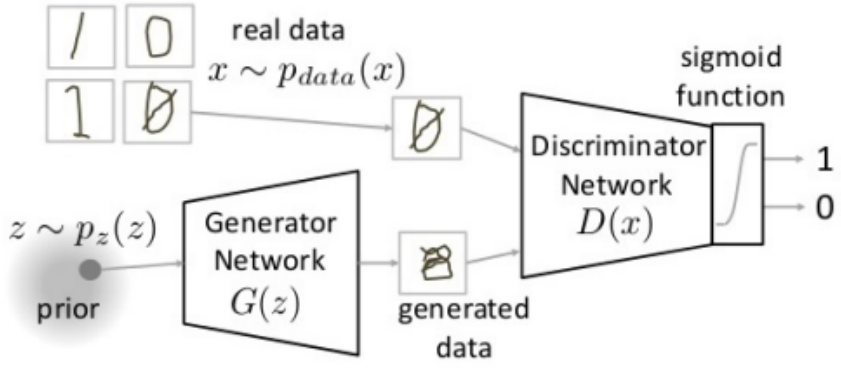
GAN

As we know, in a GAN, we want to train a generator, and then, we want to generate data that is similar to what is present in our dataset.
We then have a discriminator network, which will output 0 or 1 accordingly.



GAN's loss function From a mathematical standpoint, we could see this process as a MinMax game:

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

The generator wants to Minimize G...
...while the discriminator wants to Maximize D.

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

Loss function

The loss functoin has 2 parts: 1. negative cross entropy of the discriminator w.r.t the true data distribution 2. negative cross entropy of the "false" discriminator w.r.t the fake generator. - This is also the only part in which the Generator has an influence on the loss.

The training must be done alternately for the *discriminator* and the *generator*. Otherwise, our generator will always be shitty. In particular, we alternately train the discriminator, freezing the generator, and the generator freezing the discriminator.

We could see this process as having a student (generator) and a teacher (discriminator).

Example

In this example, we have a Green curve (associated with our generator/generated data) and a black curve (associated to the real data distribution). We could see the line x as the latent space, while the line z is the real data distribution.

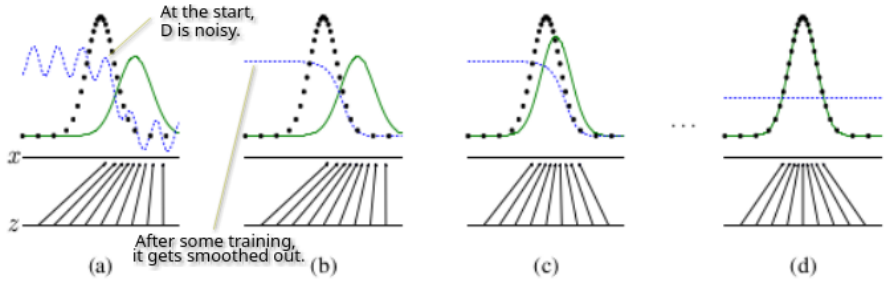


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution 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. $D(\boldsymbol{x}) = \frac{1}{2}$.

One problem: since the purpose of the generator is just to fool the discriminator, the generator could find *a single picture* that would *fool the discriminator* and always choose that picture, ingnoring the other points in the latent space, so there's no guarantees that its generations will be varied. This is called the **Mode Collapse problem**. We will see later how to solve this.

Properties and applications of GAN

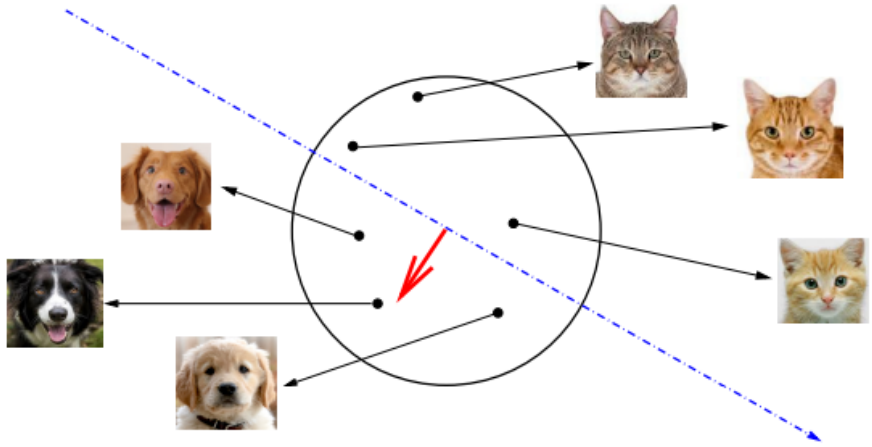
Staying inside the data manifold GAN drives the reconstruction towards the natural image manifold producing perceptually more convincing solutions wrt MSE-based solutions. So, essentially, *GANs images arent subject to the manifold problem.*

GANs problems

- ==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, ...
  - i.e. horse with 6 legs
- **mode collapse**: generative specialization on a good, fixed sample.

Latent space exploration

Small movements in the latent space can result in some small modification in the



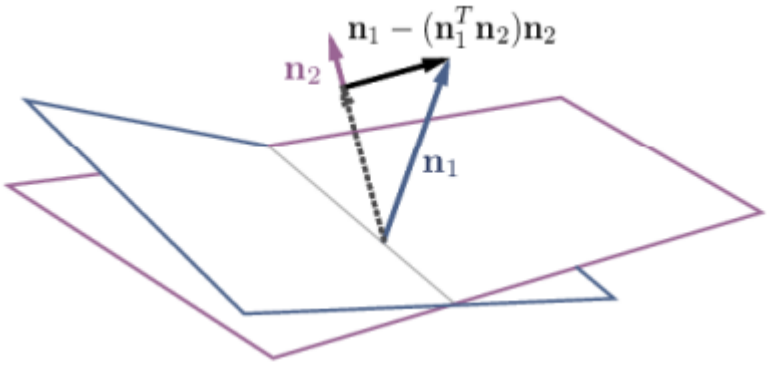
visible image.
For example, what if we wanted to transform a picture of a cat into a picture of a dog, we take all the latent representation of cats that we know, and we find the hyperplane which separates the pictures of the dogs from the picture of the cats. After that, we move into the perpendicular direction of the hyperplane.

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

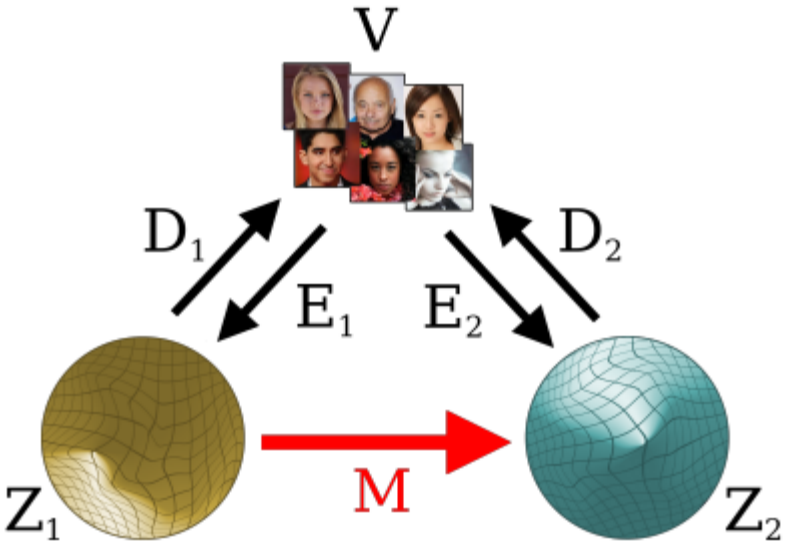
One of the problem of latent exploration/modification of attributes is that many times, these attributes that we're are interested in are not so **disentangled**. - i.e. the transformation that allows a picture to have glasses makes also the person older etc.

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

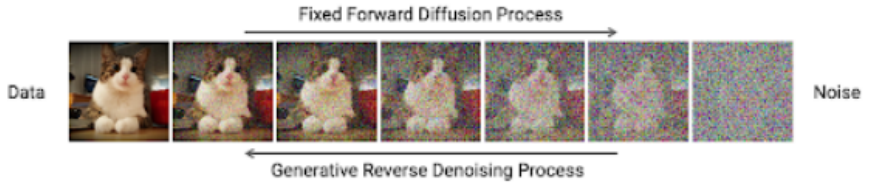
Another cool experiment is the comparison of latent spaces between GANs.



Frequently, we can map the latent space of the same GAN (trained differently) by using a simple linear mapping, and 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. The latent space mostly depends on the dataset.

Diffusion Models



In diffusion models, we have to do the reverse of what is called a **forward diffusion process**. In this type of processes, we essentially are distributing loss on a matrix of pixel, so by reversing that we are instead *denoising* the image.

After the image is denoised, we restart from this guess of what the real image would be, and then we reinject the noise at a smaller rate an we try to remove the noise again.

The number of steps highly depends on the model.

The denoising network

The denoising network implements the inverse of the operation of adding a given amount of noise to an image (direct diffusion) -> it removes a *specific amount of noise* from the image. 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: $\hat{x_0} = (x_t - \sqrt{1-\alpha_t} \cdot \epsilon_\theta(x_t, \alpha_t))/\sqrt{\alpha_t}$

In other words, it receives in input a noise quantity and tries to remove that quantity of noise from the image.

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 $\epsilon \sim N(0, 1)$

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

- generate a *noisy version* x_t of x_0 defined as
- 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 $||\epsilon - \epsilon_\theta(x_t, \alpha_t)||$

– meaning, the actual noise minus what was predicted.

Sampling procedure

With T as the number of steps: - fix a *sheduling* $\alpha_T > \alpha_{T-1} > ... > \alpha_1$ - 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 the current approximation of the result, that is the predicted image formula: $\hat{x_0} = (x_t - \sqrt{1-\alpha_t} \cdot \epsilon_\theta(x_t, \alpha_t))/\sqrt{\alpha_t}$ - obtain x_{t-1}

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

reinjecting noise at rate α_{t-1}, namely

A network for denoising

Use a (conditional) Unet, since it is very good for image to image transformations.

We could see this denoising process as sort of collapsing the space over the data points that we're interested in.