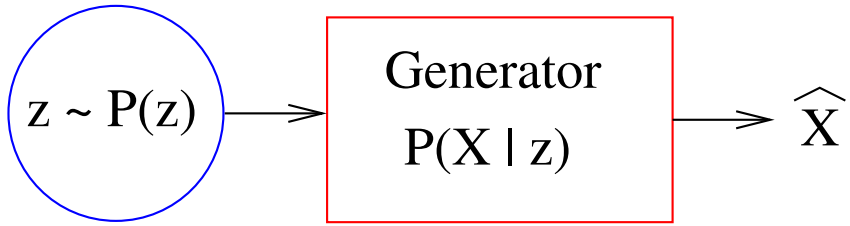
Comparing the latent space of generative models

VAE and GAN

There are two main classes of geneative models:

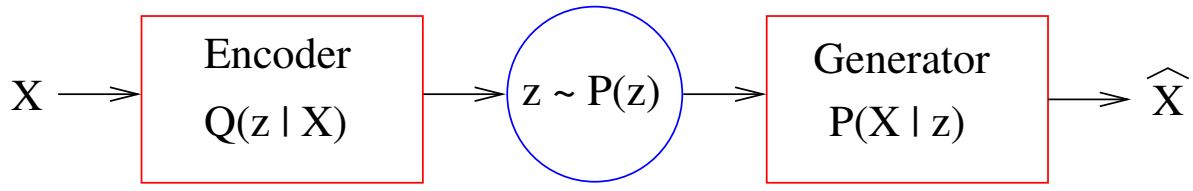
- Variational Autoencoders (VAEs)
- Generative Adversarial Networks (GANs)

The two classes 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:

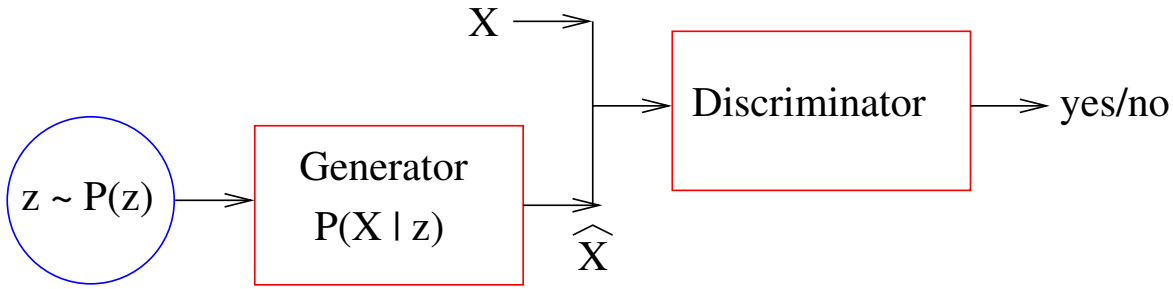
- minimize the reconstruction error between X and $\widehat{X}$
- bring the marginal inference distribution $Q(z)$ close to the prior $P(z)$

Suggested reading:

A survey on Variational Autoencoders from a GreenAI Perspective

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

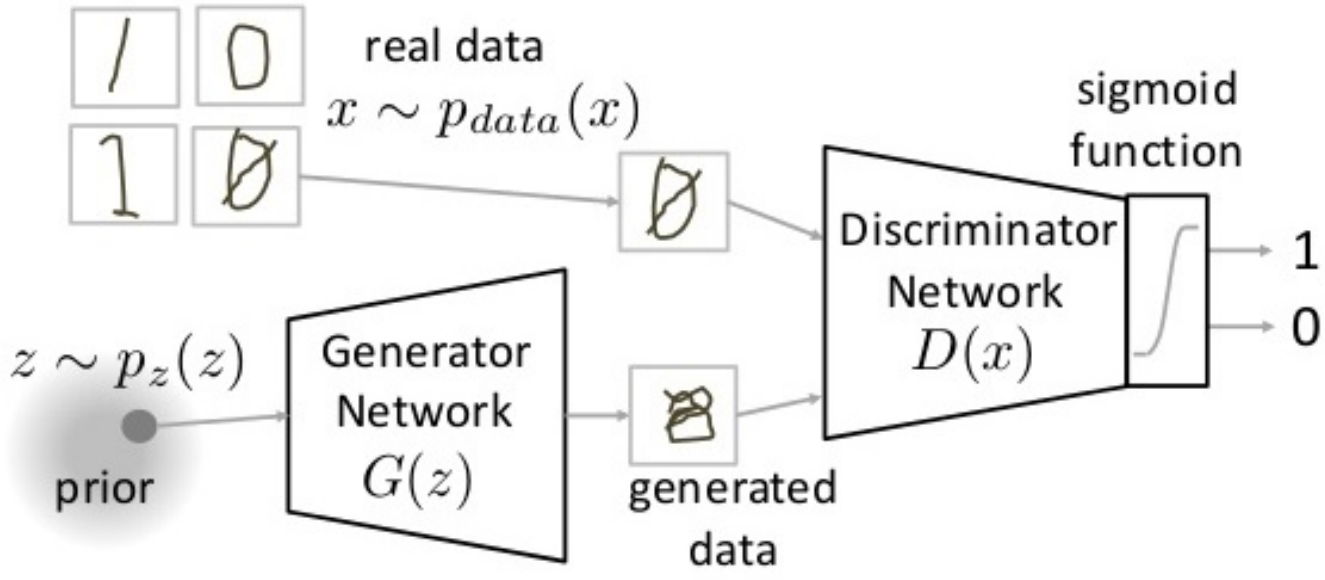
Generative Adversarial Networks

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

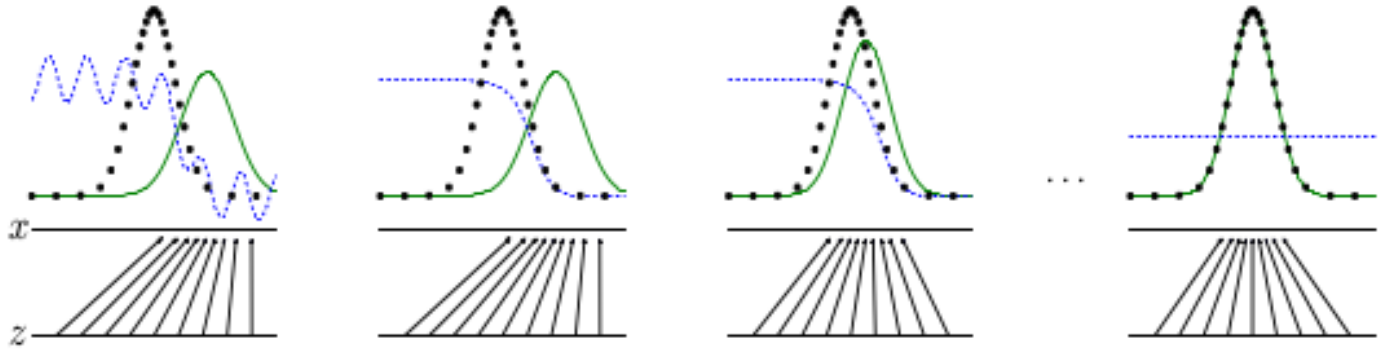


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution 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

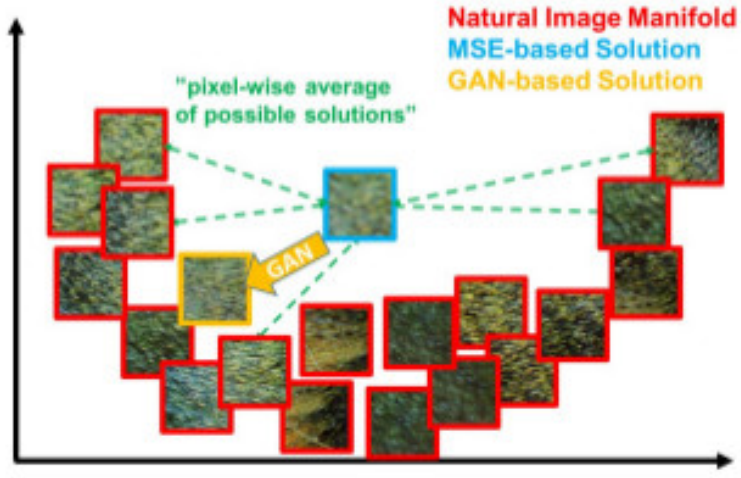
Demo!

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

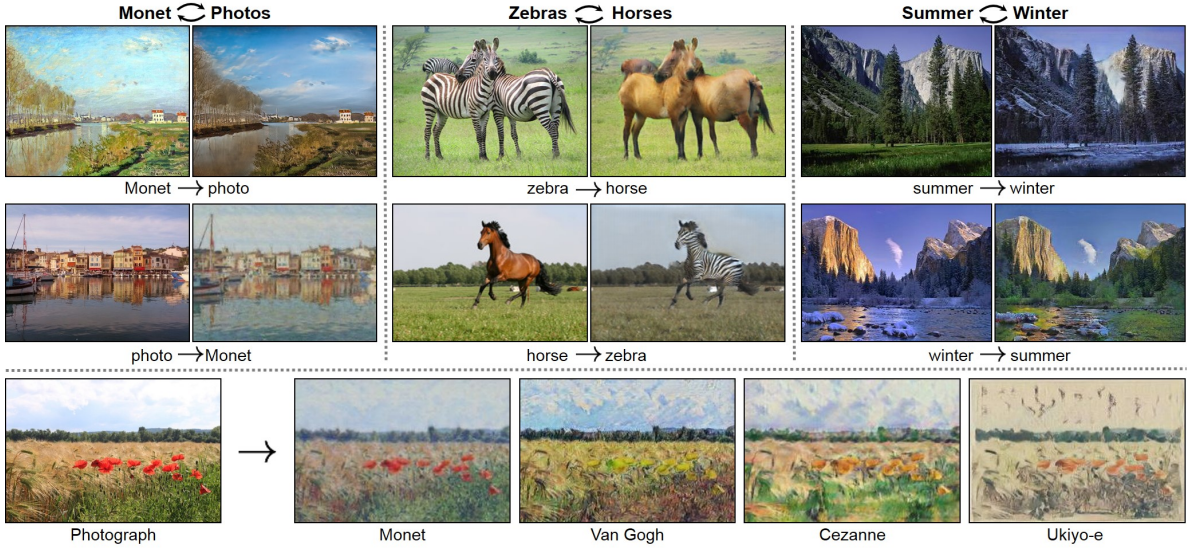
Problems with Gans

- 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. No need to mimic the actual data distribution

See Ian Goodfellows' slides

Next arguments

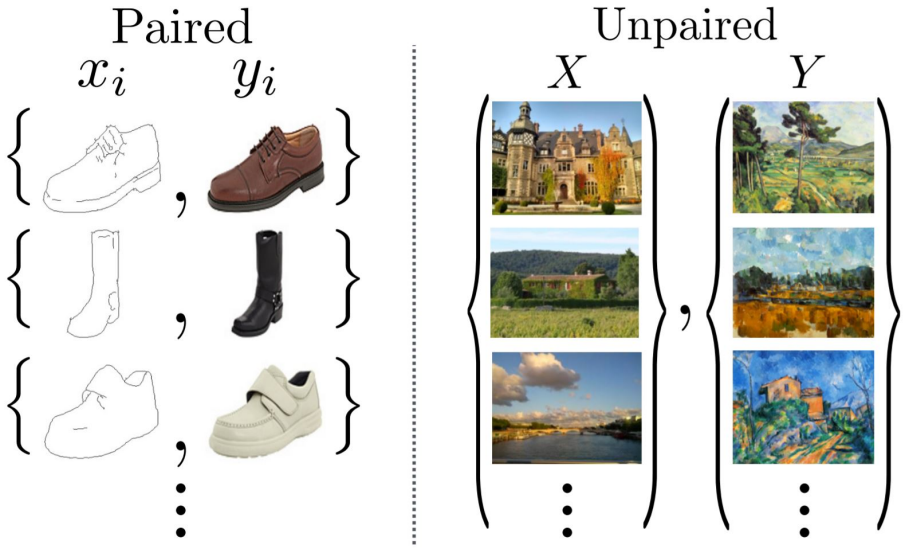
Cycle gans



Unpaired Image-to-image translation with Cycle Generative Adversarial Networks

Unpaired images

For training, we need images in two classes, but **not** paired images



picture from article

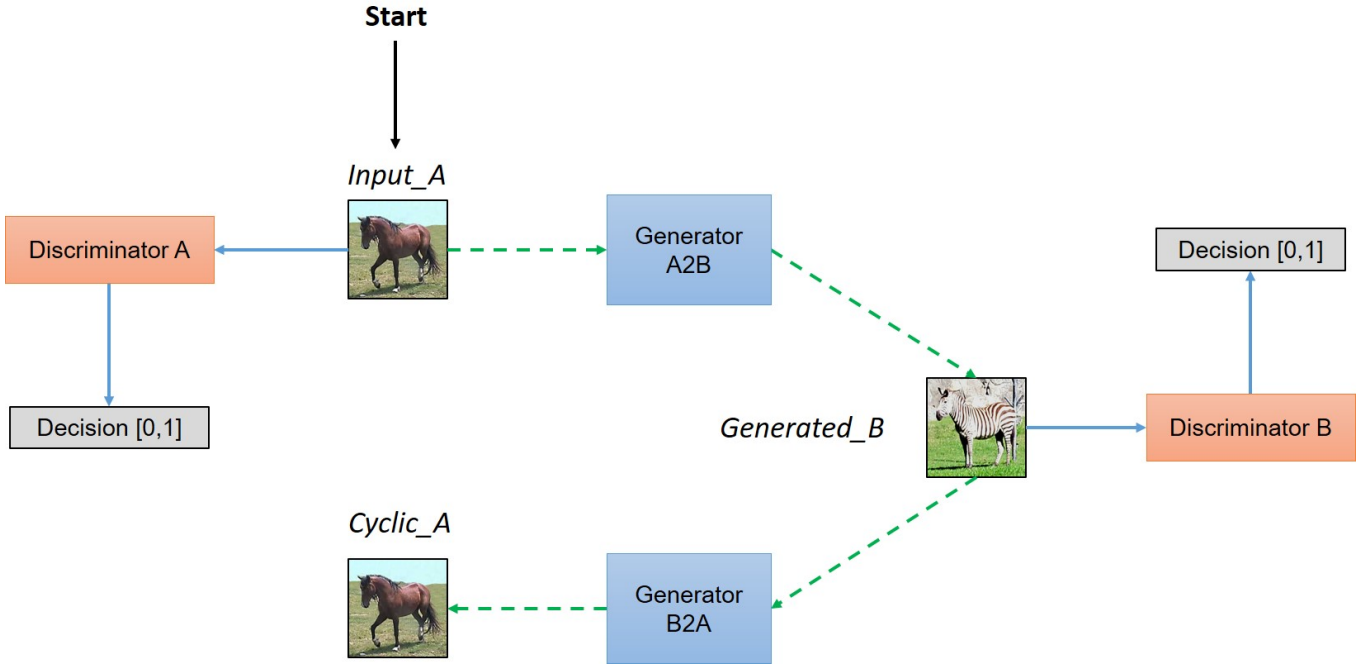
Cycle-consistent adversarial networks

A Generator and a Discriminator network playing against each other. The generator tries to produce samples from the desired distribution, based on an input of the other distibution, and the discriminator tries to predict if the sample belongs to the actual distribution or it was produced by the generator.

This cannot guarantee to generate an image in the target distribution related to the original image.

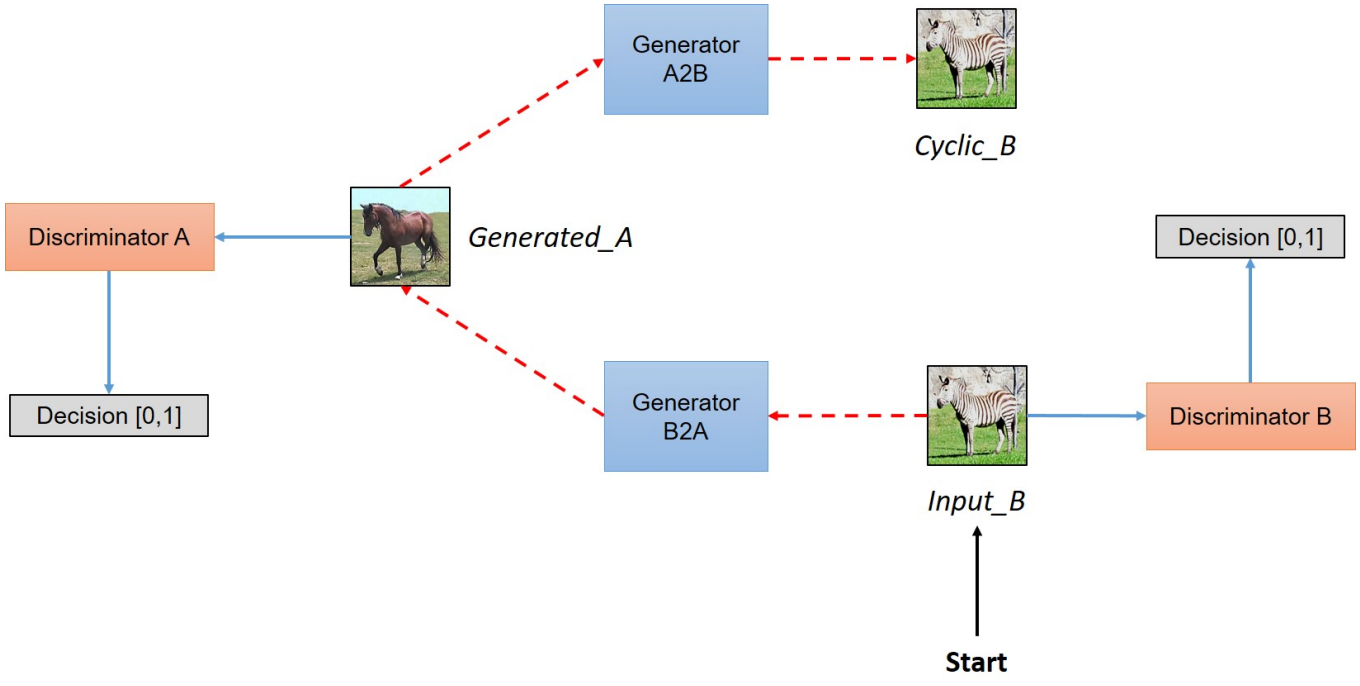
To this aim, the authors introduce the cycle-consistency constraint: if we transform from source distribution to target and then back again to source distribution, we would expect to obtain the original image.

The network architecture: forward ...



Pictures from Understanding and Implementing CycleGAN in TensorFlow

and back



Pictures from Understanding and Implementing CycleGAN in TensorFlow

The loss function

Given mappings $G : X \to Y$, $F : Y \to X$ and discriminators D_Y, D_X we have:

- Adversarial loss

$$\begin{aligned}\mathcal{L}_{GAN}(G; D_Y) &= \mathbb{E}_{y \sim p_{data}(y)}[log(D_Y(y)] \\ &+ \mathbb{E}_{x \sim p_{data}(x)}[log(1 - D_Y(G(x)))]\end{aligned}$$

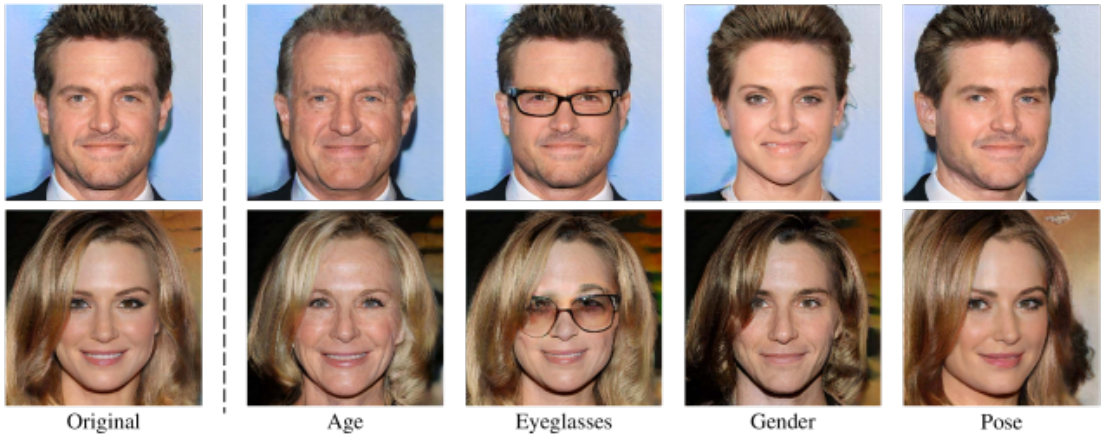
- Cycle consistency loss

$$\begin{aligned}\mathcal{L}_{Cyc}(G; F) &= \mathbb{E}_{x \sim p_{data}(x)}[\|F(G(x)) - x\|_1] \\ &+ \mathbb{E}_{y \sim p_{data}(y)}[\|G(F(y)) - y\|_1]\end{aligned}$$

The network structure

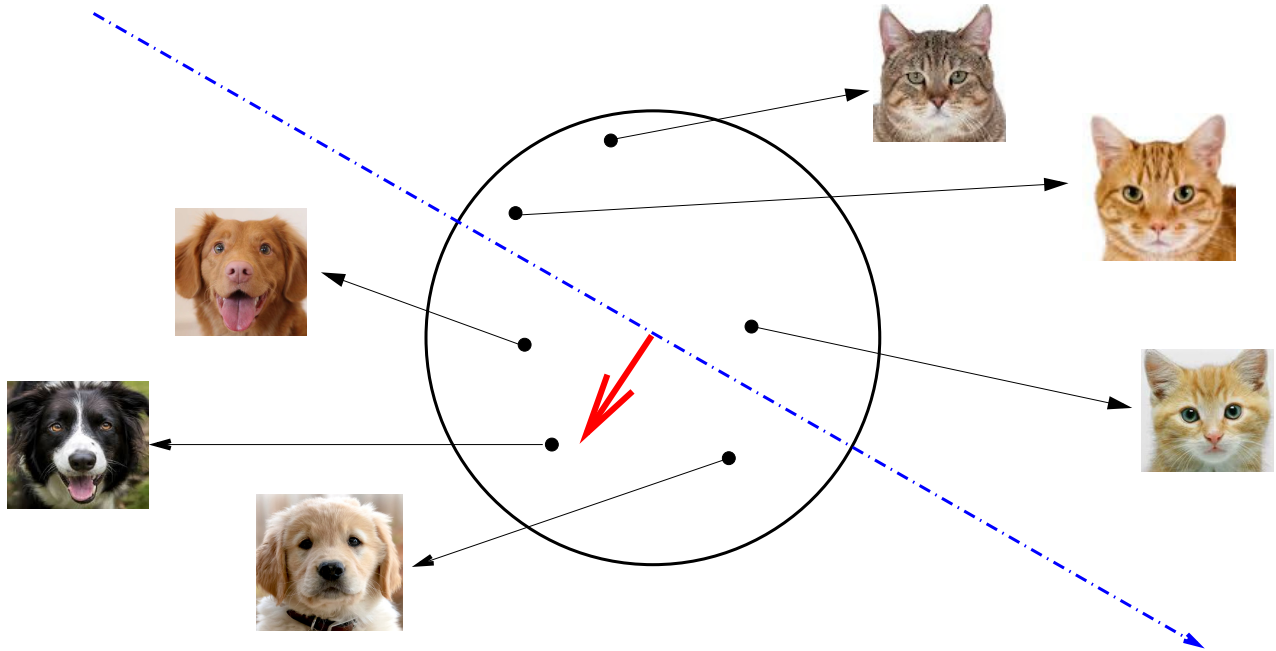
- each mapping is defined as a sequence of convolutions followed by transposed convolutions
- to improve similarity bewteen input and output, the authors also exploit residuality

Latent space exploration



Interpreting the Latent Space of GANs for Semantic Face Editing

Attribute editing



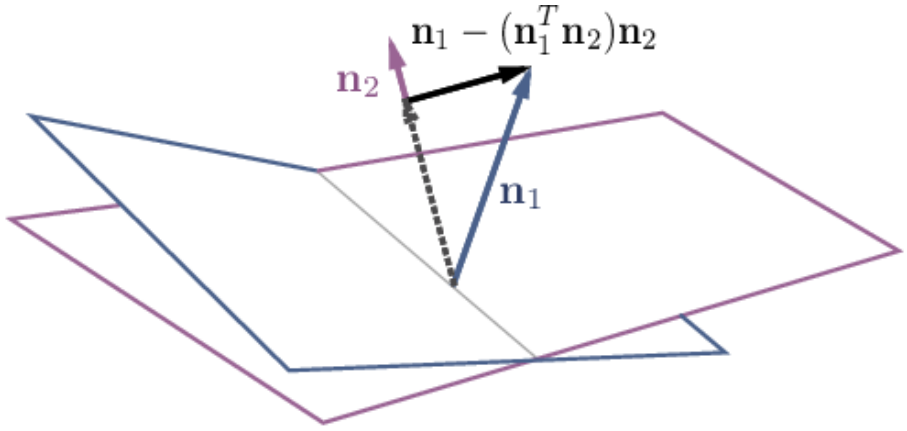
Key ideas behind Representation Learning

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

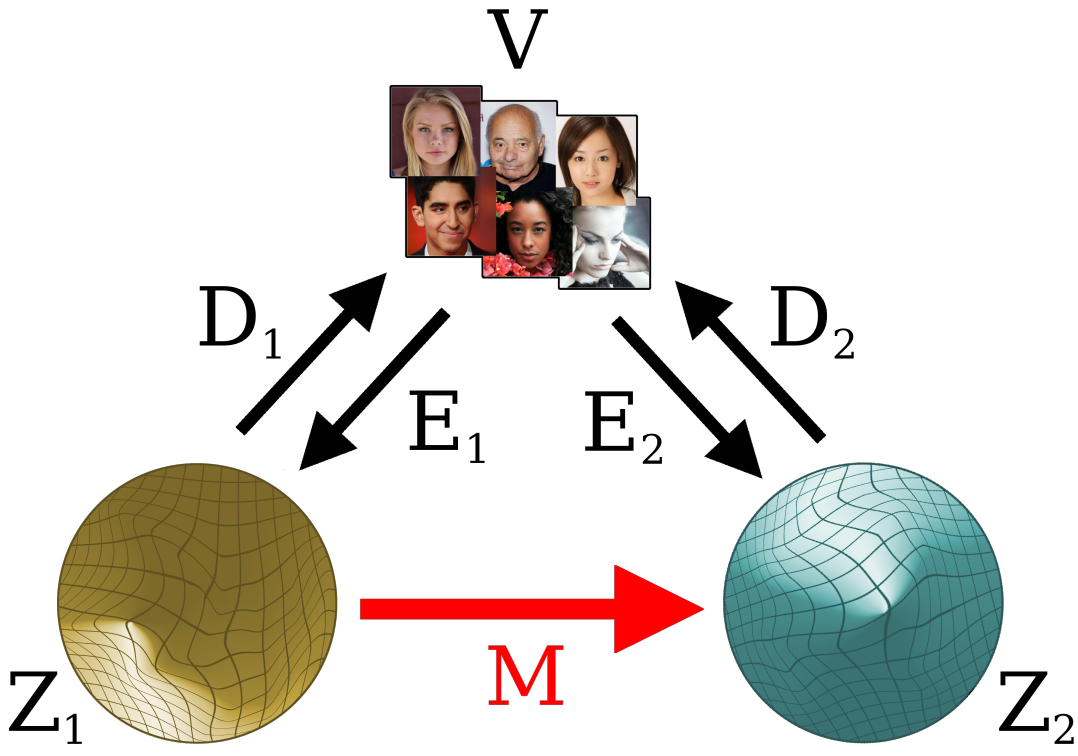


When there is more than one attribute, editing one may affect another since some semantics can be coupled with each other (entanglement).

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.