GANs notes

In classical GANs, the generator function is a neural network that takes a random input, called "latent variable" or "latent code" (denoted as $|z\rangle$ in quantum case) and generates new data samples that are similar to the real data. In the quantum case, the generator function is defined as a variational quantum circuit, which is a type of quantum circuit that is parametrized by a set of parameters, denoted as θ_G.

The generator function takes as input two things: a label $|\lambda\rangle$ and an additional state (random noise) $|z\rangle$. The label $|\lambda\rangle$ can be thought of as a type of "metadata" that tells the generator what type of data it should generate. The additional state $|z\rangle$ can be thought of as a random input, similar to the "latent variable" in classical GANs.

The generator function produces a quantum state, denoted as $G(\theta_G, |\lambda, z\rangle)$, which is a density matrix represented by $\rho_\lambda^G(\theta_G, z)$. This density matrix contains information about the generated data, and is output on a register containing n subsystems. The subsystems are similar to the real data, in that they contain information about the generated data.

The extra input state $|z\rangle$ serves two main purposes:

1. **It provides a source of unstructured noise which adds entropy to the distribution of generated data. For example, the generator could be a unitary circuit that produces a fixed state $\rho_\lambda^G(\vec{\theta_G}, z) = |\psi_\lambda(z)\rangle \langle\psi_\lambda(z)|$ for each λ and $|z\rangle$. By allowing the input $|z\rangle$ to randomly fluctuate, we can create more than one output state for each label.**
   The input state $|z\rangle$ is used to add noise or randomness to the generator circuit G. The generator is a variational quantum circuit that takes in two inputs: a label $|\lambda\rangle$ and the additional state $|z\rangle$. The label $|\lambda\rangle$ is used to specify the class or type of data that the generator should produce, while the input state $|z\rangle$ is used to add randomness to the generator's output. The generator circuit G is designed to produce a quantum state, represented by the density matrix $\rho_\lambda^G(\theta_G, z)$, that is similar to the real data. However, if the generator circuit is unitary and produces a fixed state for each λ and $|z\rangle$, it will not be able to produce different outputs for the same label. By allowing the input $|z\rangle$ to randomly fluctuate, the generator circuit can produce more than one output state for each label, thus adding entropy to the distribution of generated data. This way, the generator can produce a range of different outputs for the same label, which makes the generated data more realistic and diverse.

2. **The variable $|z\rangle$ can act as a control for the generator. By tuning $|z\rangle$, we can transform the output state prepared by the generator, varying properties of the generated data which are not captured by the labels λ. This allows the generator to learn to encode the most important intra-label factors of variation with $|z\rangle$.**
   The variable $|z\rangle$ can act as a control for the generator, meaning it allows for more fine-grained control over the generated data. By varying the value of $|z\rangle$, the generator can produce output states that differ in properties that are not captured by the labels λ. This allows the generator to learn to encode the most important factors of variation within each label. This allows for more flexibility and control over the generated data, and enables the generator to produce a more diverse set of output states.

The first role that the variable $|z\rangle$ plays is to add entropy to the distribution of generated data. This could have been achieved by coupling the generator to a bath, which is a system that is at a constant temperature, but is not under control. However, the second role of $|z\rangle$ is to act as a control for the generator, allowing it to vary properties of the generated data that are not captured by the labels λ. This requires $|z\rangle$ to be under control, even if no explicit structure is given to it during training. This allows for more fine-grained control over the output of the generator, allowing it to better capture the important factors of variation within the data.

KL divergence

KL divergence, also known as Kullback-Leibler divergence, is a measure of the difference between two probability distributions. It is a non-symmetric measure and is used to compare the similarity of two distributions.

Let's consider two probability distributions, P and Q, over the same sample space. For the discrete probability distribution P and Q, the KL divergence between P and Q is defined as:

$$D_{KL}(P||Q) = \sum_i P(i) log(\frac{P(i)}{Q(i)})$$

where i is the sample space. It measures the amount of information lost when approximating P with Q.

Properties:

- always non-negative, i.e., $D_{KL}(P||Q)$ or $D_{KL}(Q||P) \geq 0$. This can be seen from the fact that the logarithm is always non-negative, and P(i) is always greater than or equal to 0. Additionally, KL divergence is only equal to 0 when P and Q are equal, i.e. P(i) = Q(i) for all i.
- $D_{KL}(P||Q) \neq D_{KL}(Q||P)$ (Not Symmetric)

When the distributions being compared are multivariate normal distributions, the KL divergence can be calculated using the following formula:

$$D_{KL}(N(\mu_1, \Sigma_1)||N(\mu_2, \Sigma_2)) = \frac{1}{2}[\text{tr}(\Sigma_2^{-1}\Sigma_1)+(\mu_2-\mu_1)^T\Sigma_2^{-1}(\mu_2-\mu_1)-k+log(\frac{|\Sigma_2|}{|\Sigma_1|})]$$

where $N(\mu, \Sigma)$ is a multivariate normal distribution with mean μ and covariance matrix Σ, Tr denotes the trace of a matrix, and k is the dimension of the distributions.

The KL divergence for multivariate normal distributions can be used to calculate the dissimilarity between two multivariate normal distributions. The KL divergence for multivariate normal distributions has some useful properties, such as being non-negative and non-symmetric, and it equals zero if and only if the two distributions are identical.

It is important to note that KL divergence can be used to measure the dissimilarity only when the distributions are in the same family, In this case, both distributions are multivariate normal distributions, so KL divergence can be used to measure the dissimilarity.

KL divergence has several important applications in machine learning and information theory. One of the most common uses is in the field of information

compression, where it is used to measure the efficiency of data compression algorithms. It is also used in reinforcement learning, where it is used to measure the difference between the current policy and the optimal policy. In addition, KL divergence is widely used in generative models such as Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) to measure the difference between the model distribution and the true data distribution.

In conclusion, KL divergence is a useful tool for measuring the difference between two probability distributions. Its non-negativity and its ability to measure the information lost when approximating one distribution with another make it a valuable tool in various fields such as information compression, reinforcement learning and generative models.

Generative model with KL-Divergence

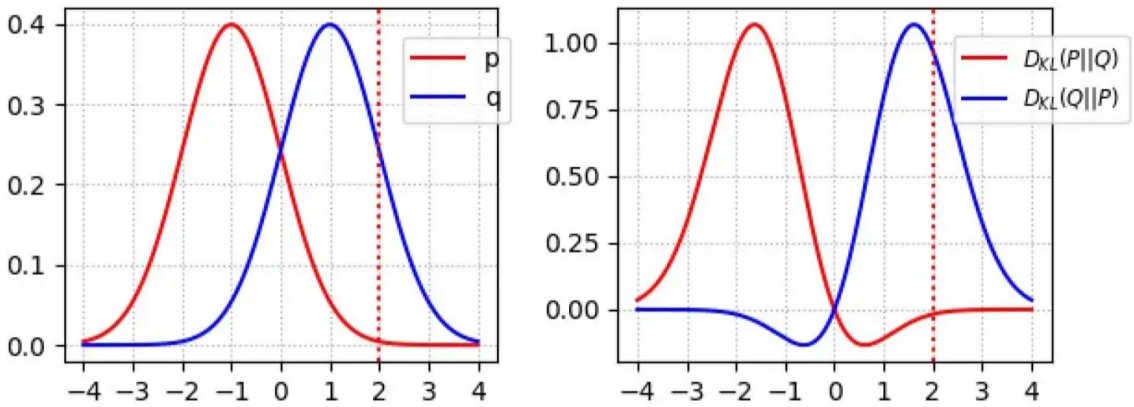
Before GAN, many generative models create a model θ that maximizes the Maximum Likelihood Estimation MLE. i.e. finding the best model parameters that fit the training data the most.

$$\hat{\theta} = arg\max_{\theta} \Pi_{i=1}^{N} p(x_i|\theta)$$

This is the same as minimizing the KL-divergence which measures how the probability distribution q (estimated distribution) diverges from the expected probability distribution p (the real-life distribution).

$$D_{KL}(p||q) = \int p(x) log(\frac{p(x)}{q(x)}) dx$$

$KL(x)$ drops to 0 for area where $p(x) \to 0$. For example, in the figure on the right below, the red curve corresponds to D(p, q). It drops to zero when $x > 2$ where p approaches 0.



Note: KL(p,q) is the integral of the red curve in the right

What is the implication? The KL-divergence $DL(p, q)$ penalizes the generator if it misses some modes of images: the penalty is high where $p(x) > 0$ but

$q(x) \to 0$. Nevertheless, it is acceptable that some images do not look real. The penalty is low when $p(x) \to 0$ but $q(x) > 0$. (Poorer quality but more diverse samples)

On the other hand, the reverse KL-divergence $DL(q, p)$ penalizes the generator if the images do not look real: high penalty if $p(x) \to 0$ but $q(x) > 0$. But it explores less variety: low penalty if $q(x) \to 0$ but $p(x) \to 0$. (Better quality but less diverse samples)

JS divergence

The Jenson-Shannon divergence, also known as the JS divergence, is a measure of similarity between two probability distributions. It is a symmetric version of the Kullback-Leibler divergence and is defined as the average of the Kullback-Leibler divergences between the two distributions and the distribution that is their average.

Mathematically, the JS divergence between two probability distributions P and Q is defined as:

$$JS(P||Q) = \frac{(D_{KL}(P||M) + D_{KL}(Q||M)}{2}$$

where $M = (P + Q)/2$ and $D_{KL}(P||Q)$ is the Kullback-Leibler divergence between P and Q.

One of the key properties of the JS divergence is that it is always non-negative, with a value of 0 if and only if P = Q. It also satisfies the triangle inequality, meaning that $JS(P||Q) + JS(Q||R) \geq JS(P||R)$ for any probability distributions P, Q, and R.

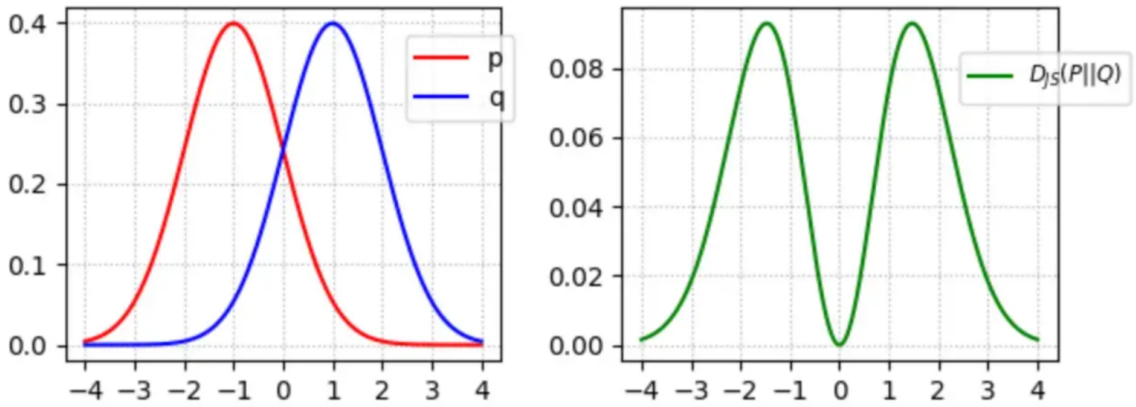
The basic idea behind the JS divergence is to provide a measure of similarity between two probability distributions that is symmetric and easy to compute. It can be used in a variety of applications, such as clustering, classification, and feature selection. In machine learning, it is often used as a loss function in generative models such as GANs.

In addition to its mathematical definition, it is also important to understand the underlying intuition behind JS divergence. It is a measure of how much information is lost when approximating one distribution with the other. The JS divergence between two distributions is zero if and only if the two distributions are identical, and it increases as the two distributions diverge.

In summary, JS divergence is a measure of similarity between two probability distributions, it is symmetric, non-negative, satisfies the triangle inequality and it is often used as a loss function in generative models such as GANs.

Generative Model with JS-Divergence

$$D_{JS}(p||q) = \frac{(D_{KL}(p||\frac{p+q}{2}) + D_{KL}(q||\frac{p+q}{2})}{2}$$



JS-divergence is symmetrical. Unlike KL-divergence, it will penalize poor images badly. (when $p(x) \to 0$ and $q(x) > 0$) In GAN, if the discriminator is optimal (performing well in distinguishing images), the generator's objective function becomes:

$$\min_G V(D*, G) = 2D_{JS}(p_r||p_g) - 2\log 2$$

So optimizing the generator model is treated as optimizing the JS-divergence. In experiments, GAN produces nicer pictures comparing to other generative models that use KL-divergence.

Issues of JS-divergence: Vanishing gradients in JS-Divergence
What happens to the JS-divergence gradient when the data distribution q of the generator's images does not match with the ground truth p for the real images. Let's consider an example in which p and q are Gaussian distributed and the mean of p is zero. Let's consider q with different means to study the gradient of $JS(p, q)$.

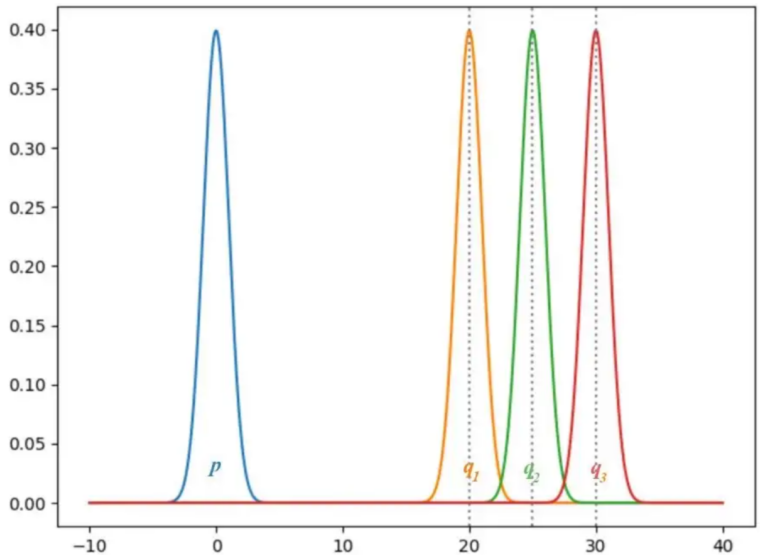
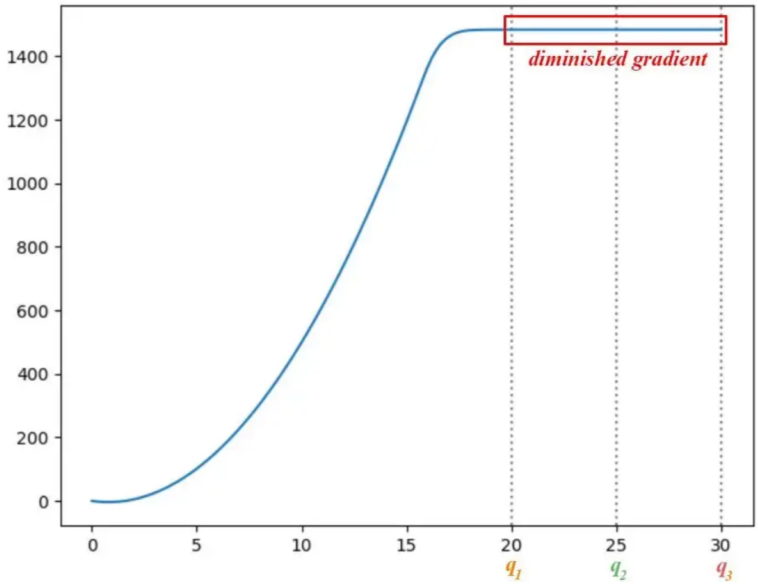


fig: plot of JS(p,q) between p and q with means of q ranging from 0 to 30



As shown above, the gradient for the JS-divergence vanishes from q1 to q3. The GAN generator will learn extremely slow to nothing when the cost is saturated in those regions. In particular, in early training, p and q are very different and the generator learns very slow.

Unstable Gradients

Because of the vanishing gradient, an alternative cost function is proposed by the original GAN paper:
$\nabla_{\theta_g} \log(1 - D(G(z^i))) \rightarrow 0$ change to $\nabla_{\theta_g} - \log(D(G(z^i)))$
The corresponding gradient according to a different research paper from Arjovsky is:

$$E_{z\sim p(z)}[-\nabla_\theta \log(D^*(g_\theta(z))|_{\theta=\theta_0})] = \nabla_\theta[KL(P_{g\theta}||P_r) - 2JSD(P_{g\theta}||P_r)]|_{\theta=\theta_0}$$

It includes a reverse KL-divergence term which Arjovsky uses it to explain why GAN has higher quality but less diverse image comparing to generative models based on KL-divergence. But the same analysis claims that the gradients fluctuate and cause instability to the model. To illustrate the point, Arjovsky freezes the generator and trains the discriminator continuously. The gradient for the generator starts increasing with larger variants.