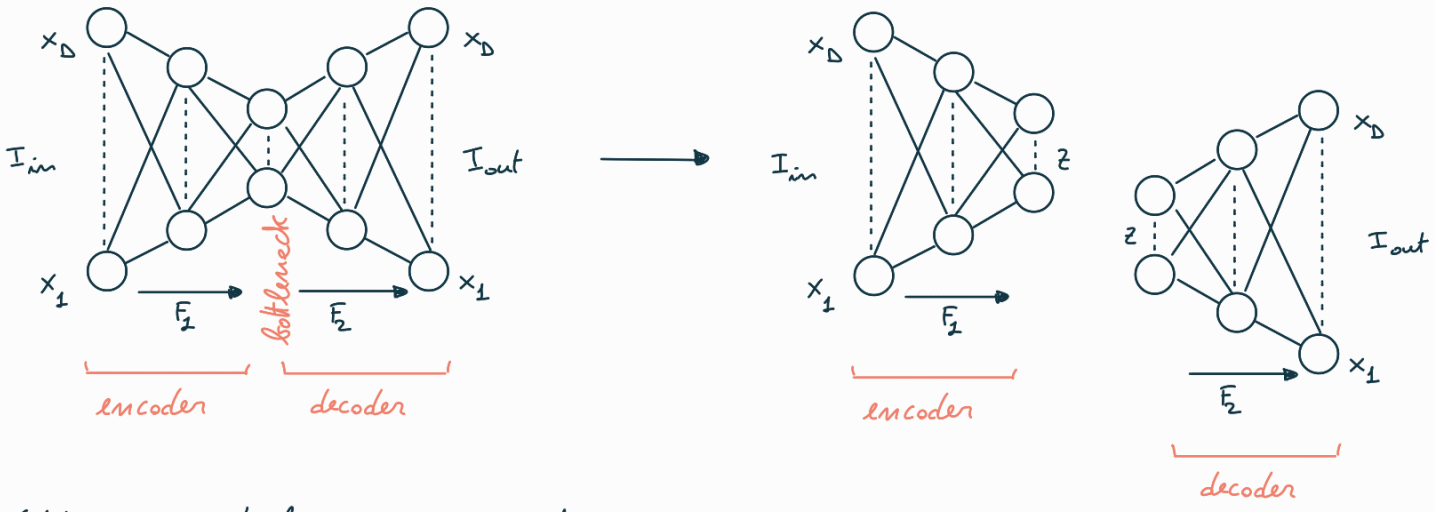
Generative Adversarial Networks (GANs)



GANs are used for image generation.
The final goal is to generate good data (images) thus we focus on the decoding part.
We can design networks with special features for our tasks.
On top of this, we use Adversarial Training to train the decoder to produce meaningful data.

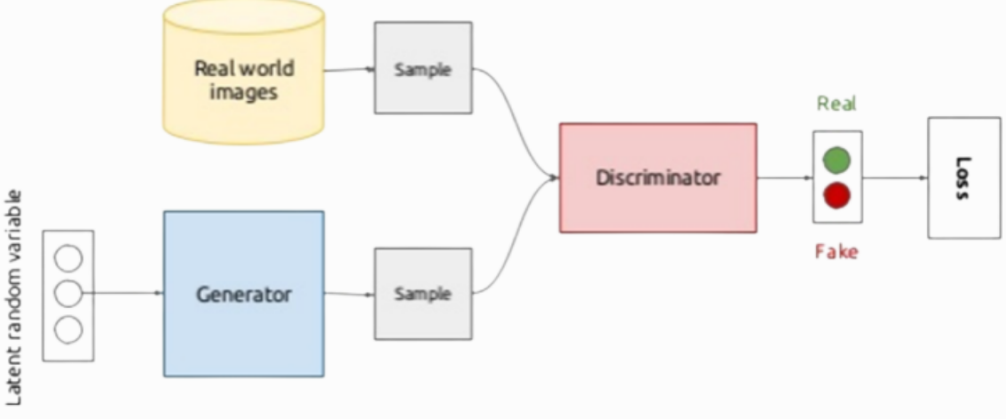
Adversarial Training → two pieces of the network are competing with each other to find a good solution. An improvement for one of these components is a disadvantage for the others. In regular networks all the components share the same goal and an improvement for a component is an improvement for all the others.

The discriminator tries to maximize its reward
The generator tries to minimize Discriminator's reward
} minmax

A GAN is formed by two components

- discriminator
- generator (decoder)

The discriminator is just a binary classifier. It takes as input either an image from the dataset (REAL) or an image produced by the decoder/generator (FAKE) and has to understand if the image is real or not.

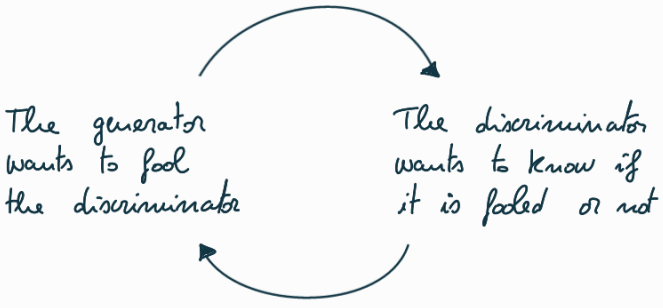


We could start with a generator that is already trained in an autoencoder to speed up the process.

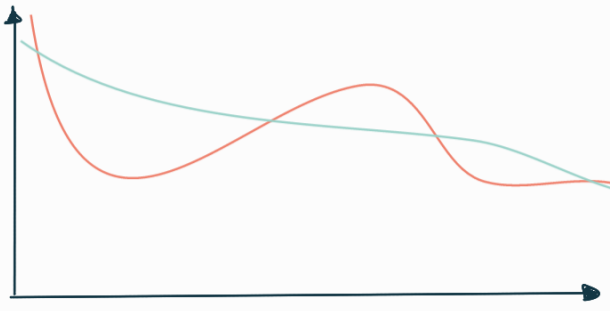
Training procedure:

Repeat:
1. Train the discriminator by keeping the generator fixed
2. Train the generator by keeping the discriminator fixed

When training the discriminator, it knows if the images are REAL or FAKE.
When training the generator, it labels the images that it produces as REAL.
The meaning of the images generated by the generator changes at every step.

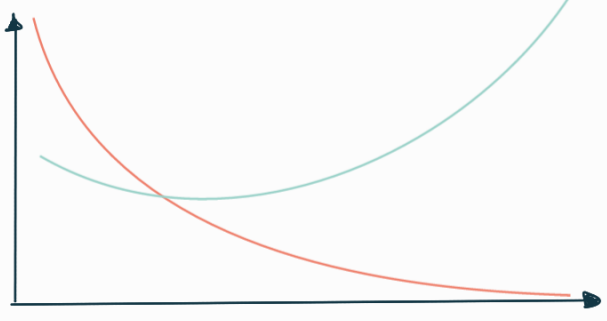


When training the discriminator, the generated images are considered FAKE
When training the generator, the generated images are considered REAL



When the loss for one of the two components increases, it decreases for the other one.

Ideally the two losses will converge.



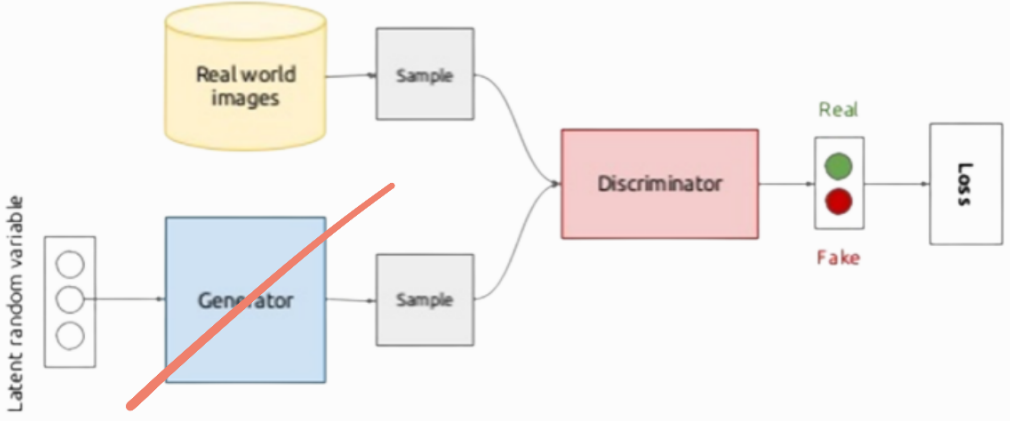
If one loss goes to 0, the other will diverge.
We don't want this to happen

When the two losses are stable (and below some threshold) and the accuracy of the discriminator is around 50% then the model has been trained correctly. It is not easy to train a GAN.
The training usually requires a lot of epochs (the two phases are repeated for each epoch) and it is more difficult than training a CNN or an AE.
As usual we use mini batches (e.g. 128 images) to train the discriminator but here we will divide the batch such that half of that will come from the dataset and the other half will come from the generator;
we will create a standard two-class dataset.

Training procedure:

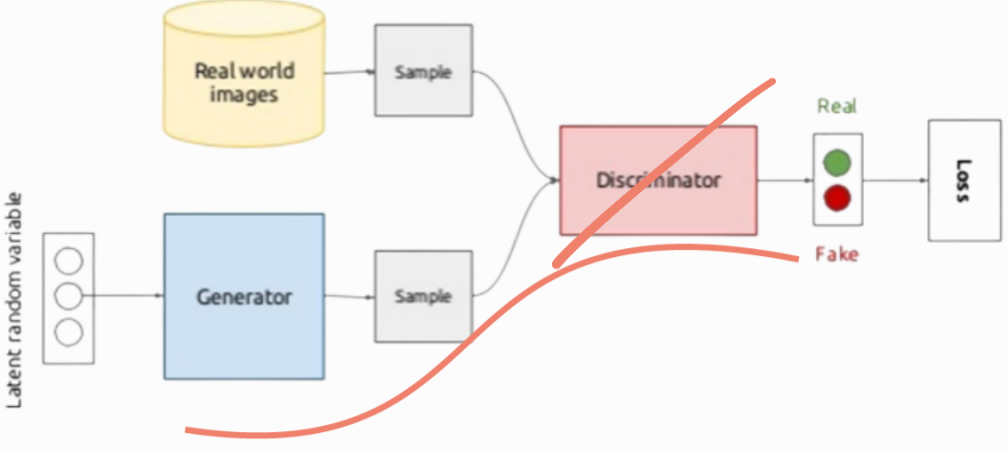
Repeat:

1. Train the discriminator with a batch of data $\{(x_m, \text{REAL})\} \cup \{(x'_m, \text{FAKE})\}$ where x_m comes from the data set, while x_m are images generated from the generator with random values of the latent variable.



The generator is fixed, meaning that the backpropagation is only applied on the discriminator: only the weights of that part of the network are updated

2. Train the generator by using the entire model (generator + discriminator) with discriminator layers fixed (not trainable) with a batch of data $\{(r_k, \text{REAL})\}$, where r_k are random values of the latent variable.



To train the generator we apply the backpropagation on the full model but we do it by keeping the discriminator fixed (freeze its weights).

Problems with GANs

Deep learning models (in general) involve a single player.
The player (agent) tries to maximize its reward (minimize loss)

$$\min_G L(G)$$

GANs instead involve two (or more) players.
Discriminator is trying to maximize its reward.
Generator is trying to minimize Discriminator's reward.

$$\min_G \max_G V(D,G)$$

Non-convergence and mode-collapse are common issues that can arise during training.

Training problems

- Non-convergence: the GAN fails to converge to an equilibrium
- Mode-collapse: occurs when the generator produces limited or repetitive samples, often ignoring a significant portion of the target data distribution. In this situation the generator may neglect to explore and capture the full diversity of the target distribution.
  Consequently the generated samples become visually similar or even identical, resulting in lack of variation and diversity in the generated output.
  To summarize: the generator produces limited or repetitive samples

Solutions

Generator produces good samples but very few of them. We are at mode collapse

→ let the discriminator look at an entire batch of samples instead of a single one. If there is lack of diversity, it will mark the examples as fake. Generator will be forced to produce diverse samples.

↓ extract features that capture diversity in the mini-batch, e.g. with L2 norms of the difference between all pairs from the batch. Use them along with the images in the discriminator. Will force the generator to match those feature values with the real data.

Empirically, label information of the real data might help and yields much better results.

Cycle GAN

Cycle GAN (Cycle-Consistent Adversarial Networks) is a machine learning model that enables the unsupervised learning of image-to-image translation task. Introduced in 2017, has since gained significant attention in the field of computer vision and it is often used as a baseline for several other models.

Image-to-image mapping is an old problem. It was tackled before, but always with paired image data : models are trained on both the original image and the corresponding required image after translation. Creating such a dataset is a pain and existing datasets are usually too small. We want a model that works on unpaired image data.

$$G : X \longrightarrow Y$$

mapping between two domains