GAN

Soham Patel

2024-08-06

GAN IN VERY DETAIL.

Stepping in the field of GEN AI, we came across Variational Encoders. There is an another very important concept called GAN.

GAN also called Generative Adversarial Networks is used in the field of graphical modelling to generate unseen data, it maybe an image, letters or a sound.

How can it do it? To understand that, lets first understand it's architecture.

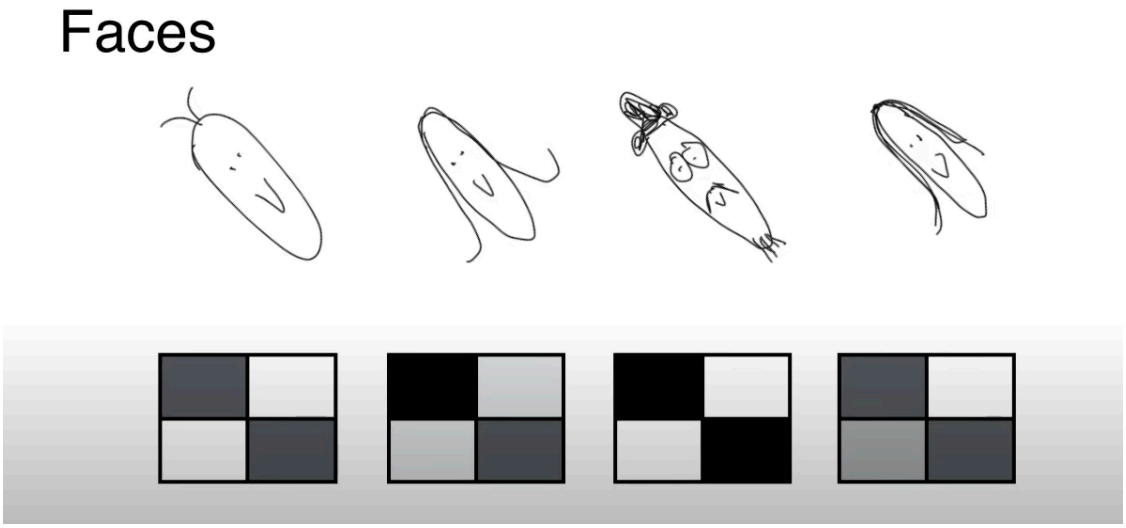
It involves GENERATOR and DISCRIMINATOR architecture where this two modules compete to defeat the other. In the process both learns and gets better. The generator generates the data and the discriminator tries to identify whether it is fake or real.

So, what's the end point? The point when our generator is so good that it could easily fool discriminator all the time, making it think that the generated or fake data is real. Right here we could use our generator to generate real like data and we no more need discriminator. The Generator improves on the feedback of discriminator, we would discuss this in very detail.

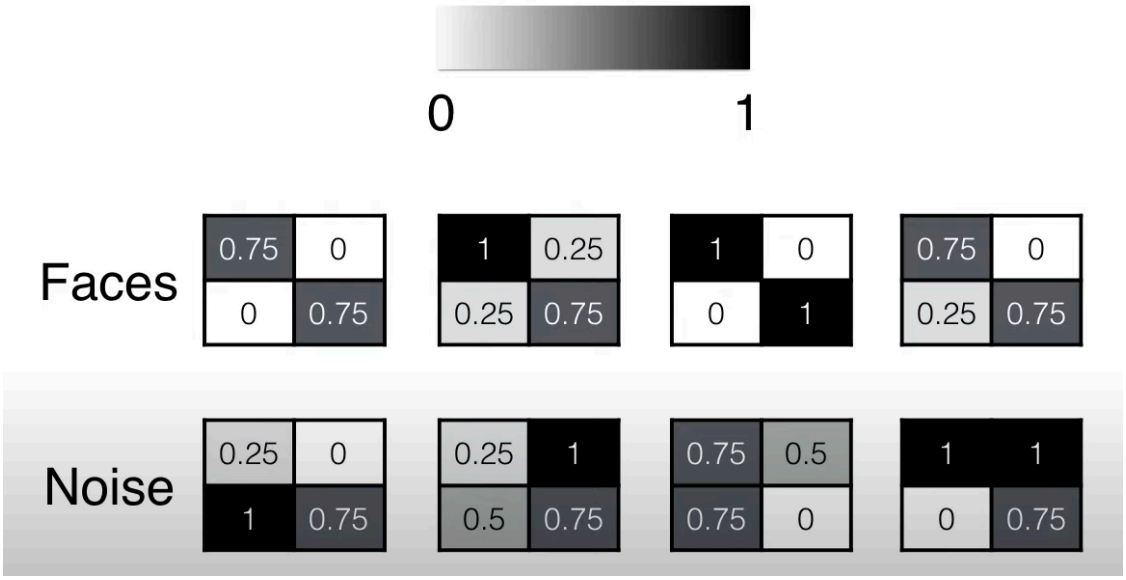
The Generator undergoes unsupervised learning cause it doesn't gets trained on any labelled data but on the other hand the discriminator is taught the labelled data in advance to discriminate noise or fake and the real data. So overall the model uses Unsupervised learning.

Lets understand with an example.

We are introduced to people living in the slanted world where they just have computers with 4 pixels and also just have knowledge of single layer neural network, We can still build a GAN for them to generate real like faces of those people.



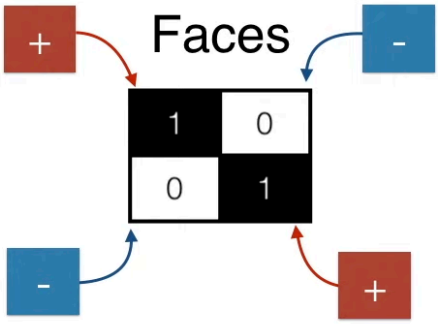
Here are the faces of those people and we converted them to pixels (2x2) cause that's what there computers are able to process. We can feed our discriminator on such inputs to make them well understand what actual faces are. So that they can tell them apart from the noise generated by generator.

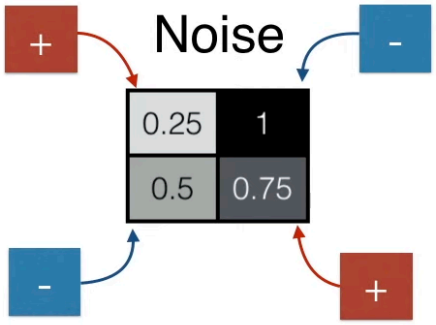


We can see the trend in the faces data. It is generally seen that the first and forth pixel have a very large value and the other two have a comparably small value. That's the patter discriminator can learn to distinguish. The discriminator is trained on both real and fake data. Real is inputted by us, fake data is taken from random generations from the generator.

FOR DISCRIMINATOR

For our example the discriminator's single level neural network in the tilted world may be using a function.



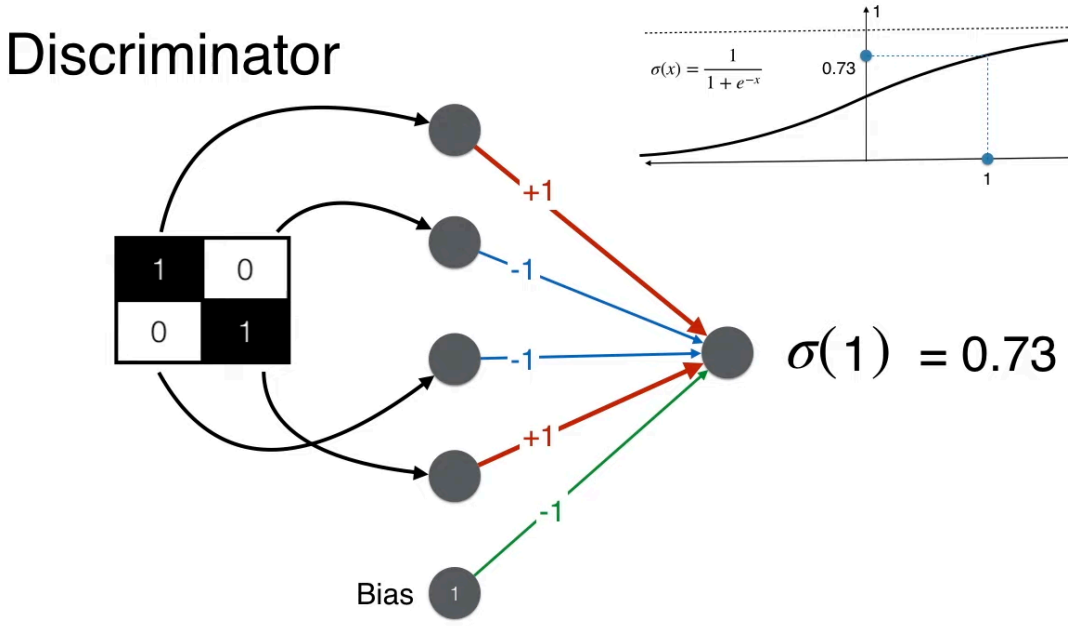


1*1 + 0*(-1) + 0*(-1) + 1*1
= 2

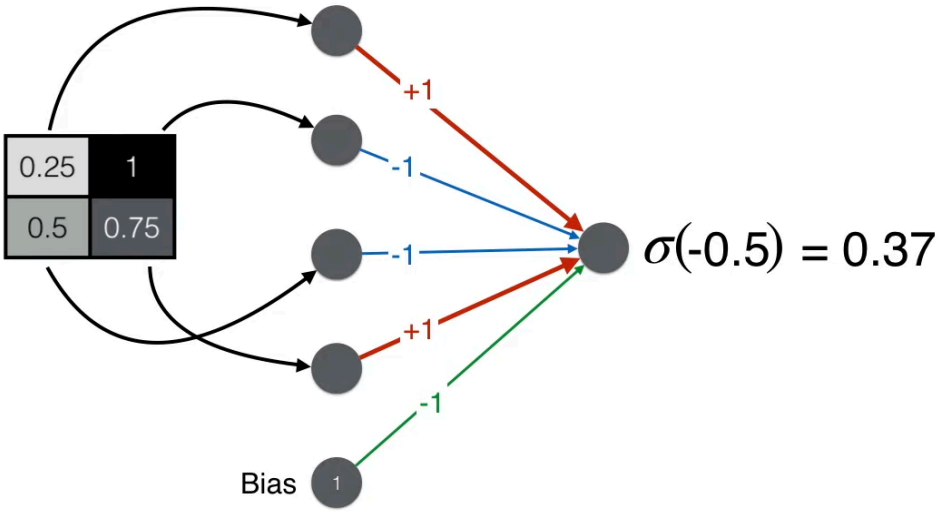
0.25*1 + 1*(-1) + 0.5*(-1) + 0.75*1
= -0.5

Threshold = 1

More than 1: face. Less than 1: no face



Here after applying the sigmoid on the output layer we would be getting 0.73 that is a high probability of getting a face. On other hand lets try for a non face.

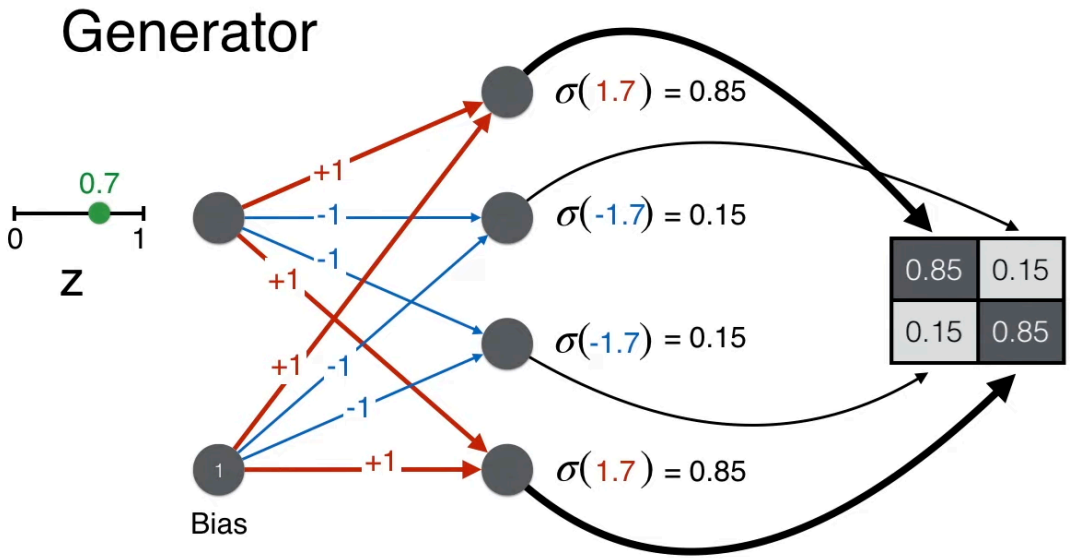


A very low probabilistic score again sets this as not face , so our discriminator is doing well.

Now for GENERATOR.

It starts with some random values to generate some very obvious fake noise that could be sent to discriminator and as discriminator would easily tell fake, generator could keep on improving.

This is one of the end result what we would expect our generator to yield.



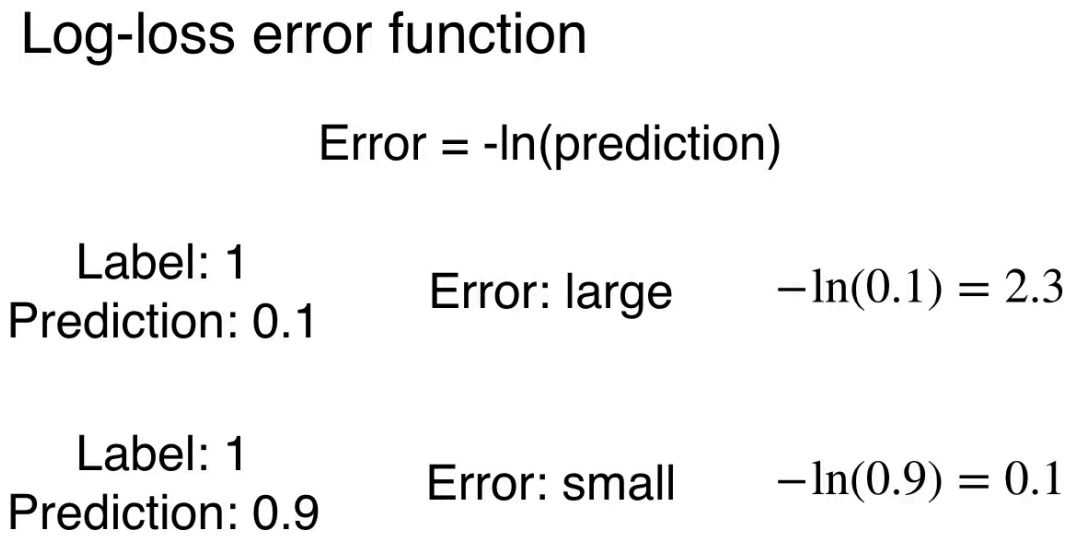
As we could see that it gave high values for to first and last pixel and small for the other two . This input could fool discriminator.

Before training let's talk about the loss functions

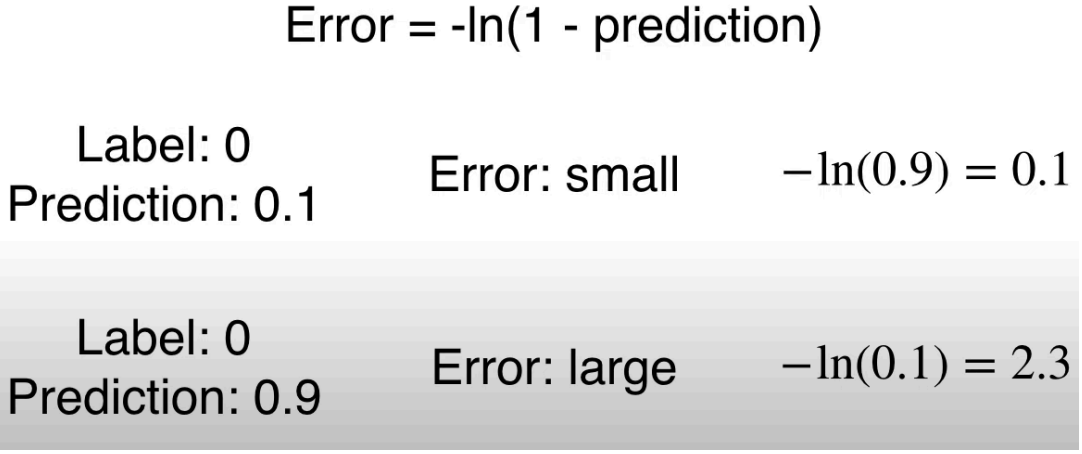
We use Log-loss error functions to train both the modules.

Lets take an example of the generator state we would want the output of the discriminator to be 1 to ensure our generator is performing well and managed to fool the discriminator. So one is our ideal value.

Now as we could see in the below image that predictions closer to 1 are giving less errors and vice versa, so negative natural log is a good error function for generator.

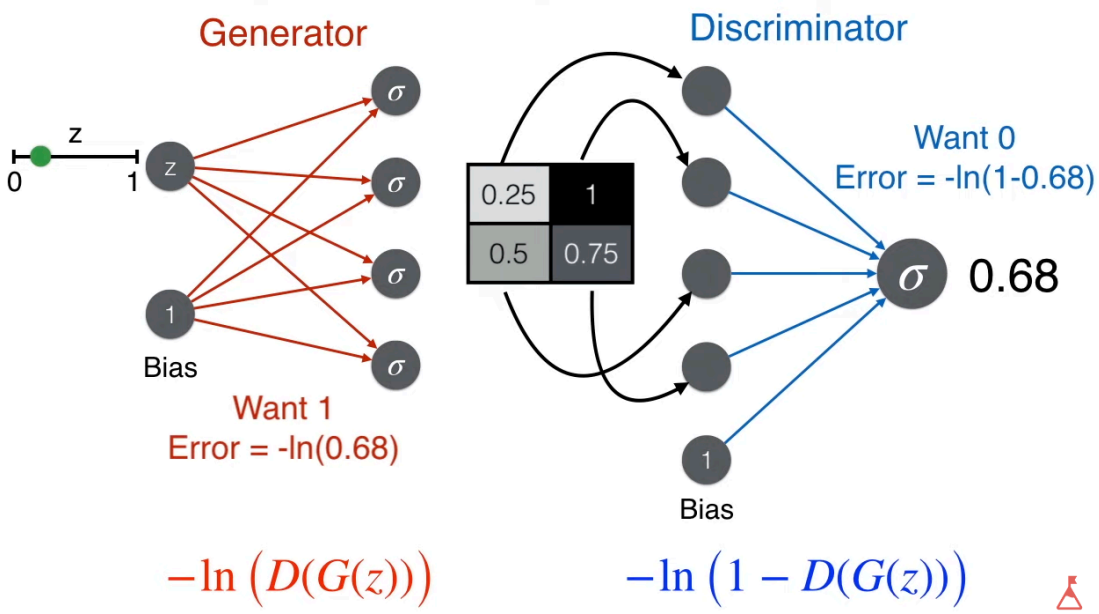


Now when talking about the discriminator we would be wanting it to classify all the images generated by the generator as 0 cause they are not any real images. So values very close to 0 should yield minimum error in case of the discriminator. For this we could use



After this we could start with the training of Generators and Discriminators.

For an image generated by a generator as shown in below diagram the discriminators perform the mathematics as stated above and after gives a score, that is 0.68



Now as we can see that the generator has an upper edge cause the error of the generator that is -ln(D(G(Z))) would be smaller compared to that of discriminator and the different errors for both could help them improve their own weights, hoping to improve the performance in the next iteration.

So by competing both could improve their performance the same process would go on multiple times until we reach a point where our generator manages to fool the discriminator every single time creating ultra real looking images.

“Also I forgot to mention the z in here, it is a point chosen from a normal or gaussian distribution. By sampling z from a simple distribution (like a normal distribution), we ensure that the latent space is well-behaved and easy to sample from. This controlled and structured starting point allows the Generator to systematically explore different variations in the data it creates. Like if we modify and change z way to much, our image may no where be seen anywhere like a face.

So in simple terms the data we feed (lots of real faces) to the discriminator follows a normal distribution when seen in lesser dimension or in a latent space and the z is the point that lies within that distribution so that the generator could easily generate images that look like face as it samples from the distribution of real faces a completely new point that on generating could look very much like the face.”

Example taken from: https://www.youtube.com/watch?v=8L11aMN5KY8&t=241s.