Choose an artificial intelligent system to discuss/analyze from a math/probabilistic point of view.

So in this task, i want to have a discuss from a math point of GANs.

Generative Adversarial Networks refer to a family of generative models that seek to discover the underlying distribution behind a certain data generating process. This distribution is discovered through an adversarial competition between a generator and a discriminator. As we saw in an earlier introductory post on GANs, the two models are trained such that the discriminator strives to distinguish between generated and true examples, while the generator seeks to confuse the discriminator by producing data that are as realistic and compelling as possible.

In other words,
Discriminator: The role is to distinguish between actual and generated (fake) data.
Generator: The role is to create data in such a way that it can fool the discriminator.

The Discriminator

The goal of the discriminator is to correctly label generated images as false and empirical data points as true.
Therefore, we might consider the following to be the loss function of the discriminator:

$$L_D = Error(D(x), 1) + Error(D(G(z)), 0)$$

Here, we are using a very generic, unspecific notation for Error to refer to some function that tells us the distance or the difference between the two functional parameters.
So always use Binary Cross Entropy loss to caluate the error, and will introduce in the next part.

The Generator

We can go ahead and do the same for the generator. The goal of the generator is to confuse the discriminator as much as possible such that it mislabels generated images as being true.

$$L_G = Error(D(G(z)), 1)$$

The key here is to remember that a loss function is something that we wish to minimize. In the case of the generator, it should strive to minimize the difference between 1, the label for true data, and the discriminator's evaluation of the generated fake data.

Binary Cross Entropy

The loss function described in the original paper [2]. can be derived from the formula of binary cross-entropy loss.
A common loss function that is used in binary classification problems is binary cross entropy. As a quick review, let's remind ourselves of what the formula for cross entropy looks like:

$$H(p,q) = E_{x\sim p(x)}[-logq(x)]$$

In classification tasks, the random variable is discrete. Hence, the expectation can be expressed as a summation.

$$H(p,q) = -\sum_{x\in\chi} p(x)logq(x)$$

We can simplify this expression even further in the case of binary cross entropy, since there are only two labels: zero and one.

$$H(y,\widehat{y}) = -\sum ylog(\widehat{y}) + (1-y)log(1-\widehat{y})$$

This is the Error function that we have been loosely using in the sections above.

Binary cross entropy fulfills our objective in that it measures how different two distributions are in the context of binary classification of determining whether an input data point is true or false.
Applying this to the loss functions in the first formulae,

$$L_D = -\sum_{x\in\chi, z\in\zeta} log(D(x)) + log(1-D(G(z)))$$

And do the same for the second formulae,

$$L_G = -\sum_{z\in\zeta} log(D(G(z))$$

Now we have two loss functions with which to train the generator and the discriminator.
Note that, for the loss function of the generator, the loss is small if $D(G(z))$ is close to 1, since log(1)=0.
This is exactly the sort of behavior we want from a loss function for the generator.

Model Optimization

Now that we have defined the loss functions for the generator and the discriminator.
it's time to discuss some math to solve the optimization problem, i.e. finding the parameters for the generator and the discriminator such that the loss functions are optimized.

Training the Discriminator

When training a GAN, we typically train one model at a time. In other words, when training the discriminator, the generator is assumed as fixed.

Let's return back to the min-max game. The quantity of interest can be defined as a function of G and D.
Let's call this the value function:

$$V(G,D) = \mathrm{E}_{x\sim p_{data}}[log(D(x))] + \mathrm{E}_{z\sim p_z}[log(1-D(G(z)))]$$

In reality, we are more interested in the distribution modeled by the generator than p_z. Therefore, let's create a new variable, $y = G(z)$, and use this substitution to rewrite the value function:

$$\begin{aligned} V(G,D) &= \mathrm{E}_{x\sim p_{data}}[log(D(x))] + \mathrm{E}_{y\sim p_g}[log(1-D(y))] \\ &= \int_{x\in\chi} p_{data}(x)log(D(x)) + p_g(x)log(1-D(x))dx \end{aligned}$$

The goal of the discriminator is to maximize this value function.
Through a partial derivative of $V(G,D)$ with respect to $D(x)$, we see that the optimal discriminator, denoted as $D*(x)$, occurs when

$$\frac{p_{data}(x)}{D(x)} - \frac{p_g(x)}{1-D(x)} = 0$$

rearranging, and get

$$D*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$$

And this is the condition for the optimal discriminator.
Note that the formula makes intuitive sense: if some sample x is highly genuine, we would expect $p_{data}(x)$ to be close to one and $p_g(x)$ to be converge to zero, in which case the optimal discriminator would assign 1 to that sample.
On the other hand, for a generated sample x=G(z), we expect the optimal discriminator to assign a label of zero, since $p_{data}(G(z))$ should be close to zero.

Training the Generator

To train the generator, i assume the discriminator to be fixed and proceed with the analysis of the value function. Let's first plug in the result i found above, namely D^*, into the value function to see what turns out.

$$\begin{aligned} V(G, D^*) &= \mathrm{E}_{x \sim p_{data}}[log(D^*(x))] + \mathrm{E}_{x \sim p_g}[log(1 - D^*(x))] \\ &= \mathrm{E}_{x \sim p_{data}}[\log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}] + \mathrm{E}_{x \sim p_g}[\log \frac{p_g(x)}{p_{data}(x) + p_g(x)}] \end{aligned}$$

and then,

$$\begin{aligned} V(G, D^*) &= \mathrm{E}_{x \sim p_{data}}[\log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}] + \mathrm{E}_{x \sim p_g}[\log \frac{p_g(x)}{p_{data}(x) + p_g(x)}] \\ &= -log4 + \mathrm{E}_{x \sim p_{data}}[\log p_{data}(x) - \log \frac{p_{data}(x) + p_g(x)}{2}] \\ &+ \mathrm{E}_{x \sim p_g}[\log p_g(x) - \log \frac{p_{data}(x) + p_g(x)}{2}] \end{aligned}$$

For this, what we want to do is that we are exploiting the properties of logarithms to pull out a −log4 that previously did not exist. In pulling out this number, we inevitably apply changes to the terms in the expectation, specifically by dividing the denominator by two.

And then, we can now interpret the expectations as Kullback-Leibler divergence:

$$V(G, D^*) = -\log 4 + D_{KL}(p_{data} || \frac{p_{data} + p_g}{2}) + D_{KL}(p_g || \frac{p_{data} + p_g}{2})$$

And it is here that we reencounter the Jensen-Shannon divergence, which is defined as:

$$J(P, Q) = \frac{1}{2}(D(P||R) + D(Q||R))$$

where $R = \frac{1}{2}(P + Q)$.
This means that the expression in $V(G, D^*)$ can be expressed as a JS divergence:

$$V(G, D^*) = -\log 4 + 2 \cdot D_{JS}(p_{data} || p_g)$$

The conclusion of this analysis is simple:
the goal of training the generator, which is to minimize the value function V(G,D), we want the JS divergence between the distribution of the data and the distribution of generated examples to be as small as possible.

This conclusion certainly aligns with our intuition:
we want the generator to be able to learn the underlying distribution of the data from sampled training examples. In other words, p_g and p_{data} should be as close to each other as possible. The optimal generator G is thus one that which is able to mimic p_{data} to model a compelling model distribution p_g.

About Jensen–Shannon divergence

In probability theory and statistics, the Jensen–Shannon divergence is a method of measuring the similarity between two probability distributions.
It is also known as information radius (IRad) or total divergence to the average.It is based on the Kullback–Leibler divergence, with some notable (and useful) differences, including that it is symmetric and it always has a finite value. The square root of the Jensen–Shannon divergence is a metric often referred to as Jensen-Shannon distance.

Reference

1. Wang Y. A Mathematical Introduction to Generative Adversarial Nets (GAN)[J]. arXiv preprint arXiv:2009.00169, 2020.
2. Goodfellow I J, Pouget-Abadie J, Mirza M, et al. Generative adversarial networks[J]. arXiv preprint arXiv:1406.2661, 2014.
3. Goodfellow I. Nips 2016 tutorial: Generative adversarial networks[J]. arXiv preprint arXiv:1701.00160, 2016.
4. The Math Behind GANs
5. Jensen–Shannon divergence