Lecture 6 Generative Adversarial Networks

6.1.1 Basic Idea of Generative Adversarial Networks (GAN)

Generative Adversarial Networks (GAN) is one of the most promising recent developments in Deep Learning. The goal of generative modeling is to autonomously identify patterns in input data, enabling the model to produce new examples that feasibly resemble the original dataset. GANs tackle this challenge through a unique setup, involving two key components: Generator (G), which learns to produce new instances, and Discriminator (D), which is tasked with distinguishing between genuine and generated instances.

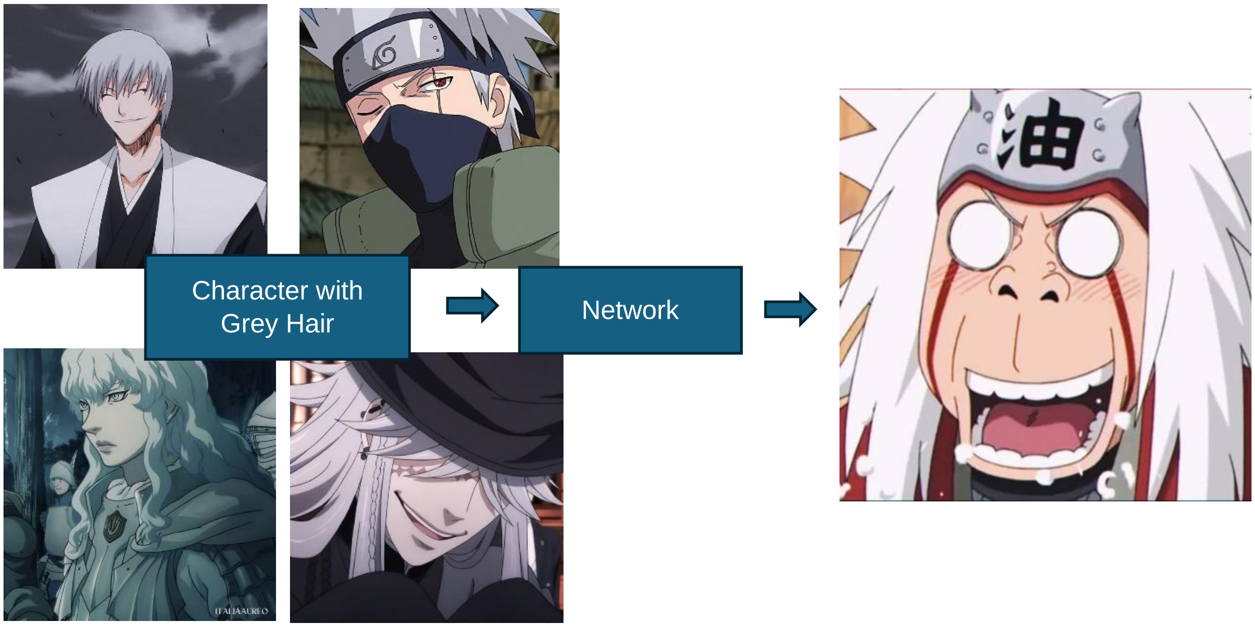
- Generator (G). Model that is used to generate new plausible examples from the problem domain.
- Discriminator (D). Model that is used to classify examples as real (from the domain) or fake (generated).

GAN is almost always motivated by the case of a counterfeiter (Generative) and the policeman (Discriminator). Initially, the counterfeiter will show the policeman a fake bill. The policeman says it is fake. The policeman gives feedback to the counterfeiter to explain why the money is fake. The counterfeiter attempts to make a new fake bill based on the feedback it received. The policeman says the money is still fake and offer a new set of feedback. The counterfeiter attempts to make a new fake bill based on the latest feedback. The cycle continues indefinitely until the policeman is fooled by the fake money because it looks real. In this sense, a well-performing GAN model should be able to generate new examples that are not just plausible, but indistinguishable from real examples from the problem domain.

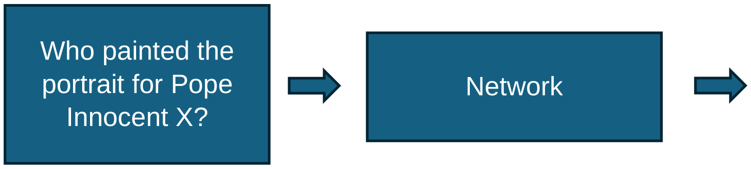
GAN is widely used in applications developed for image generation and sentence generation. GAN is especially useful for applications that need Conditional Generation. For instance, you could input certain conditions and ask the machine to generate images or videos (e.g. SORA). You could also input one image and ask the machine to generate another image.

GAN is especially good at tasks that need "creativity."

<u>Drawing</u>



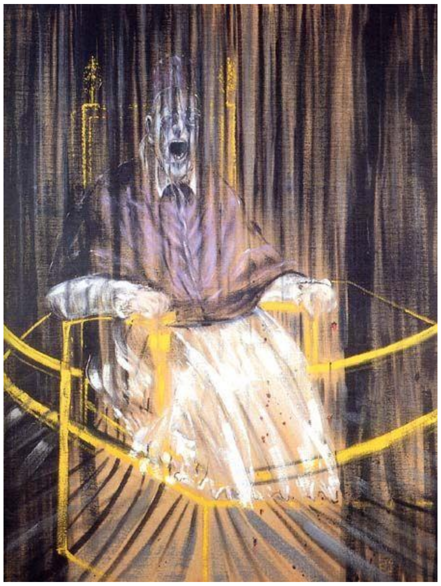
Chatbot



Who painted the portrait for Pope Innocent X?

Network

(Diego Velázquez, 1650)



(Francis Bacon, 1953)

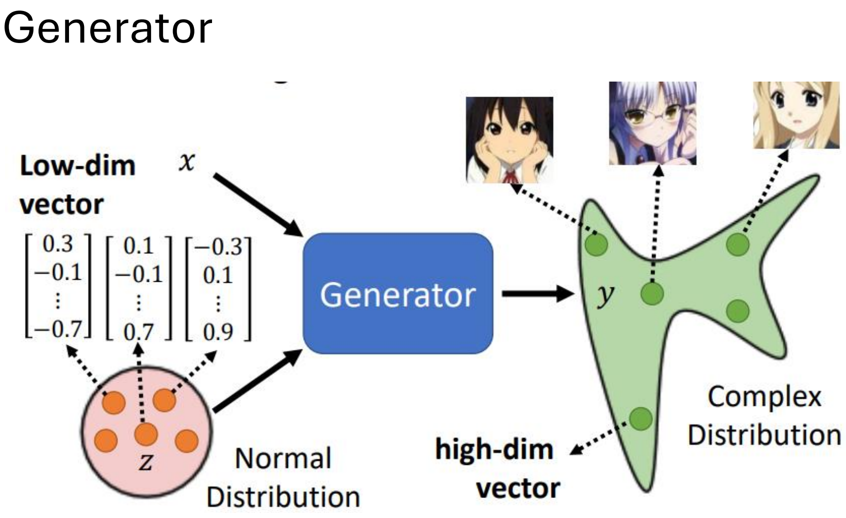
Video (Diffusion Model)

https://www.wsj.com/tech/personal-tech/openai-cto-sora-generative-video-interview-b66320bb?mod=hp_lead_pos9

The primary goal of training a GAN is to identify parameters of Generator G. The generator is essentially a Neural Network (NN). In the meantime, what makes GAN special is that when you train the Generator G, You need to train a Discriminator (D). Discriminator (D) is also a Neural Network (NN).

- The input of GAN could be a sentence or a picture.
- The output of GAN is a scalar. The scalar represents the quality of the output. The higher the scalar, the better the output quality.

- The relationship between Generator G and Discriminator D is like the predator and the prey. Discriminator D forces the Generator G to improve and to evolve. This is essentially where the term “adversarial” comes from.



Let y be the output vector from Generator G that corresponds to different manga characters. There are two types of data used in GAN: input and noise.

- x is the input vector, which has a size of $M \times 1$.
- z is the noise to be added to the input vector.

We use input in combination with noise to obtain the generated sample. If z is $M \times 1$, we can obtain the generated sample by

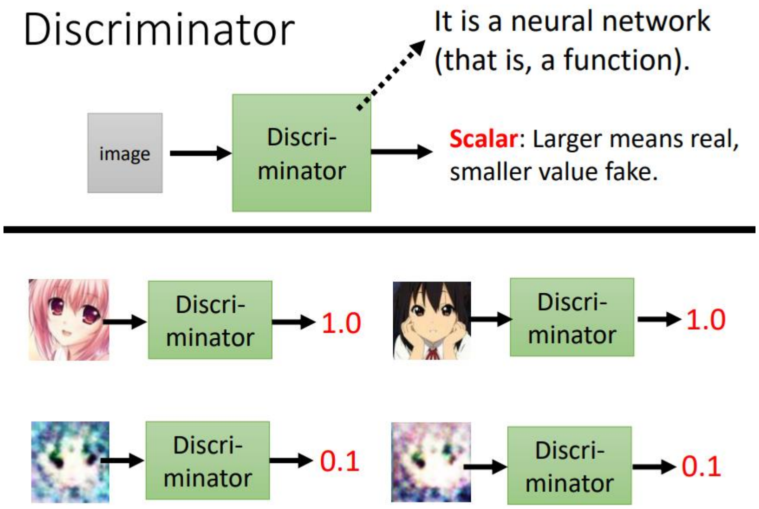
$$\tilde{x} = x + z$$

As z is often configured to be a low dimensional vector, if z is $K \times 1$, we can obtain the generated sample by dot product:

$$\tilde{x} = x \cdot z$$

The role of the generator is to G produce a high-dimensional vector that corresponds to manga characters. For instance, the output from the hidden state is 27×1, which could be further organized into an image of size $3 \times 3 \times 3$.

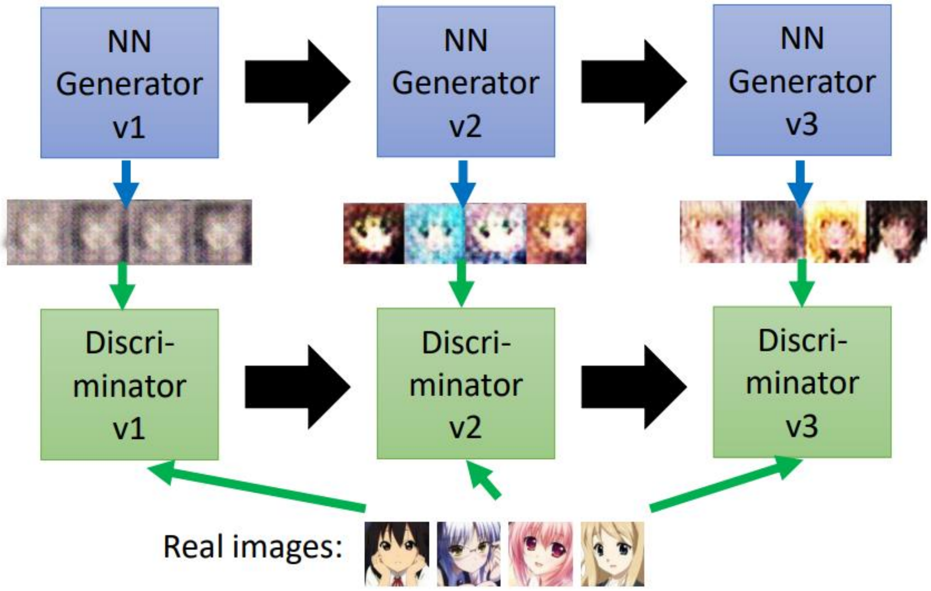
- When you change the input vector, the output vector changes correspondingly. The vector that starts with 0.3 corresponds to brown hair. The vector that starts with 0.1 corresponds to purple hair. The vector that starts with -0.3 corresponds to blonde hair.
- The choice of using different distributions for z is found to have limited impacts on the model outcome.



The discriminator D takes an image or sentence as input and produces a scalar as the output. The larger the scalar value, the more likely the input image is real. The less the scalar value, the more likely the input image is fake.

Both the Generator G and the Discriminator D are neural networks. The specification of these neural networks is all at user's choice. You can use CNN. You can also use Transformer. As long as the two components obtain the desired output, which says, the Generator G produces high dimensional vector that takes the same form as the input, and Discriminator D produces a scalar, the choice is acceptable.

In the first Generator G_{v1}, the parameter values are randomly chosen (initialization). The output from Generator G_{v1} are blurred images. Then the first Discriminator D_{v1} learns to differentiate the fake images from the real ones. The second Generator G_{v2} then update its parameters from the learning, generates a second collection of images, in which, eyes are forming in shape. The purpose of training the second Generator G_{v2} is to fool the Discriminator. If the first Discriminator D_{v1} uses eyes alone to determine whether an image is real or fake, the training of the second Generator G_{v2} can well fool the first Discriminator D_{v1}. However, Discriminator evolves. The first Discriminator D_{v1} then evolves into the second Discriminator D_{v2}. The second Discriminator is then trained to tell the difference between the real images and the generated images with eyes alone. The discriminator then may find the generated images have no mouth. Finally, the third Generator G_{v3} is trained to fool the second Discriminator D_{v2}. If the second Discriminator D_{v2} uses mouth to determine whether an image is real or fake, the third Generator G_{v3} will add mouth to all images it generates. As the Discriminator requires more and more conditions to pass its screening, the Generator evolves to improve the images it generates.

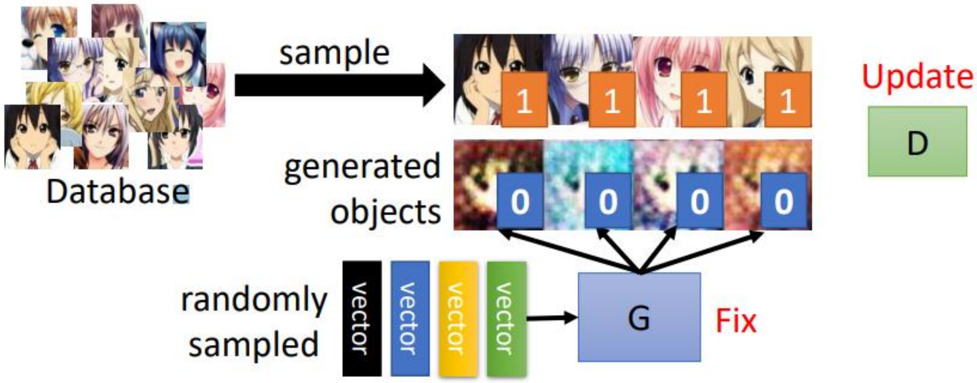


6.1.2 Architecture of Generative Adversarial Networks (GAN)

Initialize Generator (G) and Discriminator (D)

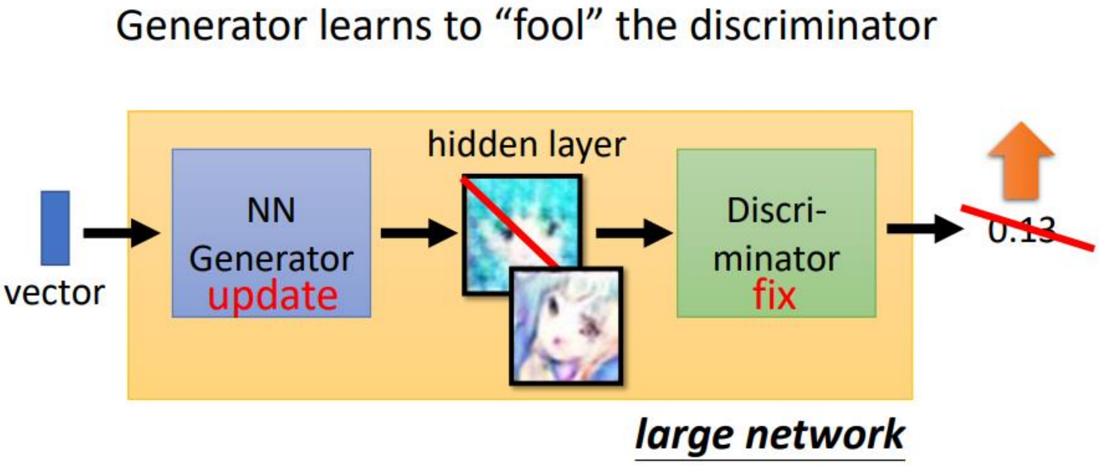
In each training iteration:

Step 1: Fix Generator (G) while updating Discriminator (D). The discriminator learns to assign high scores to real objects and low scores to generated objects. It can take the form of either classifier or regression. For example, when feeding Discriminator (D) with real images, the output from Discriminator (D) is large (close to 1). When feeding Discriminator (D) with fake pictures, the output from Discriminator (D) is small (close to 0). Using this rule as the criteria to train (update) Discriminator (D). The process is pretty much the same as training the fully connected neural network to perform regression, or to make classification. Discriminator (D) is trained to differentiate the generated images from the real ones.



Step 2: Fix Discriminator D, while updating Generator G. Generator (G) learns to “fool” Discriminator (D) – the images generated by Generator (G) should progressively receive higher scores in Discriminator (D). In particular, the Generator takes an input vector to generate an image, then the image is fed into the Discriminator and receives a score (scalar), 0.13. The goal in the training step of Generator is to maximize

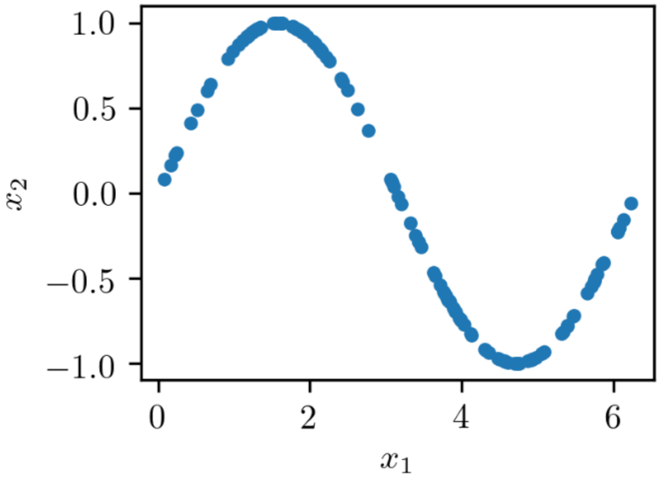
the output score from the Discriminator. Once the Generator updates its parameters toward the optimum and generates a new image, the new image should receive a higher score in the Discriminator.



For instance, in the above example, the Generator uses five layers of neural networks. The Discriminator uses five layers of neural networks too. We hereby connect the Generator and the Discriminator to form a large neural network of 10 layers. Suppose that each hidden layer produced by the Generator has the same size as the input vector, say, 48×1, which could be further organized into an image of size $4 \times 4 \times 3$. The image is then fed into the Discriminator to receive a score. Discriminator learns the difference between the fake and the real via parameter training and then the parameter values are fixed. Throughout the training of Generator, the last few layers of the neural network (Discriminator) are fixed. Only parameters of the first few layers (Generator) are trained and updated. Each iteration of training targets a particular feature to be included in the generated images.

6.1.3 Generative Adversarial Networks (GAN) Algorithm

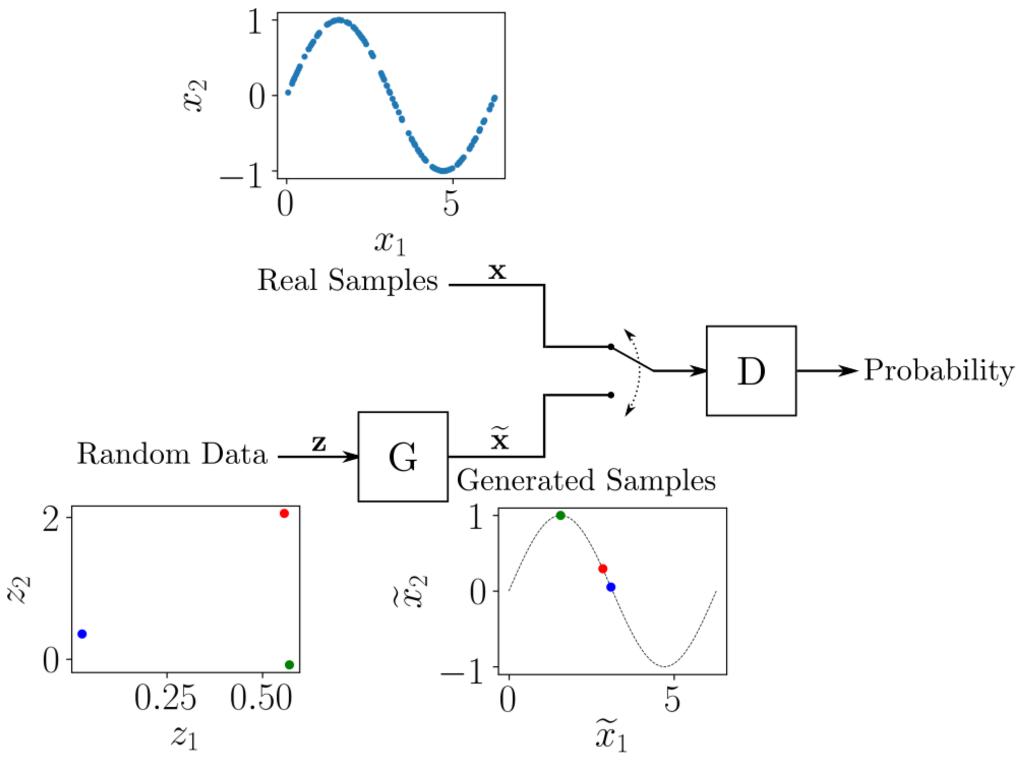
To understand how GAN training works, consider a toy example with a dataset composed of two-dimensional samples (x_1, x_2), with x_1 in the interval from 0 to 2π and $x_2 = sin(x_1)$, as illustrated in the following figure:



<u>Learning Discriminator (D)</u>

- Sample m real examples $\{x^1, x^2, \ldots, x^m\}$ from database.
- Sample m random examples as the noise $\{z^1, z^2, \ldots, z^m\}$ from a distribution, e.g., standard normal distribution with mean μ and variance σ^2.
- Obtain the generated (fake) sample $\{\tilde{x}^1, \tilde{x}^2, \ldots, \tilde{x}^m\}$, $\tilde{x}^i = G(z^i)$ by feeding the noise sample into Generator G.
- Update the parameter θ_d of Discriminator (D) by maximizing the log likelihood $\tilde{V}$:
  - $\tilde{V} = \frac{1}{m}\sum_{i=1}^{m}\log D(x^i) + \frac{1}{m}\sum_{i=1}^{m}\log[1 - D(\tilde{x}^i)]$
  - The parameter θ_d can be estimated via Gradient Descent: $\theta_d \leftarrow \theta_d + \eta\nabla\tilde{V}(\theta_d)$

In the objective function $\tilde{V}$, the firs term aims to maximize the log likelihood when feeding m real examples $\{x^1, x^2, \ldots, x^m\}$ into Discriminator (D). The log likelihood is calculated by taking average of the logarithmic predictive densities: $\log D(x^i)$. Gradient Descent is set to look for parameters θ_d in way to maximize the log likelihood. The second term aims to impose penalties onto the generated (fake) sample. When feeding the generated sample $\{\tilde{x}^1, \tilde{x}^2, \ldots, \tilde{x}^m\}$ (i.e., the real sample mingled with the noise by Generator G), we want to keep $D(\tilde{x}^i)$ as small as possible, while $\tilde{V}$ gets larger and larger by updating parameter θ_d using Gradient Descent.



The generator G is fed with random data from a latent space, and its role is to generate data resembling the real samples. In this example, you have a two-dimensional latent space, so that the generator is fed with random noise (z_1, z_2) pairs and is required to transform them so that they resemble the real samples.

The structure of the neural network G can be arbitrary, allowing you to use neural networks as a multilayer perceptron (MLP), a convolutional neural network (CNN), or any other structure as long as the dimensions of the input and output match the dimensions of the latent space and the real data.

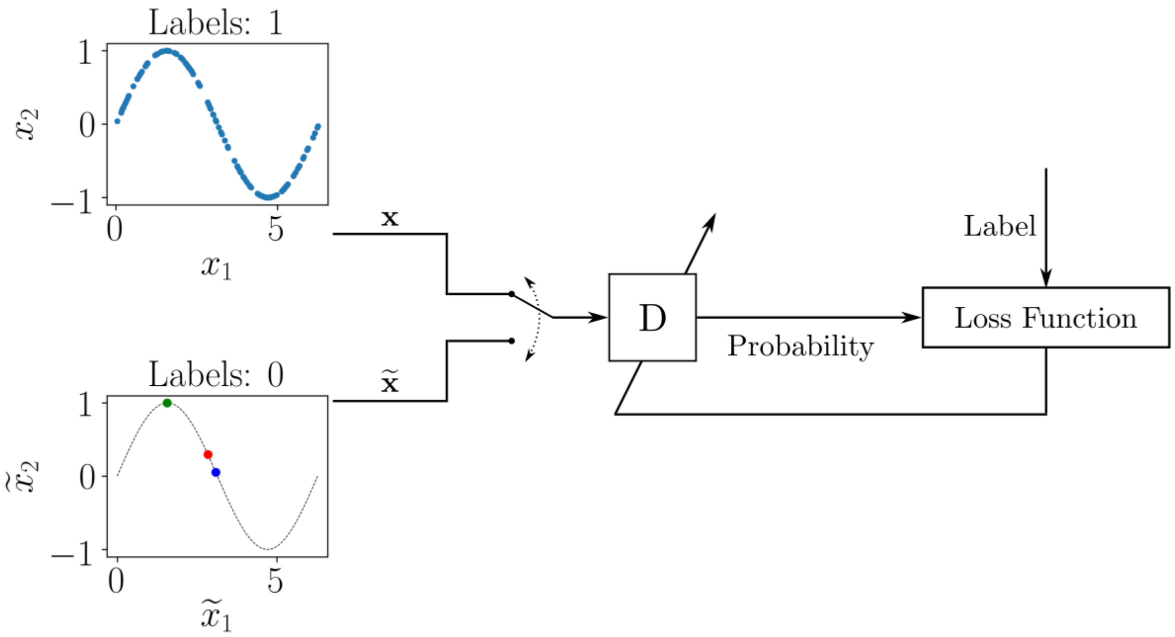
The discriminator D is fed with either real samples from the training dataset or generated samples provided by G. Its role is to estimate the probability that the input belongs to the real dataset. The training is performed so that D outputs 1 when it's fed a real sample and 0 when it's fed a generated sample.

As with G, you can choose an arbitrary neural network structure for D as long as it respects the necessary input and output dimensions. In this example, the input is two-dimensional. For a binary discriminator, the output may be a scalar ranging from 0 to 1.

The GAN training process consists of a two-player minimax game in which D is adapted to minimize the discrimination error between real and generated samples, and G is adapted to maximize the probability of D making a mistake.

Although the dataset containing the real data isn't labeled, the training processes for D and G are performed in a supervised way. At each step in the training, D and G have their parameters updated. In fact, in the original GAN proposal, the parameters of D are updated k times, while the parameters of G are updated only once for each training step. However, to make the training simpler, you can consider k equal to 1.

To train D, at each iteration you label some real samples taken from the training data as 1 and some generated samples provided by G as 0. This way, you can use a conventional supervised training framework to update the parameters of D in order to minimize a loss function, as shown in the following scheme:

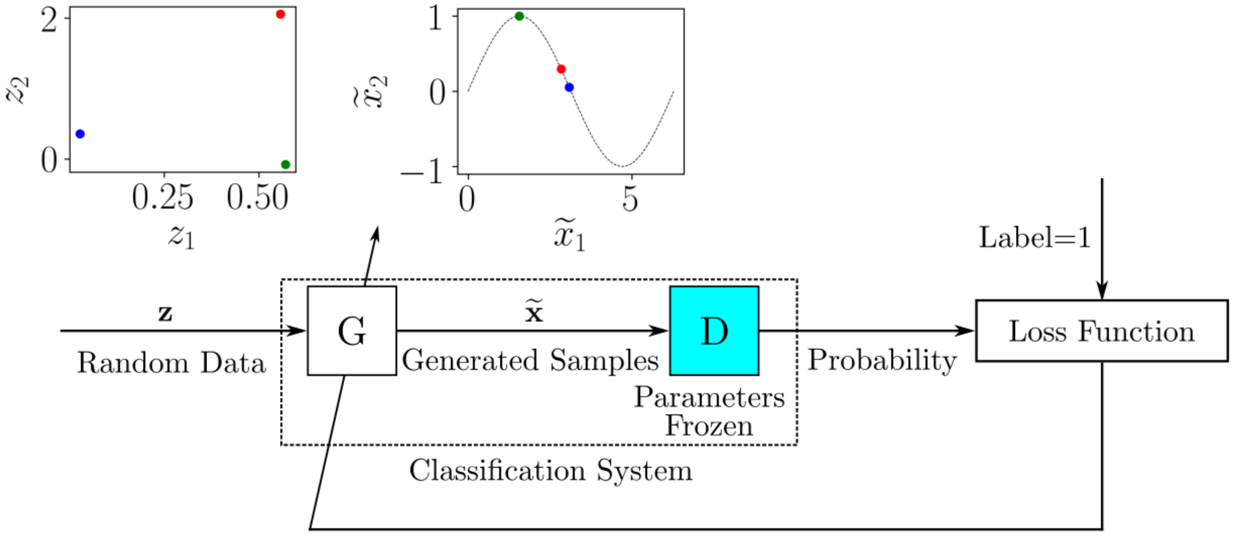


Learning Generator (G)

- Sample m noise samples $\{z^1, z^2, \dots, z^m\}$ from a distribution, e.g., Uniform distribution from $[0, 1]$. Student-t distribution with mean μ, variance σ^2 and the degree of freedom γ. This noise sample is different from the noise prepared when learning Discriminator (D).
- Update the parameter θ_g of Generator G by maximizing the log likelihood $\tilde{V}$:
  - ○ $\tilde{V} = \frac{1}{m}\sum_{i=1}^{m} \log\{D[G(z^i)]\}$
  - ○ $\theta_g \leftarrow \theta_g + \eta\nabla\tilde{V}(\theta_g)$

The log likelihood function indicates that when feeding the generated (fake) sample from the Generator G into the Discriminator (D), we want to train the Generator G in way to maximize the log likelihood from applying the generated (fake) sample onto Discriminator (D). In this sense, Generator G learns to "fool" the Discriminator (D).

For each batch of training data containing labeled real and generated samples, you update the parameters of D to minimize a loss function. After the parameters of D are updated, you train G to produce better generated samples. The output of G is connected to D, whose parameters are kept frozen, as depicted here:



You can imagine the system composed of G and D as a single classification system that receives random samples as input and outputs the classification, which in this case can be interpreted as a probability.

When G does a good enough job to fool D, the output probability should be close to 1. You could also use a conventional supervised training framework here: the dataset to train the classification system composed of G and D would be provided by random input samples, and the label associated with each input sample would be 1.

During training, as the parameters of D and G are updated, it's expected that the generated samples given by G will more closely resemble the real data, and D will have more trouble distinguishing between real and generated data.

Now that you know how GANs work, you’re ready to implement your first GAN.

6.1.4 Application Of Generative Adversarial Networks (GANs)

GANs, or Generative Adversarial Networks, have many uses in many different fields. Here are some of the widely recognized uses of GANs:

Image Synthesis and Generation: GANs are often used for picture synthesis and generation tasks, They may create fresh, lifelike pictures that mimic training data by learning the distribution that explains the dataset. The development of lifelike avatars, high-resolution photographs, and fresh artwork have all been facilitated by these types of generative networks.

Image-to-Image Translation: GANs may be used for problems involving image-to-image translation, where the objective is to convert an input picture from one domain to another while maintaining its key features. GANs may be used, for instance, to change pictures from day to night, transform drawings into realistic images, or change the creative style of an image.

Text-to-Image Synthesis: GANs have been used to create visuals from descriptions in text. GANs may produce pictures that translate to a description given a text input, such as a phrase or a caption. This application might have an impact on how realistic visual material is produced using text-based instructions.

Data Augmentation: GANs can augment present data and increase the robustness and generalizability of machine-learning models by creating synthetic data samples.

Data Generation for Training: GANs can enhance the resolution and quality of low-resolution images. By training on pairs of low-resolution and high-resolution images, GANs can generate high-resolution images from low-resolution inputs, enabling improved image quality in various applications such as medical imaging, satellite imaging, and video enhancement.

Reference

Ian Goodfellow et al. in 2014:
https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

https://www.tensorflow.org/tutorials/generative/dcgan

https://realpython.com/generative-adversarial-networks/

https://www.datacamp.com/tutorial/generative-adversarial-networks

https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/

https://towardsdatascience.com/gan-by-example-using-keras-on-tensorflow-backend-1a6d515a60d0

https://www.geeksforgeeks.org/generative-adversarial-network-gan/