1 Prerequisites

1.1 Inverse transform method

With the *inverse transform method*, we can represent a difficult random variable as the result of a function applied to a uniform random variable (which is usually easy to generate).

In this example (in 1D), let X be a difficult random variable (which is hard to sample from but we know the distribution function of it) and U be a uniform random variable over the interval $[0, 1]$.

As we know, a random variable is defined by its Cumulative distribution function. The CDF of a random variable is a function from the domain of the definition of the variable to the interval $[0, 1]$, and in 1D is defined as:

$$CDF_X(x) = P(X \leq x) \in [0, 1]$$

In other words: $CDF_X(x)$ is the probability that the random variable X is smaller (in its respective distribution) than the parameter x.

For the uniform random variable U, we have:

$$CDF_U(u) = P(U \leq u) = u \in [0, 1]$$

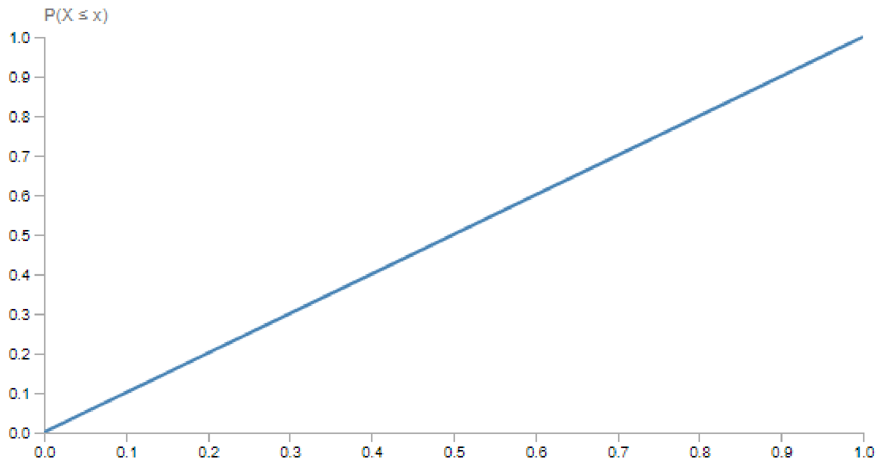

Fig.1: Plotted CFD_U

We will suppose the function CDF_X is invertible and its inverse is denoted by CDF_X^{-1}. Then we define:

$$Y = CDF_X^{-1}(U)$$

Now, we can use this inverse CDF to define

$$Y = CDF_X^{-1}(U)$$

which results in:

$$CDF_Y(y) = P(Y \leq y) = P(CDF_X^{-1}(U) \leq y) = P(U \leq CDF_X(y)) = CDF_X(y)$$

This means that X and Y define the same random variable, i.e. they both follow the same distribution. As we have defined Y as a function of the uniform random variable U, we can sample from U (which is easy) and then compute X.

In other words: The inverse transform method is a way to generate a random variable that follows a given distribution by making a uniform random variable go through a transform function (inverse CDF).

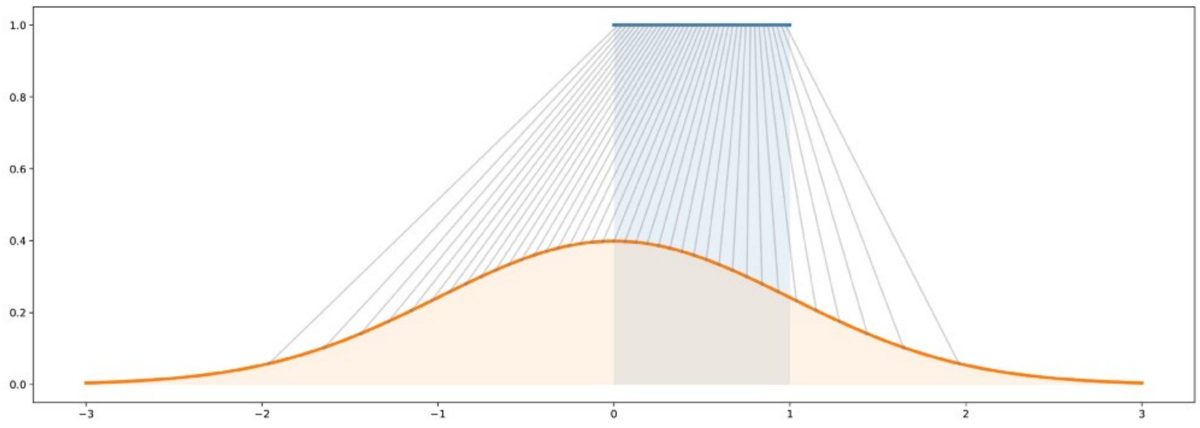

Fig. 2: Illustration of the inverse transform method. Blue: Uniform distribution over $[0, 1]$. Orange: Standard gaussian distribution. Grey: The mapping from the uniform to the gaussian distribution (inverse CDF).

2 Implicit generative models

2.1 Example

Let's start with an example: suppose we have $x \times n$-Images, that all repsesent black-and-white images of cats. We can reshape each image to an $N = n \cdot n$ vector such that each image can be represented by a vector. However, we can't just create a random vector, reshape it into a square and get an image of a cat.
This means that we can say that all vectors that represent an image of a cat follow some distribution that we'd like to learn. This means that if we find the distribution of the "cat image vectors", we can create new images of cats.

However, the distribution that generates such vectors is highly complex over a large space, and even if we assume such a distribution exists, we can't express it explicitly.

What we can do on the other hand is take a more simple distribution, e.g. a low-level Gaussian to draw some random numbers, and then use the *inverse transform method* from before with a neural network as the function to compute a "cat image vector" from the random noise.

2.2 Using a neural network

We are given a sample of (unlabeled) points $\{x_1, \ldots, x_n\}$ drawn from X. We wish to learn the model $\hat{X} = G(Z; w)$ where Z is a "simple" distribution (e.g. low-dimensional Gaussian) and G is some flexible nonlinear function (neural net).

Our goal is to generate a new datapoint $\hat{x}$ by sampling from Z, and then applying the neural network G to obtain $\hat{x}$ that "looks" like it's directly drawn from X.

The key challenge is that it's hard to compute the likelihood of the data. This means we need alternative / surrogate objective functions for the training.

3 Generative Adversarial Network (GAN)

3.1 General Idea

A *generative adversarial network* uses discriminative learning (classification) to train a generative model. The illustration below provides an overview of the idea.

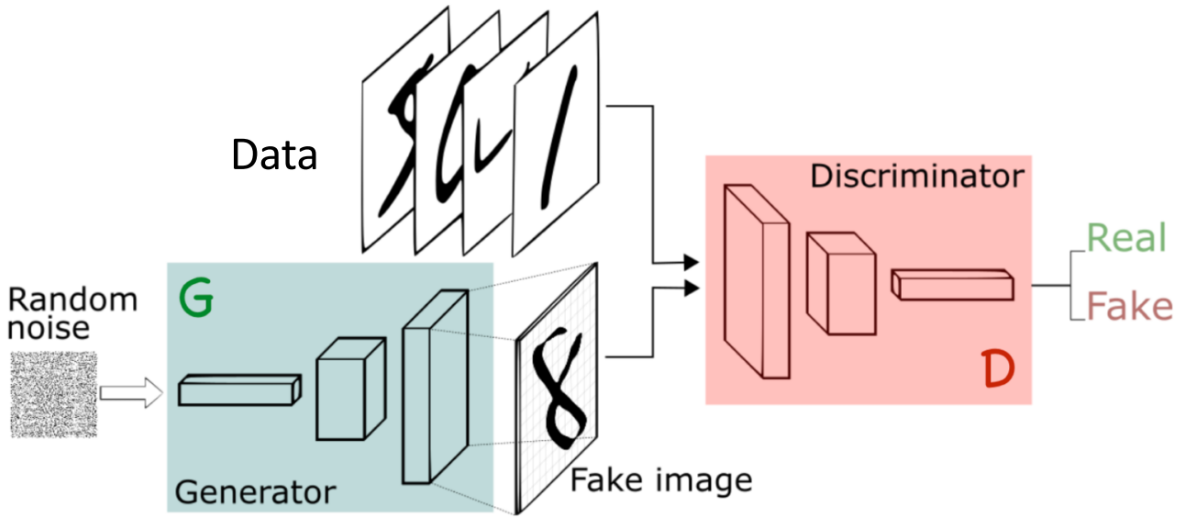

Fig. 3: Illustration of a GAN

Here, we have a discriminator D that gets an image as input, which is either real (from training data) or fake (generated from generator G). Its goal is to predict the Real/Fake label as correctly as possible. On the other side, we have the generator G that gets random noise (e.g. drawn from a Gaussian) as input, and then generates a new, fake image. Its goal is to mimic the "distribution" from which the original training data is from as closely as possible, such that the classifier cannot distringuish a real from a fake image.

Key idea: optimize the parameters w to make samples from model hard to distinguish from data sample.

3.2 GAN from 2 Neural Networks

We can implement a GAN by using two neural networks:

- The generator G tries to produce realistic examples
- The discriminator D tries to detect "fake" examples

This can be viewed as a "game", where:

$$D : \mathbb{R}^d \text{ wants: } D(x) = \begin{cases} \approx 1 \text{ if } x \text{ is real} \\ \approx 0 \text{ if } x \text{ is fake} \end{cases}$$

$$G : \mathbb{R}^m \to \mathbb{R}^d \text{ wants: } D(G(z)) \approx 1 \text{ for samples (noise) } z$$

This can be formulated as:

$$\min_{w_G} \max_{w_D} \underbrace{\underset{X\sim Data}{\mathbb{E}}[\log D(x;w_D)] \; + \; \underset{Z\sim Noise}{\mathbb{E}}[1 - D(G(z;w_G);w_D)]}_{=M(w_G,w_D)}$$

Training this GAN requires finding a *saddle point* rather than a (local) minimum:

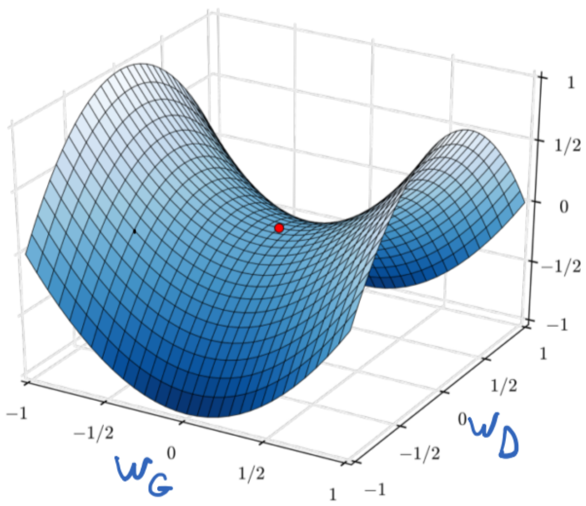

Fig. 4: The red dot mark the saddle point which we want to find as the solution (minimizes w_G and maximizes w_D).

3.3 Convergence

The capacity of a neural network is the ”scope” of the types of mapping functions that it’s able to learn. A neural network with a too small capacity will underfit and a neural network with too large capacity will overfit (memorize the training dataset).

Now, if G and D have ”enough” capacity, then the data generating distribution is a saddle point of

$$\min_{w_G} \max_{w_D} M(w_G, w_D)$$

In practice, we train the model on finite training data. This means we have the danger of memorization, and the discriminator should not be ”too powerful”.

3.4 Simultaneous gradient descent

The common training approach to train a GAN on a set of training data is to simultaneously apply (stochastic) gradient descent to the empirical GAN objective:

$$w_G^{(t+1)} = w_G^{(t)} - \eta_t \nabla_{w_G} M(w_G, w_D^{(t)})$$
$$w_D^{(t+1)} = w_D^{(t)} - \eta_t \nabla_{w_D} M(w_G^{(t)}, w_D)$$

The gradients are approximated by samples of (minibatches of) data points.

As illustrated in the picture below, the (stochastic) gradient descent converges (with correct choice of learning rate etc.) to the saddle point:

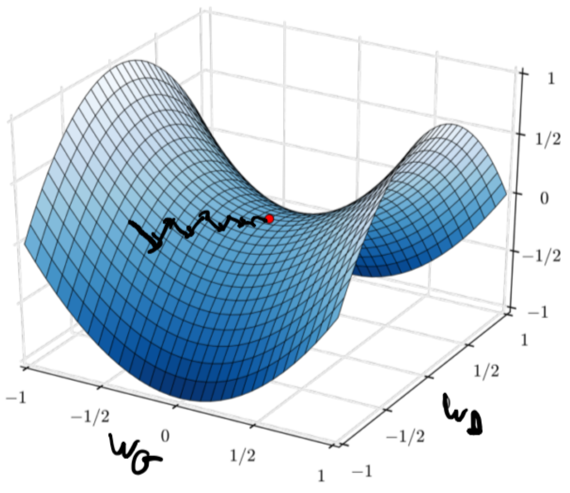

Fig. 5: The gradient descent method approaching the saddle point of the distribution.

3.5 Challenges

As the GANs are a relatively recent idea, a lot of active research is done to solve some of the problems. Some of the problems are listed in the following:

Oscillations / divergence: As with all neural networks, a GAN might oscillate or even diverge, making it impossible to find a solution. One solution is to use a regularizer (gradient penalties). As this is still a recent research topic, the lecture didn't elaborate on that topic.

Mode collapse: Another challenge is that, due to the discriminator, the GAN might "collapse" on a few modes of the data, this means the GAN can only represent a subset of the original data.

4 Sources

- Introduction to Machine Learning Class at ETH (Prof A. Krause) (https://las.inf.ethz.ch/teaching/introml-s19)
- Understanding Generative Adversarial Networks (GANs) (J. Rocca) (https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29)

4.1 Images

1. Plotted with https://www.essycode.com/distribution-viewer/
2. https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29
3. Introduction to Machine Learning, Lecture on the 22.05.2019, Page 14
4. Introduction to Machine Learning, Lecture on the 22.05.2019, Page 20
5. Introduction to Machine Learning, Lecture on the 22.05.2019, Page 24