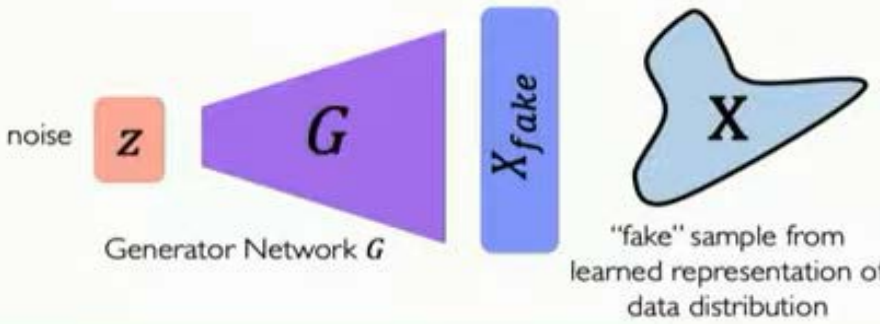
Generative Adversarial Networks (GANs)

What if we just want to sample?

Idea: don't explicitly model density, and instead just sample to generate new instances.

Problem: want to sample from complex distribution – can't do this directly!

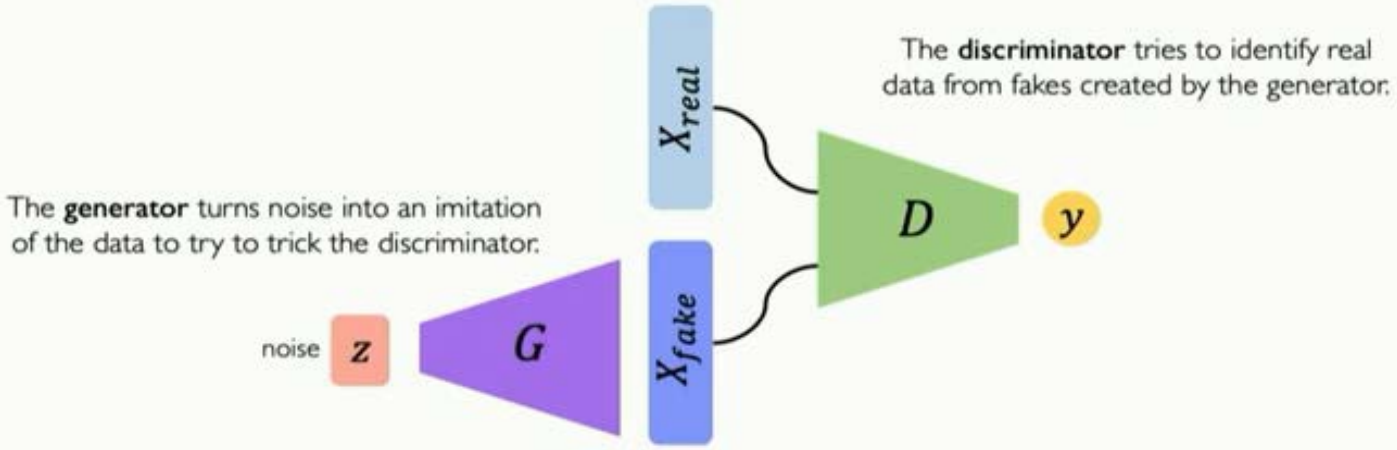
Solution: sample from something simple (e.g., noise), learn a transformation to the data distribution.



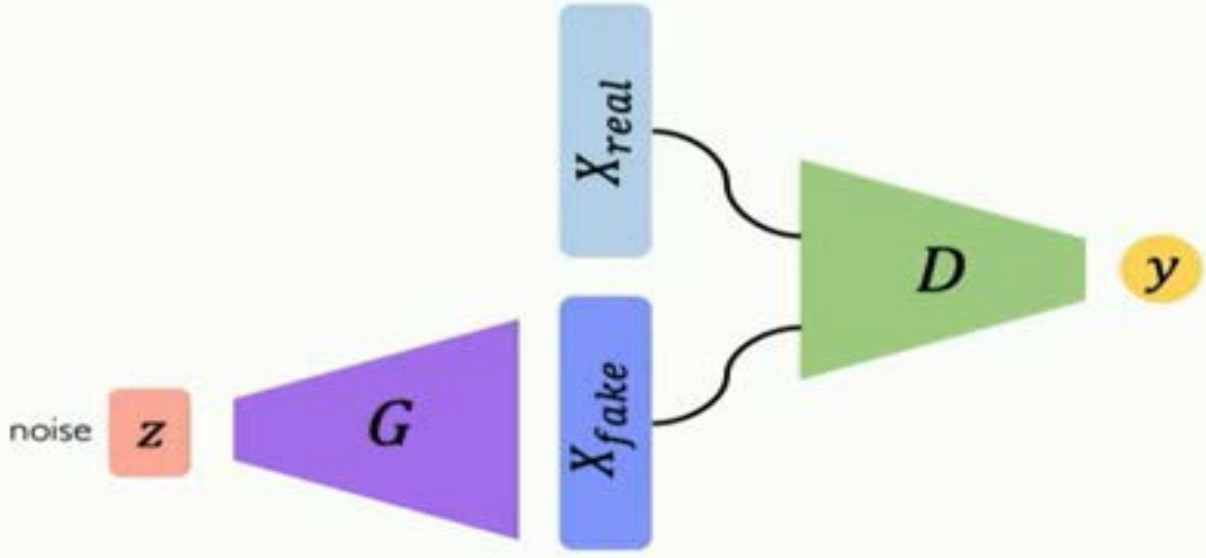
GAN : The distribution is not assumed as Normal. It is a very complex distribution. It's a multi-Gaussian distribution. From the complex distribution, select a very simple noise and try to generate the synthetic new images.

Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are a way to make a generative model by having two neural networks compete with each other.

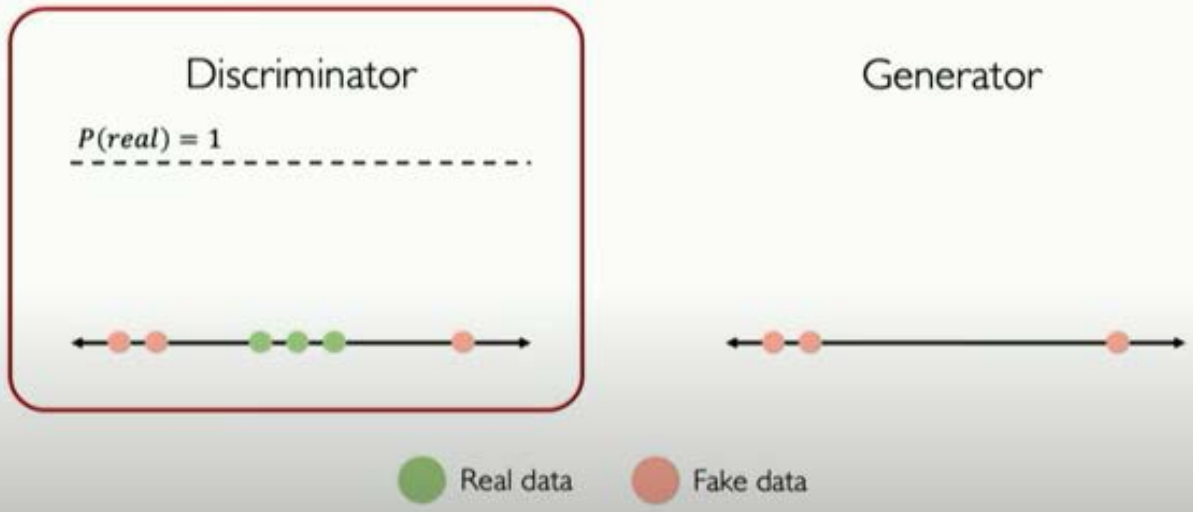


Training GANs



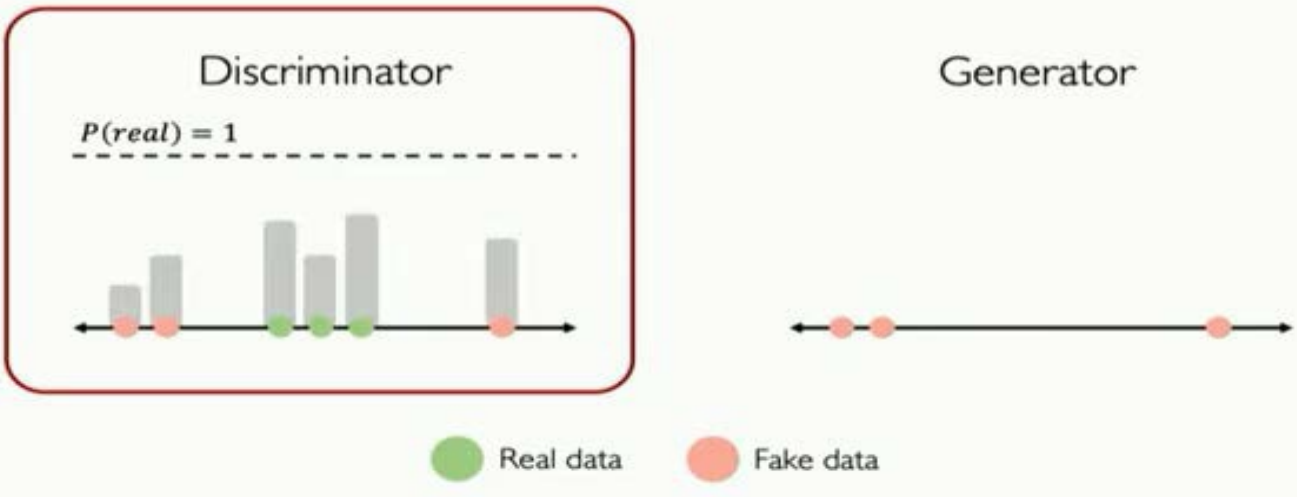
Intuition behind GANs

Discriminator tries to predict what's real and what's fake.



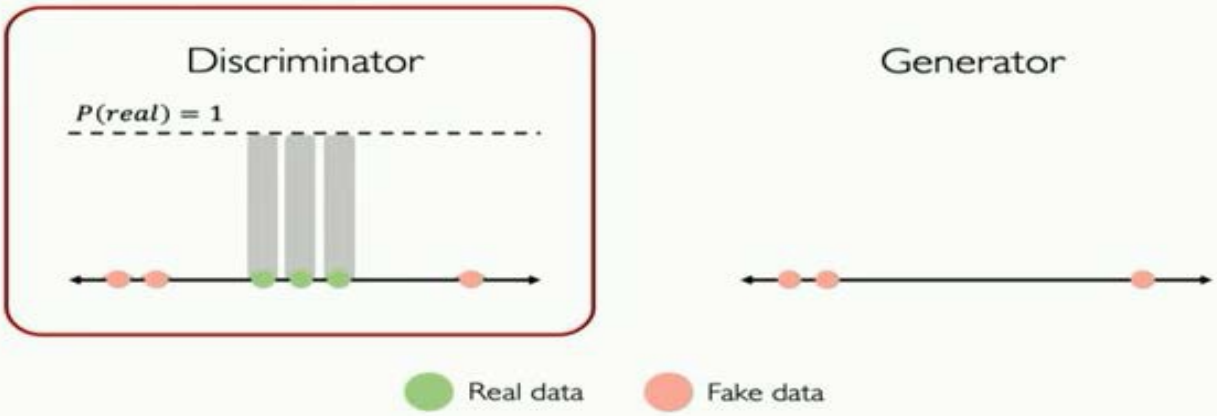
Intuition behind GANs

Discriminator tries to predict what's real and what's fake.



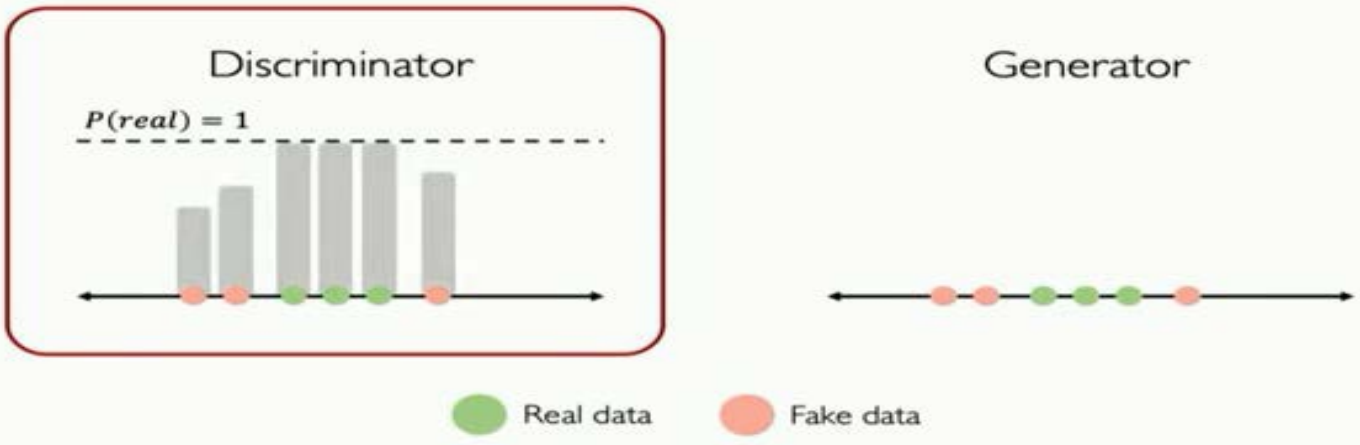
Intuition behind GANs

Discriminator tries to predict what's real and what's fake.



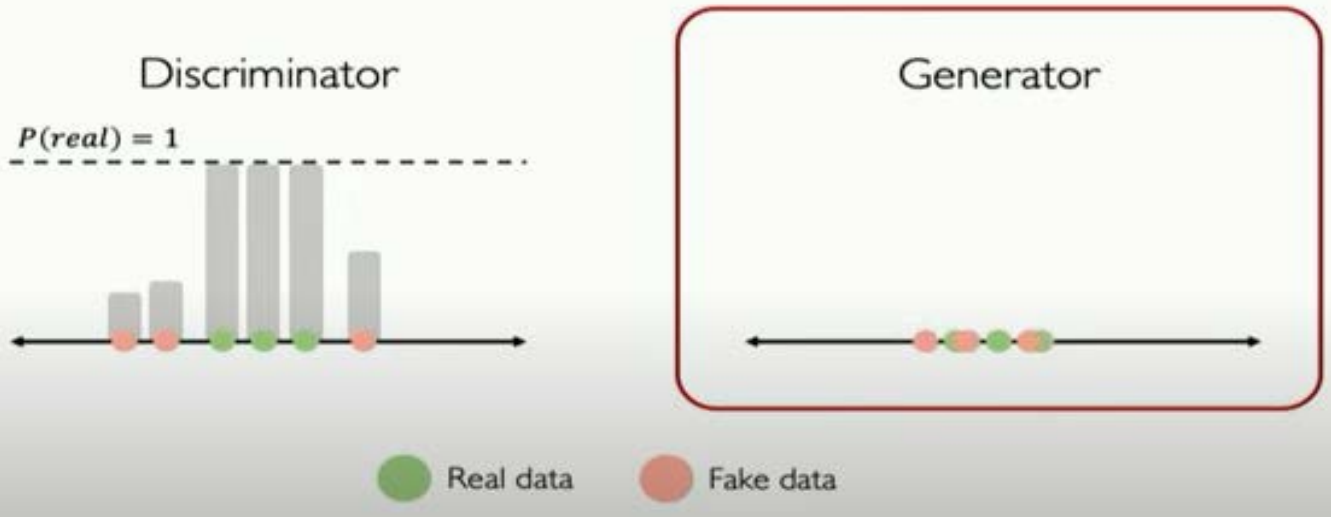
Intuition behind GANs

Discriminator tries to predict what's real and what's fake.



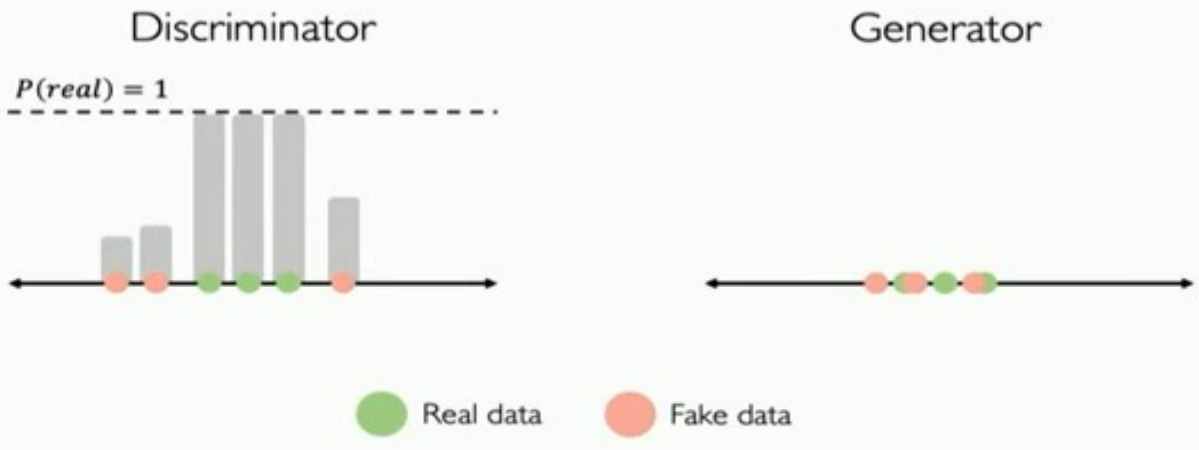
Intuition behind GANs

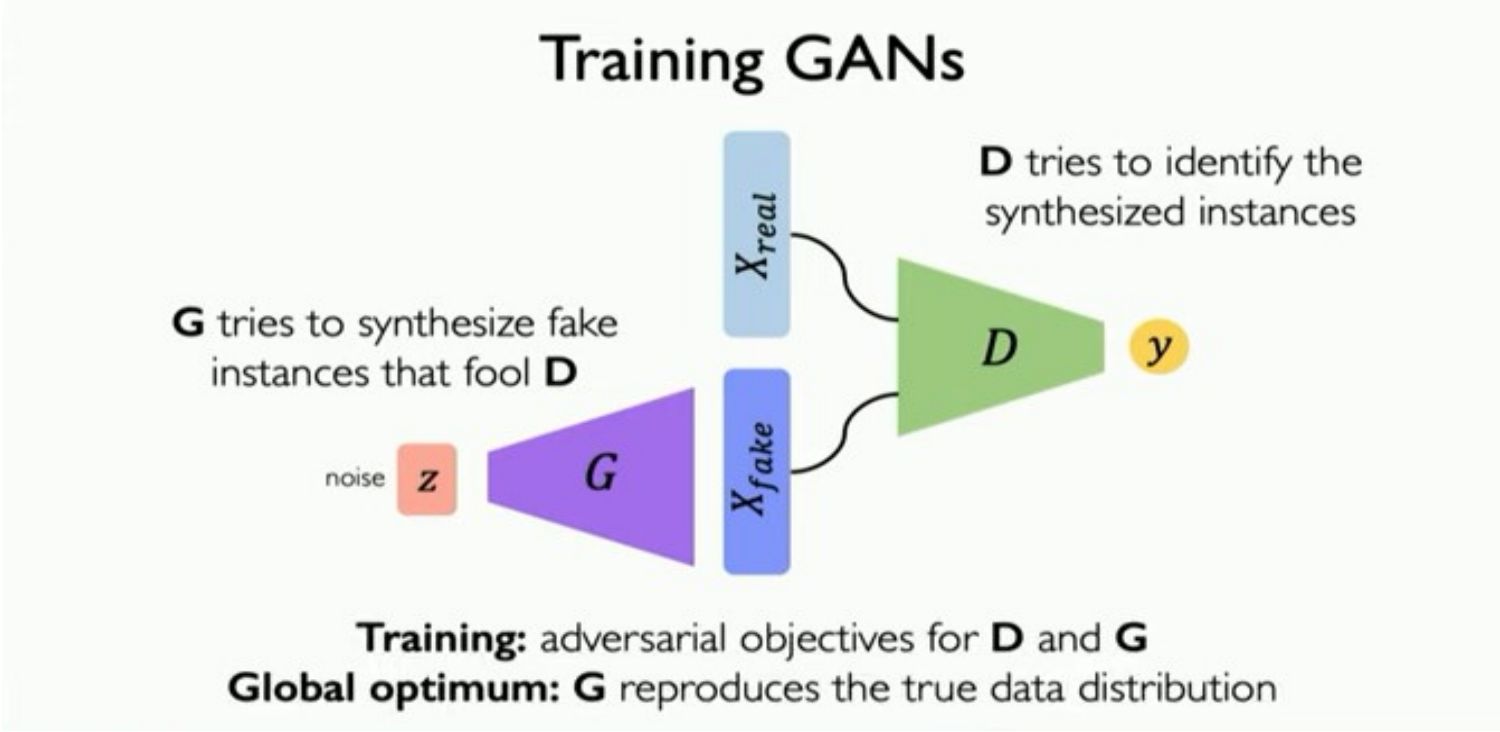
Generator tries to improve its imitation of the data.



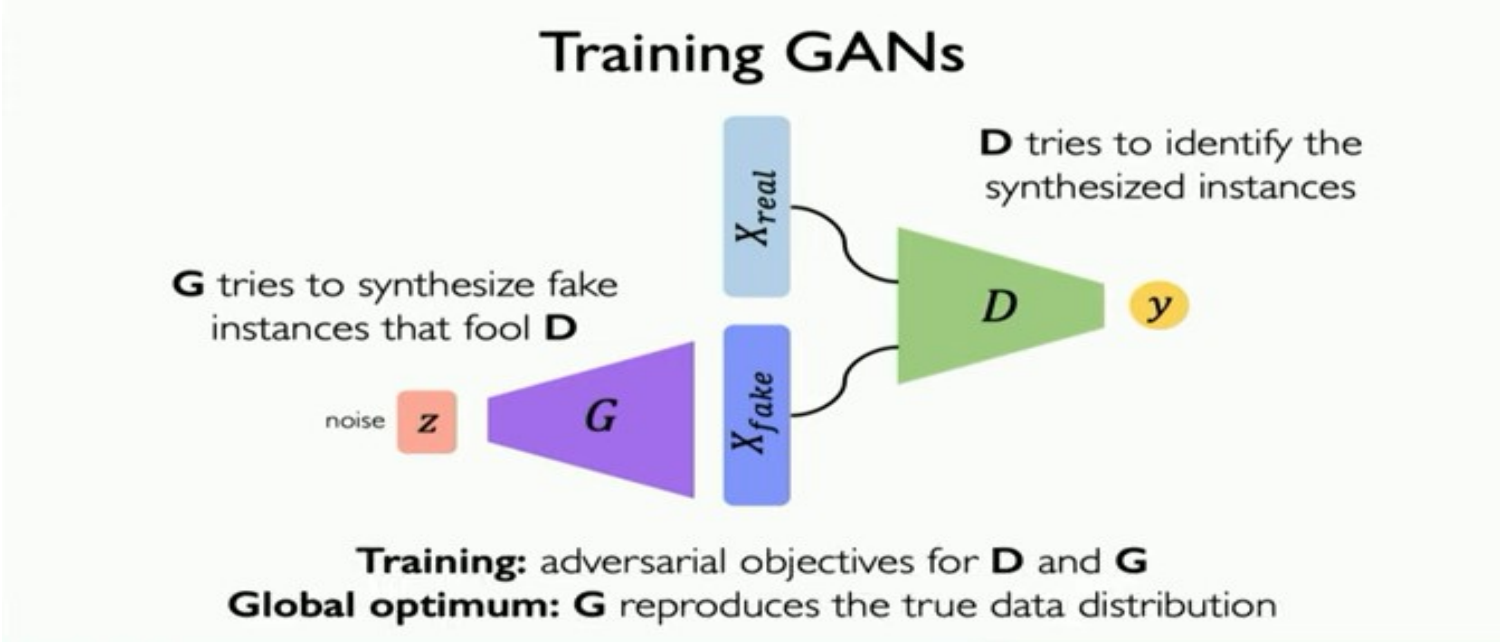
Intuition behind GANs

Discriminator tries to identify real data from fakes created by the generator.
Generator tries to create imitations of data to trick the discriminator.





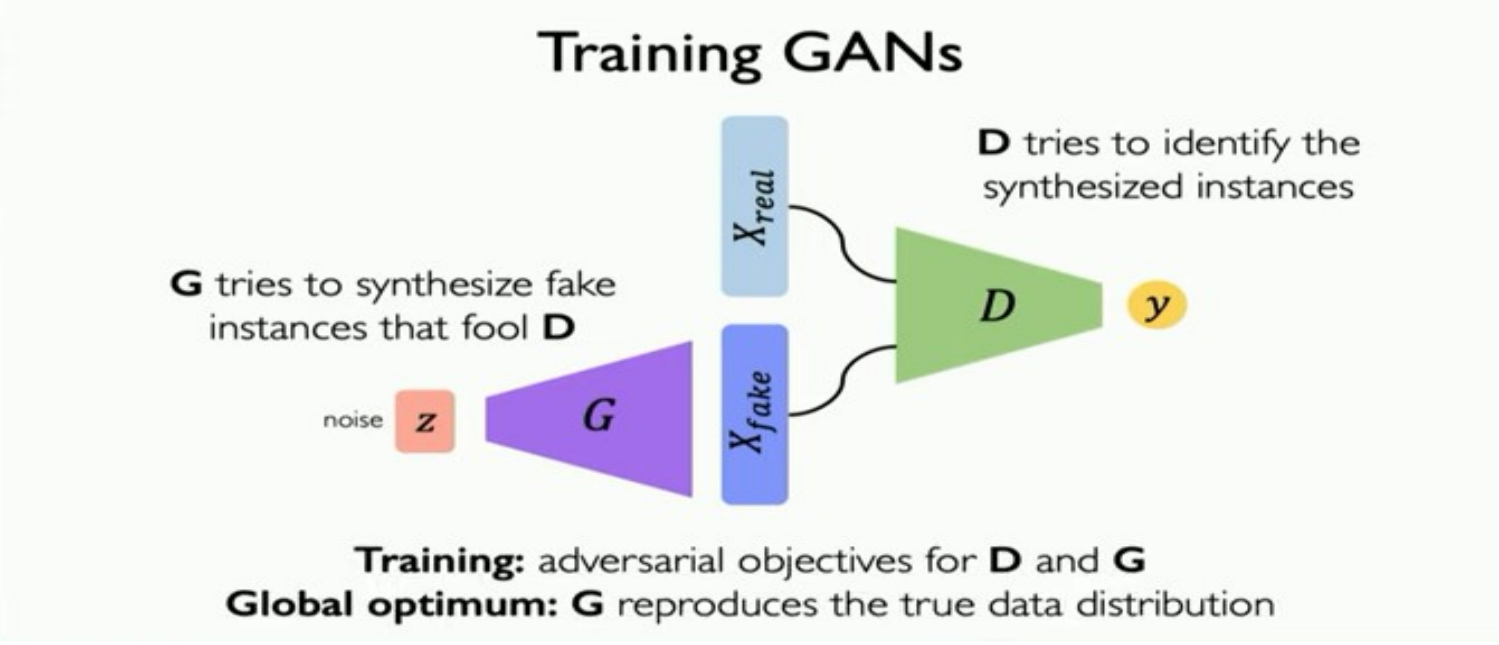
GAN : Gen: Fake Data. Dis: Estimated probability as input is Fake.
The weights in the generator model are updated based on the performance of the discriminator model.
When the discriminator is good at detecting fake samples, the generator is updated more, and when the discriminator model is relatively poor or confused when detecting fake samples, the generator model is updated less.



GAN : Specifically, a new GAN model can be defined that stacks the generator and discriminator such that the generator receives as input random points in the latent space and generates samples that are fed into the discriminator model directly, classified, and the output of this larger model can be used to update the model weights of the generator.

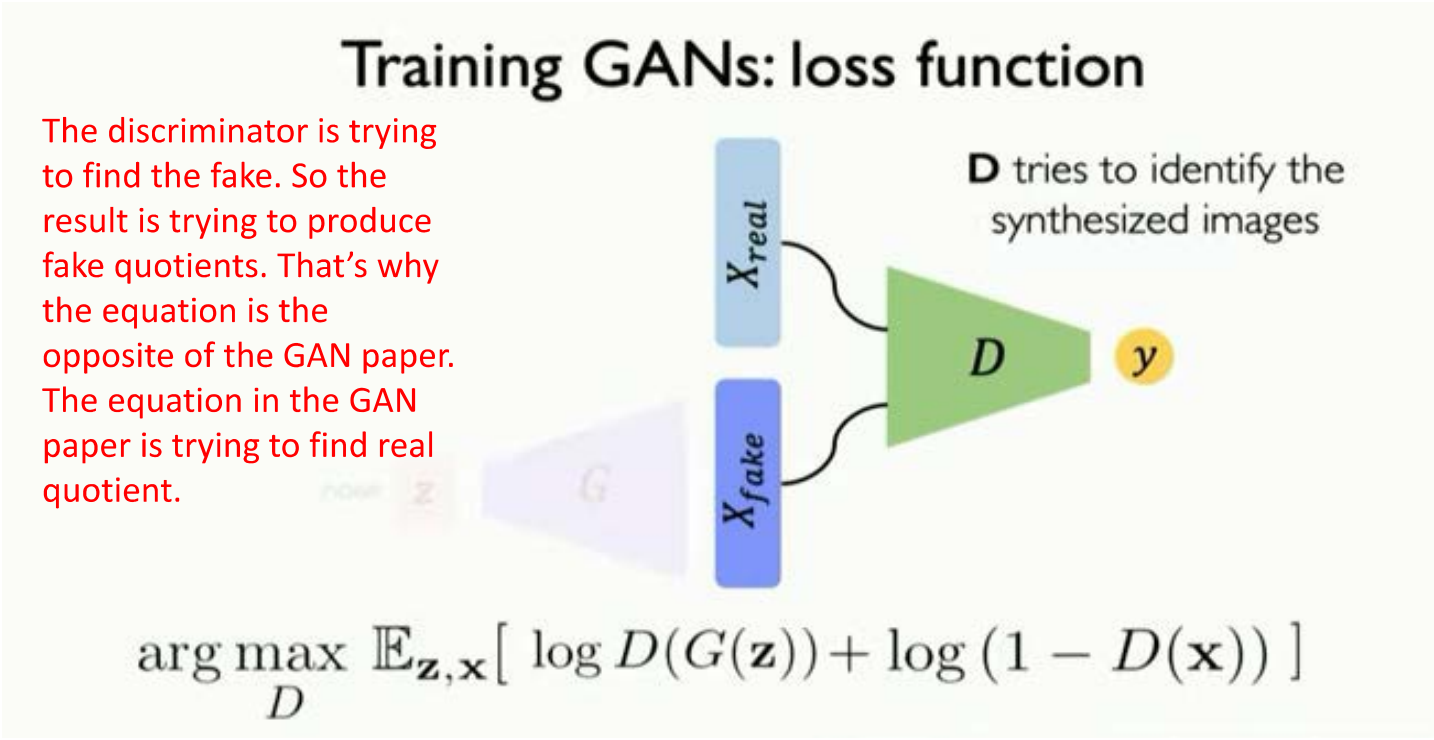
To be clear, we are not talking about a new third model, just a new logical model that uses the already-defined layers and weights from the standalone generator and discriminator models.

The discriminator is concerned with distinguishing between real and fake examples, therefore the discriminator model can be trained in a standalone manner on examples of each.

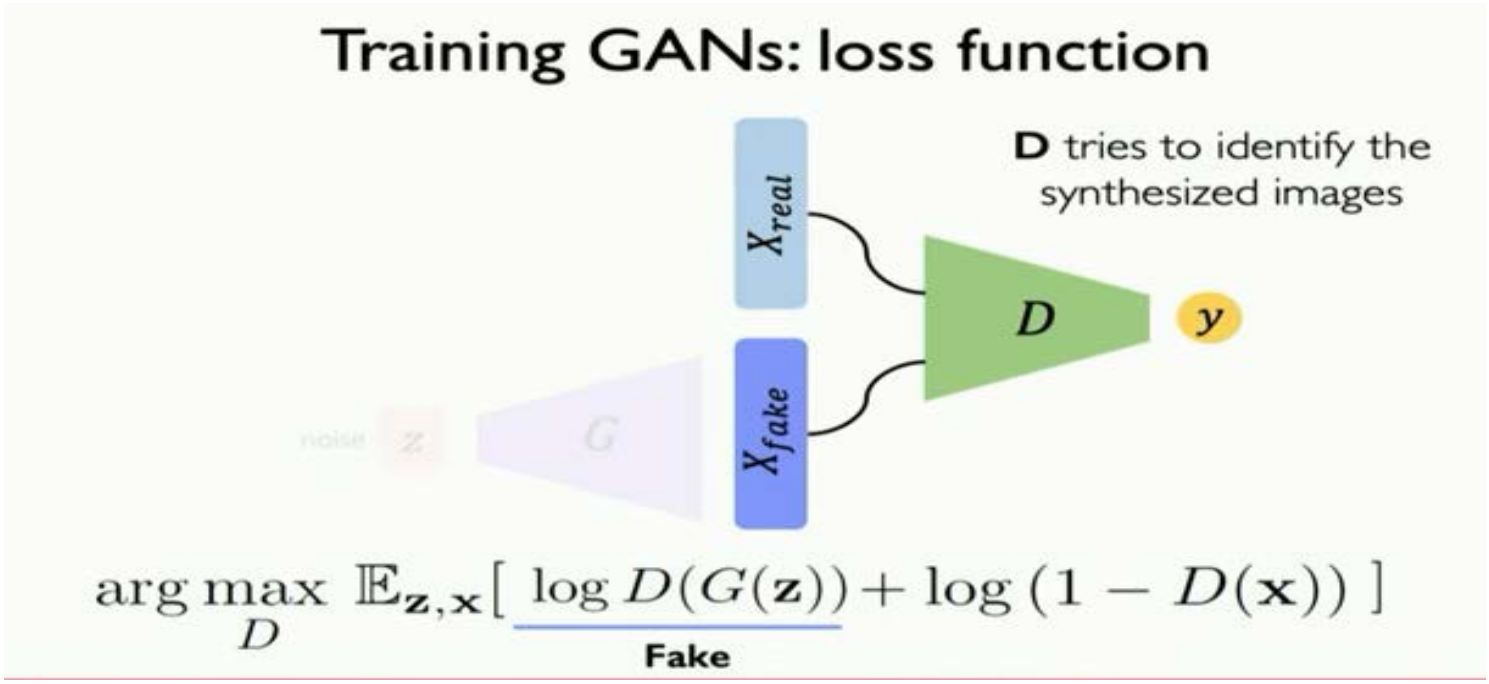


GAN :The generator model is only concerned with the discriminator's performance on fake examples. Therefore, we will mark all of the layers in the discriminator as not trainable when it is part of the GAN model so that they can not be updated and overtrained on fake examples.

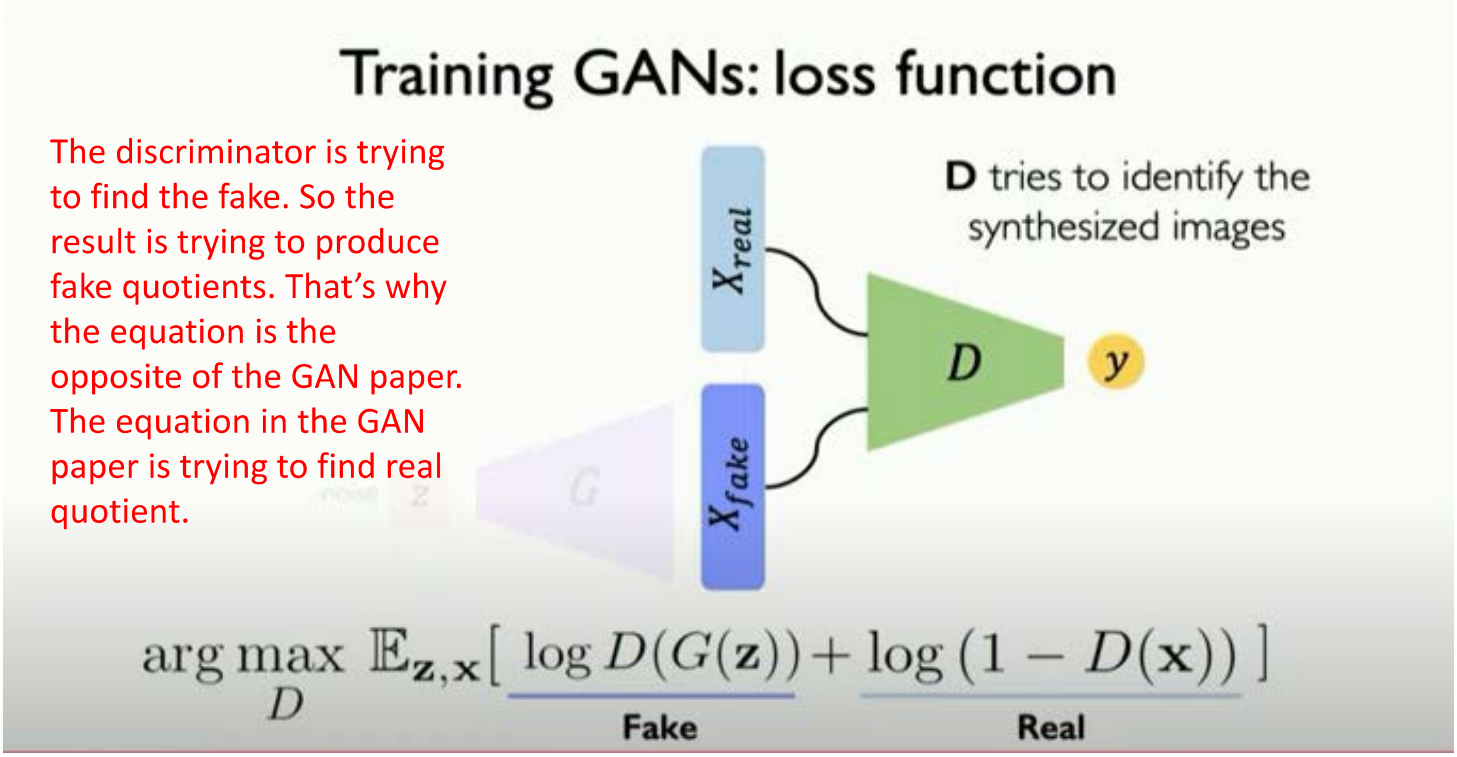
When training the generator via this logical GAN model, there is one more important change. We want the discriminator to think that the samples output by the generator are real, not fake. Therefore, when the generator is trained as part of the GAN model, we will mark the generated samples as real (class 1).



GAN : Maximize the probability of decisions of real data as classified as real and fake data as fake.

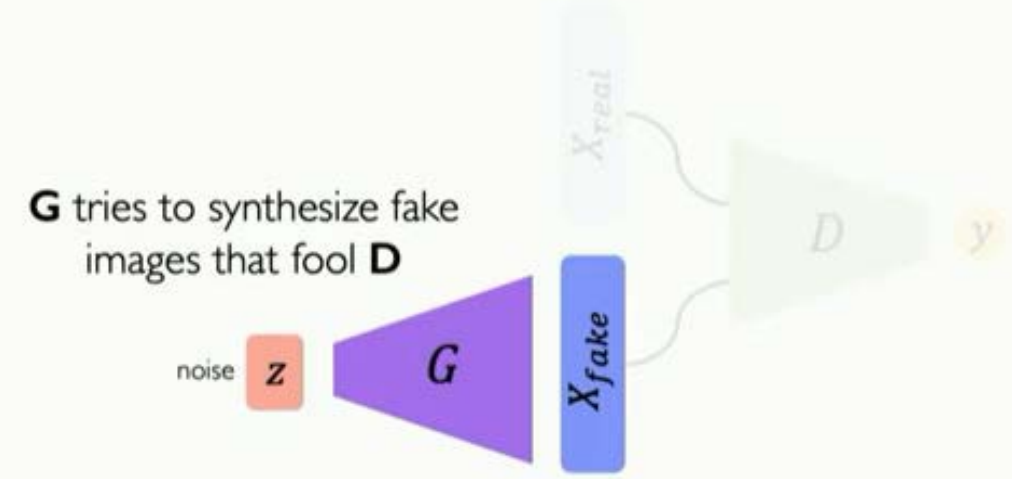


Dis: D(Gen(input=z)) = estimate as being fake. D(X) = estimate of real instance 'X' as fake. i.e. (1 -D(X)) = real as real. Fake data , real data . Together it tries to maximize probability to get answer correct.



Dis: 'X' is data from real dataset. D(X) = estimate of real instance 'X' as fake. i.e. (1 - D(X)) = real as real. Fake data , real data, together it tries to maximize probability to get answer correct.

Training GANs: loss function



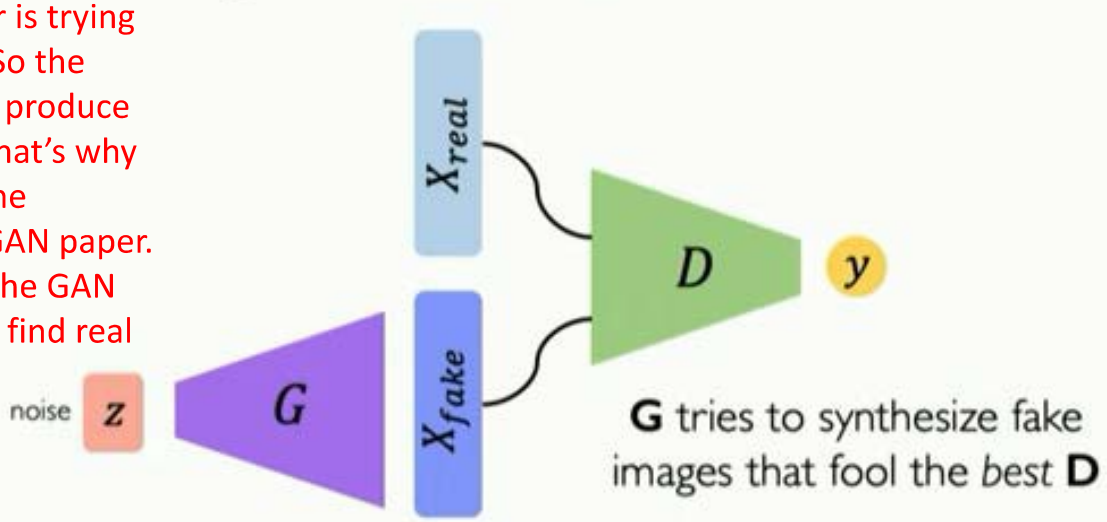
$$\arg\min_G \mathbb{E}_{\mathbf{z},\mathbf{x}}[\log D(G(\mathbf{z})) + \log(1 - D(\mathbf{x}))]$$

Real Fake

Generator: generator generates the examples. Generator focus on minimizing the probability of generated data identified as fake.

Training GANs: loss function

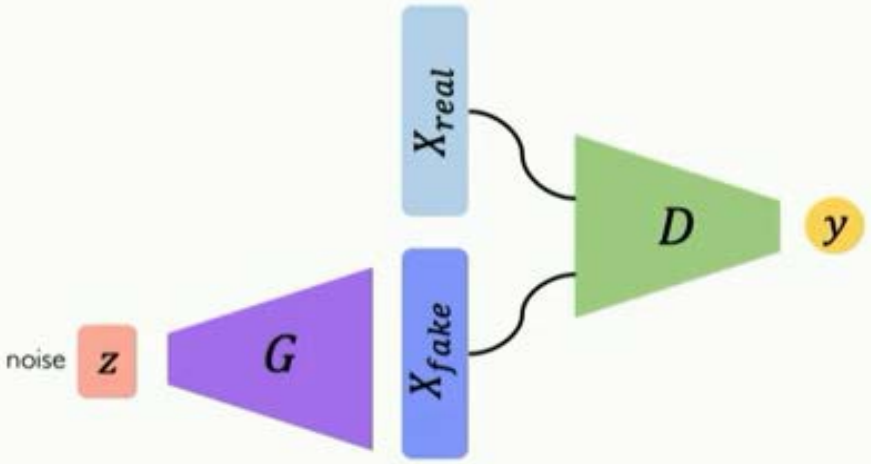
The discriminator is trying to find the fake. So the result is trying to produce fake quotients. That's why the equation is the opposite of the GAN paper. The equation in the GAN paper is trying to find real quotient.



$$\arg\min_G \max_D \mathbb{E}_{\mathbf{z},\mathbf{x}}[\log D(G(\mathbf{z})) + \log(1 - D(\mathbf{x}))]$$

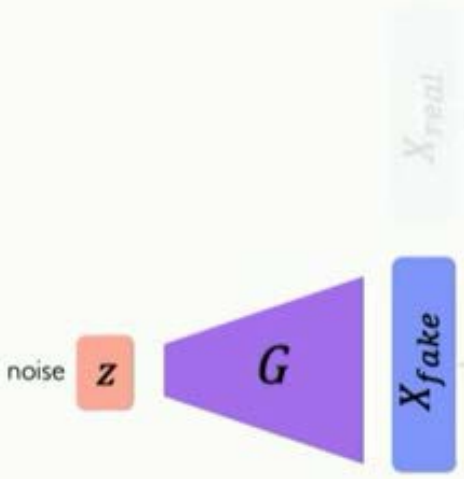
There may be many ways to implement this using the Keras API, but perhaps the simplest approach is to create a new model that combines the generator and discriminator models.

Generating new data with GANs



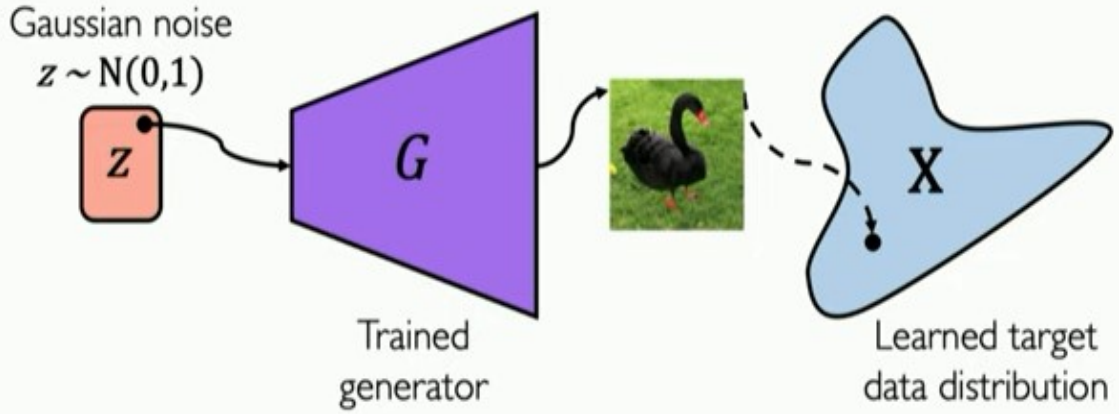
After training, use generator network to create **new data** that's never been seen before.

Generating new data with GANs

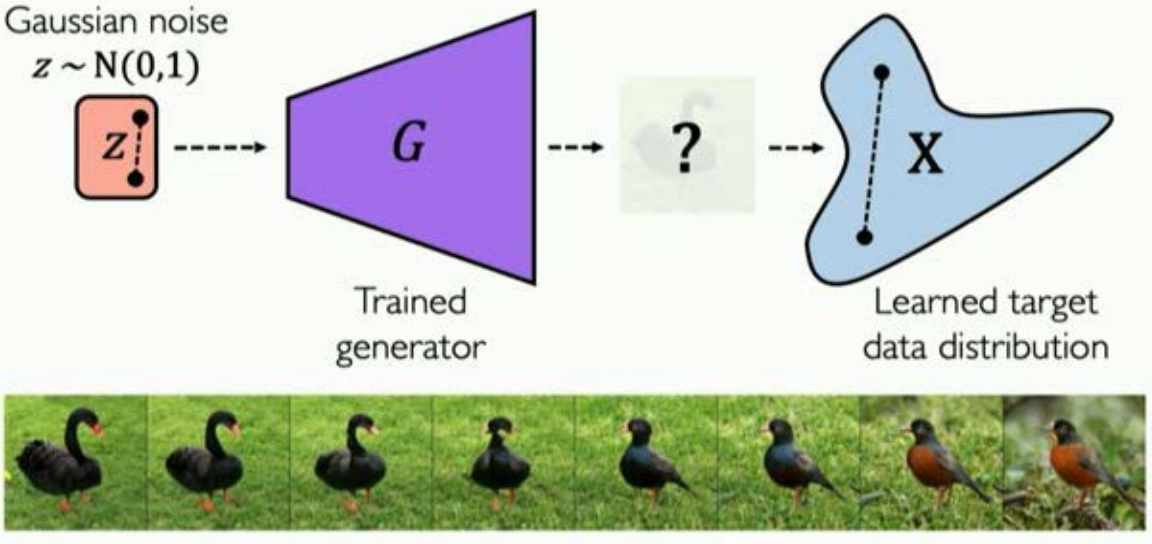


After training, use generator network to create **new data** that's never been seen before.

GANs are distribution transformers



GANs are distribution transformers



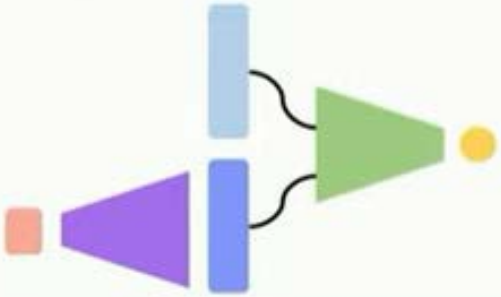
Deep Generative Modeling: Summary

Autoencoders and Variational Autoencoders (VAEs)

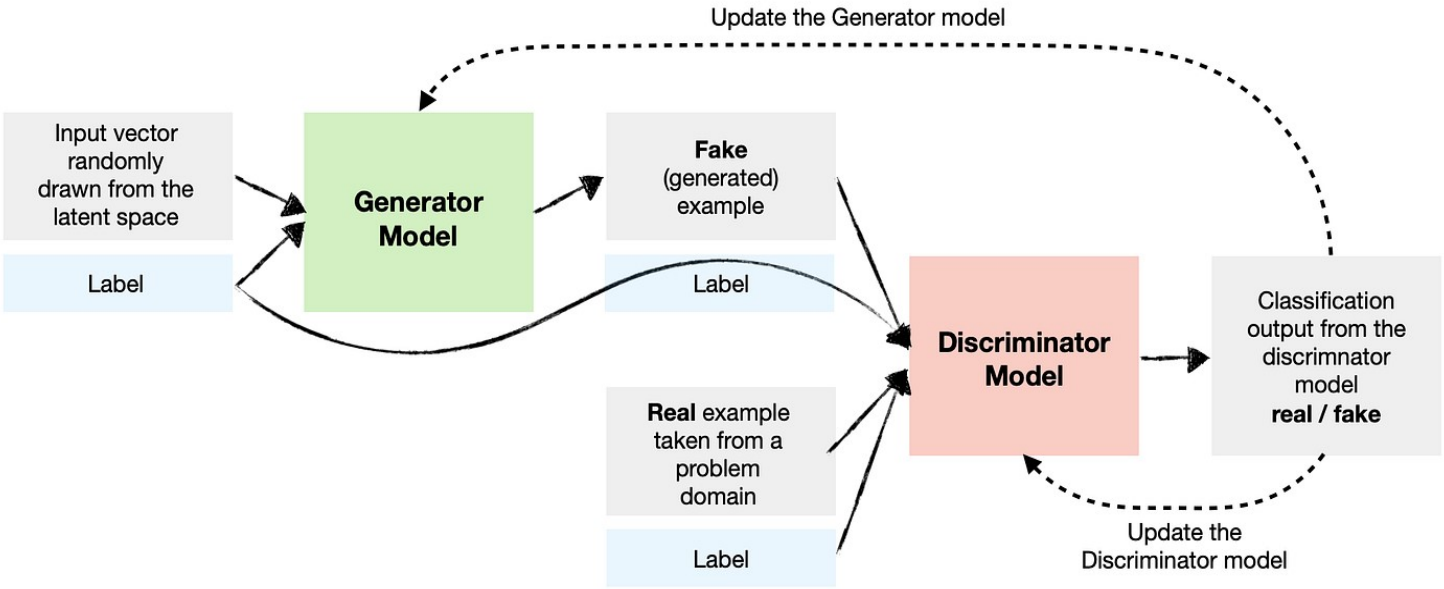
Learn lower-dimensional **latent space** and **sample** to generate input reconstructions

Generative Adversarial Networks (GANs)

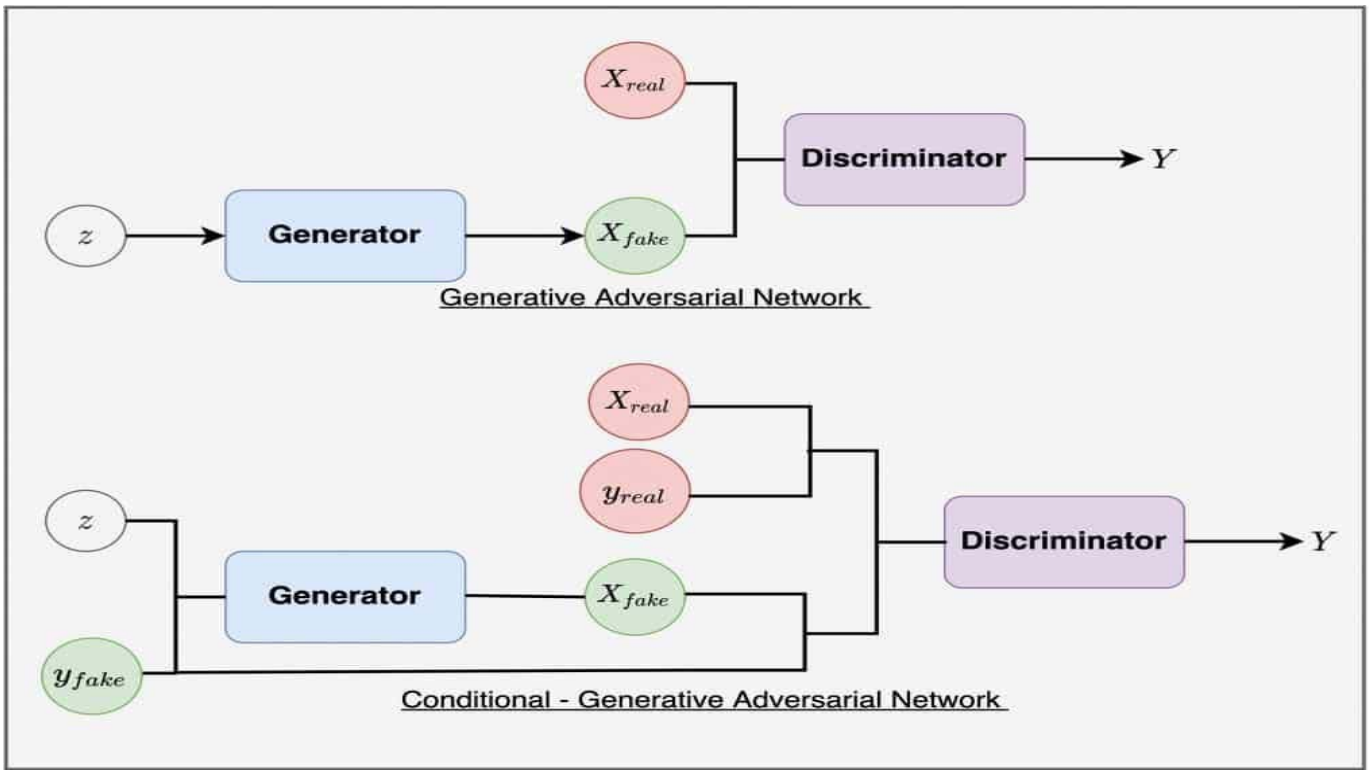
Competing **generator** and **discriminator** networks



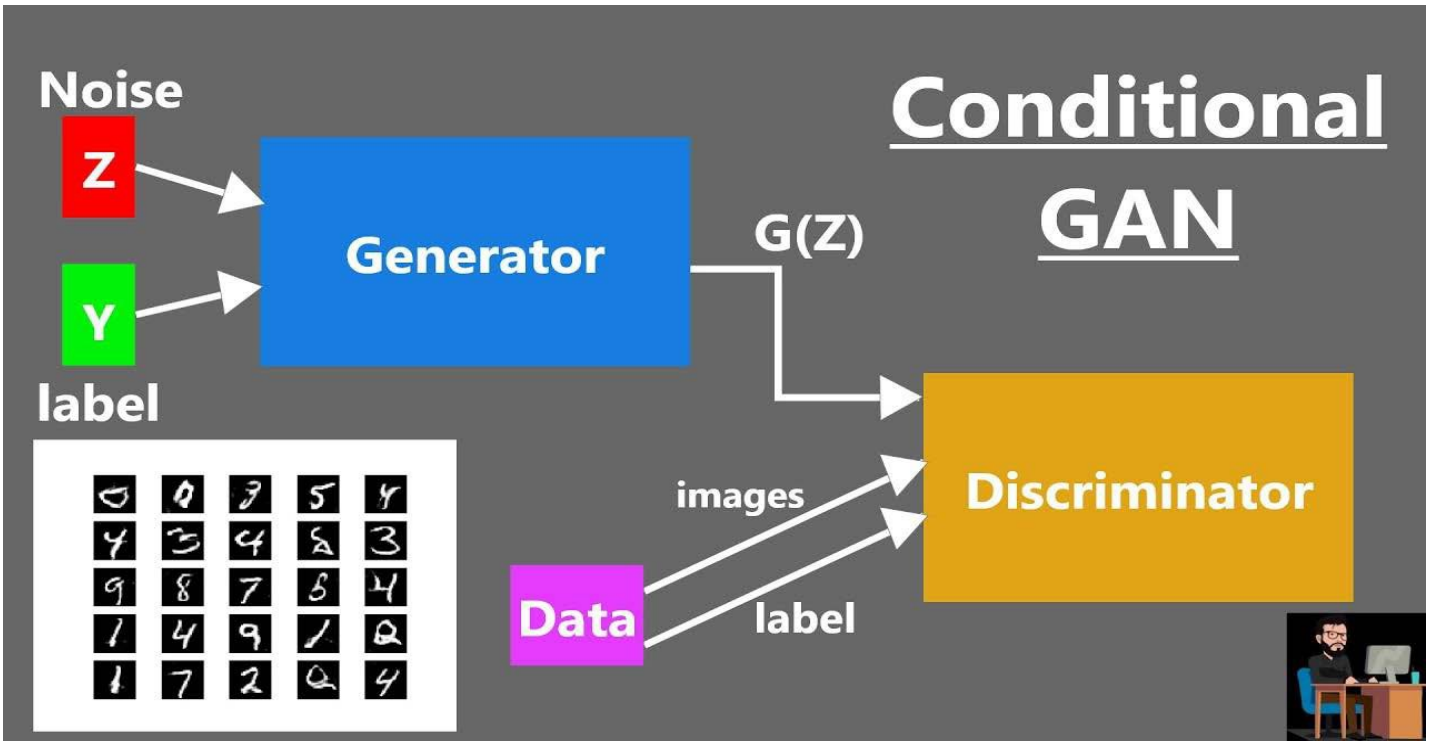
cGAN: Conditional Generative Adversarial Network

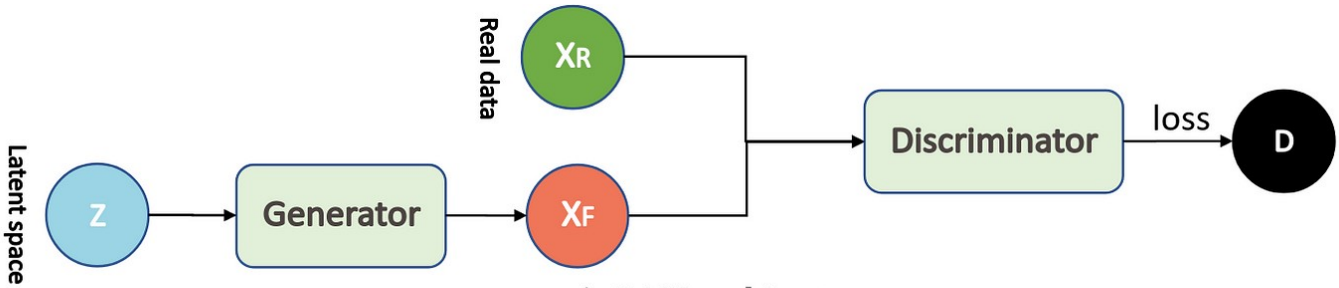


cGAN: Conditional Generative Adversarial Network

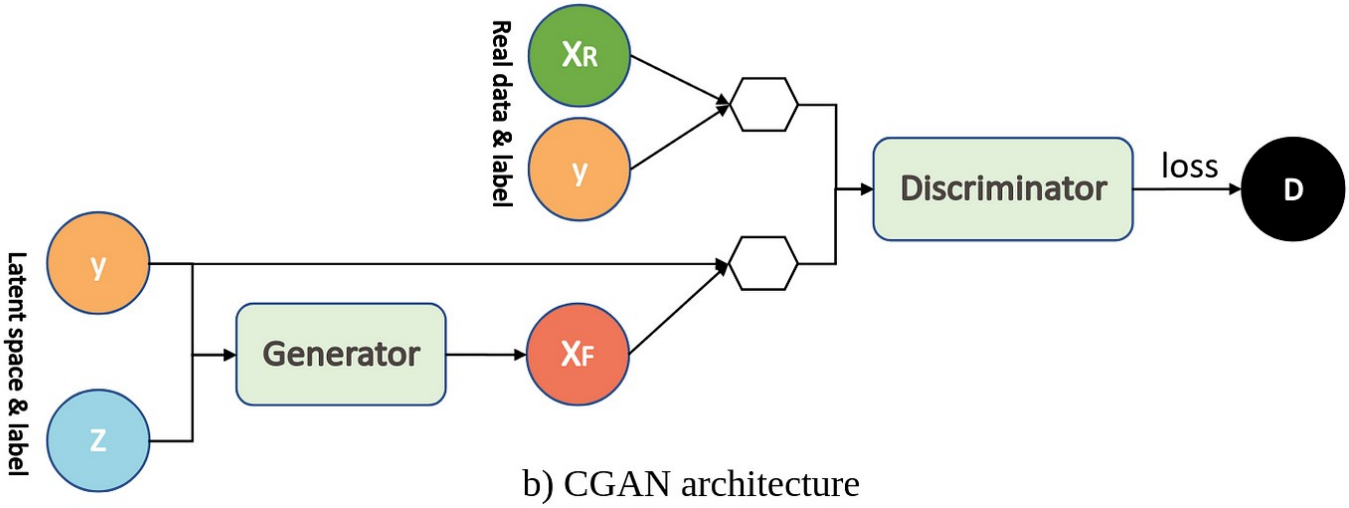


cGAN: Conditional Generative Adversarial Network

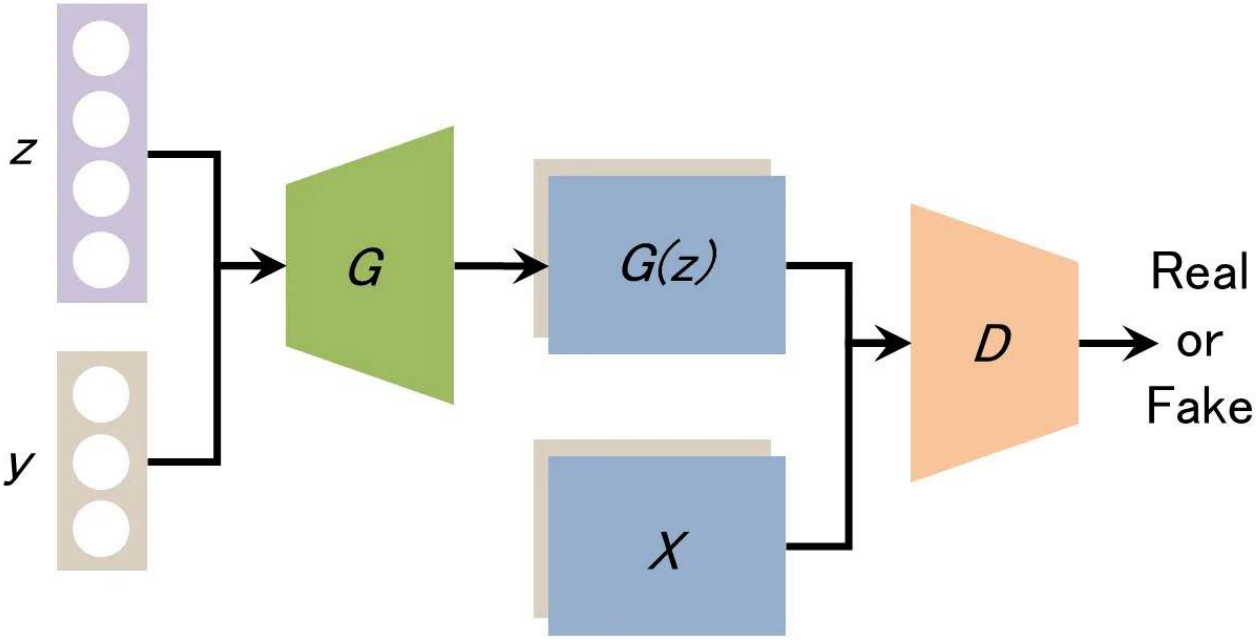


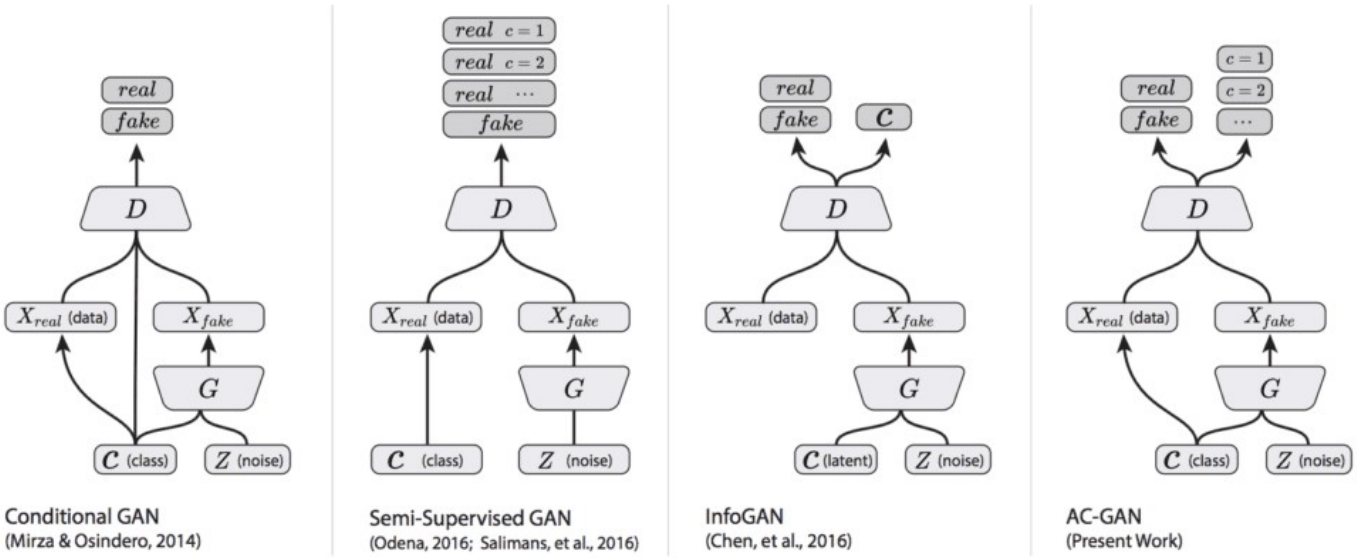


a) GAN architecture



b) CGAN architecture





In a Conditional Generative Adversarial Network (cGAN) and an Auxiliary Classifier Generative Adversarial Network (ACGAN), the loss functions are similar but with some differences due to the presence of an auxiliary classifier in the discriminator of ACGAN. Let's break down the loss functions for both:

1.Conditional Generative Adversarial Network (cGAN):

1. **Discriminator Loss**:
    1. The discriminator's goal is to distinguish between real and fake samples, conditioned on their respective class labels.
    2. The discriminator loss is the sum of two terms:
        1. Real Sample Loss: Measures how well the discriminator classifies real samples. It is typically calculated using binary cross-entropy loss.
        2. Fake Sample Loss: Measures how well the discriminator classifies fake samples generated by the generator. Again, it is typically calculated using binary cross-entropy loss.
    3. The total discriminator loss is the sum of the real and fake sample losses.
2. **Generator Loss**:
    1. The generator's goal is to generate realistic samples that can fool the discriminator, conditioned on specific class labels.
    2. The generator loss is the fake sample loss, which measures how well the generator's output is classified by the discriminator as real. This loss encourages the generator to produce samples that are indistinguishable from real samples.

Auxiliary Classifier Generative Adversarial Network (ACGAN):

•**Discriminator Loss**:

- In addition to distinguishing between real and fake samples, the discriminator in ACGAN also predicts the class labels of the samples.
- The discriminator loss is the sum of three terms:
  - Real Sample Loss: Measures how well the discriminator classifies real samples as real. It is typically calculated using binary cross-entropy loss.
  - Fake Sample Loss: Measures how well the discriminator classifies fake samples generated by the generator as fake. It is also calculated using binary cross-entropy loss.
  - Class Loss: Measures how well the discriminator predicts the class labels of both real and fake samples. It is calculated using categorical cross-entropy loss.
- The total discriminator loss is the sum of the real sample loss, fake sample loss, and class loss.

•**Generator Loss**:

- The generator's goal remains the same as in cGAN: to generate realistic samples that can fool the discriminator.
- The generator loss is the sum of two terms:
  - Fake Sample Loss: Measures how well the generator's output is classified by the discriminator as real. This loss encourages the generator to produce samples that are indistinguishable from real samples.
  - Class Loss: Measures how well the discriminator predicts the class labels of the fake samples. It encourages the generator to produce samples that belong to specific classes.
- The total generator loss is the sum of the fake sample loss and the class loss.

Conditional GAN (cGAN):

1. **Discriminator Loss:**
   - Real Sample Loss (L_{real}):
     $L_{\text{real}} = -\frac{1}{m}\sum_{i=1}^{m} \log D(x_i, y_i)$
   - Fake Sample Loss (L_{fake}):
     $L_{\text{fake}} = -\frac{1}{m}\sum_{i=1}^{m} \log(1 - D(G(z_i, c_i), c_i))$
   - Total Discriminator Loss (L_{D}):
     $L_{\text{D}} = L_{\text{real}} + L_{\text{fake}}$
2. **Generator Loss:**
   - Fake Sample Loss (L_{fake}):
     $L_{\text{fake}} = -\frac{1}{m}\sum_{i=1}^{m} \log D(G(z_i, c_i), c_i)$
   - Total Generator Loss (L_{G}):
     $L_{\text{G}} = L_{\text{fake}}$

Where:

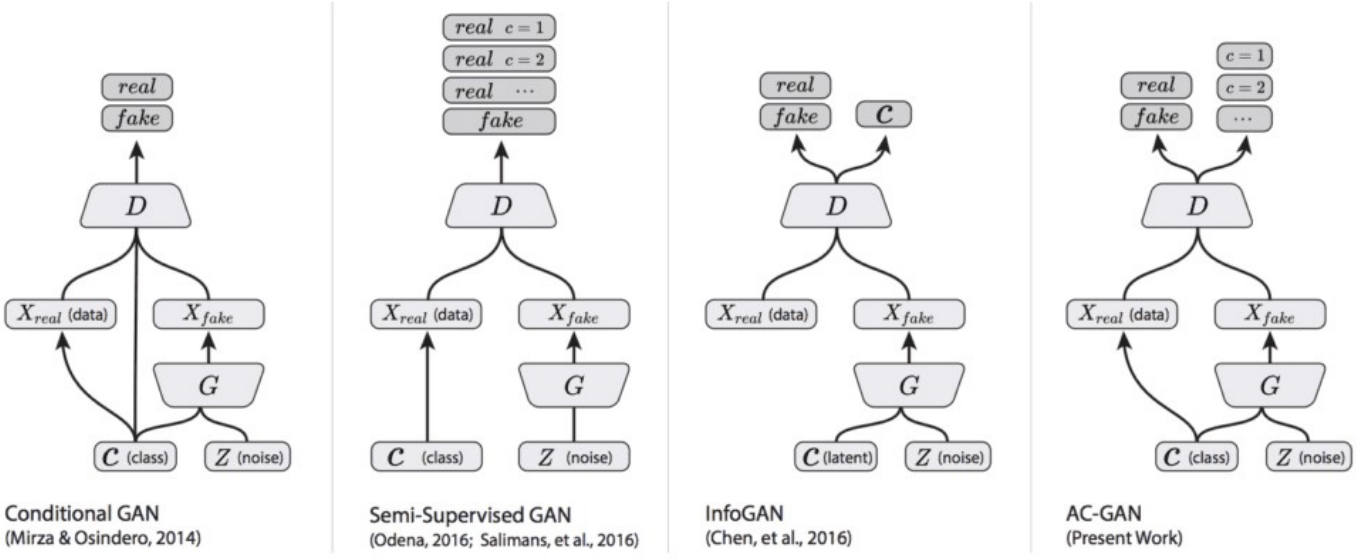
- $D(x, y)$ represents the discriminator's ou[illegible]t for sample x given label y.
- $G(z, c)$ represents the generator's output for noise z and label c.

Auxiliary Classifier GAN (ACGAN):

1. **Discriminator Loss**:
   - Real Sample Loss (L_{real}):
     $L_{\text{real}} = -\frac{1}{m}\sum_{i=1}^{m}\log D(x_i)$
   - Fake Sample Loss (L_{fake}):
     $L_{\text{fake}} = -\frac{1}{m}\sum_{i=1}^{m}\log(1 - D(G(z_i, c_i)))$
   - Class Loss (L_{class}):
     $L_{\text{class}} = -\frac{1}{m}\sum_{i=1}^{m}\log C(y_i|x_i)$
   - Total Discriminator Loss (L_{D}):
     $L_{\text{D}} = L_{\text{real}} + L_{\text{fake}} + \lambda \cdot L_{\text{class}}$
2. **Generator Loss**:
   - Fake Sample Loss (L_{fake}):
     $L_{\text{fake}} = -\frac{1}{m}\sum_{i=1}^{m}\log D(G(z_i, c_i))$
   - Class Loss (L_{class}):
     $L_{\text{class}} = -\frac{1}{m}\sum_{i=1}^{m}\log C(c_i|G(z_i, c_i))$
   - Total Generator Loss (L_{G}):
     $L_{\text{G}} = L_{\text{fake}} + \lambda \cdot L_{\text{class}}$

Where:

- $C(y|x)$ represents the auxiliary classifier's output for label y given sample x.
- m is the batch size.
- $\log$ represents the natural logarithm.
- λ is a hyperparameter controlling the importance of the class loss relative to the fake sample loss.



Conditional GAN
(Mirza & Osindero, 2014)

Semi-Supervised GAN
(Odena, 2016; Salimans, et al., 2016)

InfoGAN
(Chen, et al., 2016)

AC-GAN
(Present Work)

In a Conditional Generative Adversarial Network (cGAN) and an Auxiliary Classifier Generative Adversarial Network (ACGAN), the loss functions are similar but with some differences due to the presence of an auxiliary classifier in the discriminator of ACGAN. Let's break down the loss functions for both:

1.Conditional Generative Adversarial Network (cGAN):

1. **Discriminator Loss**:
    1. The discriminator's goal is to distinguish between real and fake samples, conditioned on their respective class labels.
    2. The discriminator loss is the sum of two terms:
        1. Real Sample Loss: Measures how well the discriminator classifies real samples. It is typically calculated using binary cross-entropy loss.
        2. Fake Sample Loss: Measures how well the discriminator classifies fake samples generated by the generator. Again, it is typically calculated using binary cross-entropy loss.
    3. The total discriminator loss is the sum of the real and fake sample losses.
2. **Generator Loss**:
    1. The generator's goal is to generate realistic samples that can fool the discriminator, conditioned on specific class labels.
    2. The generator loss is the fake sample loss, which measures how well the generator's output is classified by the discriminator as real. This loss encourages the generator to produce samples that are indistinguishable from real samples.

Auxiliary Classifier Generative Adversarial Network (ACGAN):

•**Discriminator Loss**:

- In addition to distinguishing between real and fake samples, the discriminator in ACGAN also predicts the class labels of the samples.
- The discriminator loss is the sum of three terms:
    - Real Sample Loss: Measures how well the discriminator classifies real samples as real. It is typically calculated using binary cross-entropy loss.
    - Fake Sample Loss: Measures how well the discriminator classifies fake samples generated by the generator as fake. It is also calculated using binary cross-entropy loss.
    - Class Loss: Measures how well the discriminator predicts the class labels of both real and fake samples. It is calculated using categorical cross-entropy loss.
- The total discriminator loss is the sum of the real sample loss, fake sample loss, and class loss.

•**Generator Loss**:

- The generator's goal remains the same as in cGAN: to generate realistic samples that can fool the discriminator.
- The generator loss is the sum of two terms:
    - Fake Sample Loss: Measures how well the generator's output is classified by the discriminator as real. This loss encourages the generator to produce samples that are indistinguishable from real samples.
    - Class Loss: Measures how well the discriminator predicts the class labels of the fake samples. It encourages the generator to produce samples that belong to specific classes.
- The total generator loss is the sum of the fake sample loss and the class loss.

Conditional GAN (cGAN):

1. **Discriminator Loss:**
   - Real Sample Loss (L_{real}):
     $L_{\text{real}} = -\frac{1}{m}\sum_{i=1}^{m} \log D(x_i, y_i)$
   - Fake Sample Loss (L_{fake}):
     $L_{\text{fake}} = -\frac{1}{m}\sum_{i=1}^{m} \log(1 - D(G(z_i, c_i), c_i))$
   - Total Discriminator Loss (L_{D}):
     $L_{\text{D}} = L_{\text{real}} + L_{\text{fake}}$
2. **Generator Loss:**
   - Fake Sample Loss (L_{fake}):
     $L_{\text{fake}} = -\frac{1}{m}\sum_{i=1}^{m} \log D(G(z_i, c_i), c_i)$
   - Total Generator Loss (L_{G}):
     $L_{\text{G}} = L_{\text{fake}}$

Where:

- $D(x, y)$ represents the discriminator's output for sample x given label y.
- $G(z, c)$ represents the generator's output for noise z and label c.

Conditional GAN

- In a conditional GAN, both the generator and discriminator are conditioned on additional information, usually in the form of labels or some other auxiliary information.
- This conditioning allows for more control over the generation process and enables the generation of samples with specific characteristics or attributes.

Auxiliary Classifier GAN (ACGAN):

1. **Discriminator Loss:**
   - Real Sample Loss (L_{real}):
     $L_{\text{real}} = -\frac{1}{m}\sum_{i=1}^{m} \log D(x_i)$
   - Fake Sample Loss (L_{fake}):
     $L_{\text{fake}} = -\frac{1}{m}\sum_{i=1}^{m} \log(1 - D(G(z_i, c_i)))$
   - Class Loss (L_{class}):
     $L_{\text{class}} = -\frac{1}{m}\sum_{i=1}^{m} \log C(y_i|x_i)$
   - Total Discriminator Loss (L_{D}):
     $L_{\text{D}} = L_{\text{real}} + L_{\text{fake}} + \lambda \cdot L_{\text{class}}$
2. **Generator Loss:**
   - Fake Sample Loss (L_{fake}):
     $L_{\text{fake}} = -\frac{1}{m}\sum_{i=1}^{m} \log D(G(z_i, c_i))$
   - Class Loss (L_{class}):
     $L_{\text{class}} = -\frac{1}{m}\sum_{i=1}^{m} \log C(c_i|G(z_i, c_i))$
   - Total Generator Loss (L_{G}):
     $L_{\text{G}} = L_{\text{fake}} + \lambda \cdot L_{\text{class}}$

Where:

- $C(y|x)$ represents the auxiliary classifier's output for label y given sample x.
- m is the batch size.
- $\log$ represents the natural logarithm.
- λ is a hyperparameter controlling the importance of the class loss relative to the fake sample loss.

Here's how a works:
1.Conditional Input: In addition to random noise, the generator is provided with conditional information, such as class labels, attributes, or any other relevant information that describes the desired characteristics of the generated samples.

2.Generator: The generator takes both the random noise and the conditional input and generates samples that are conditioned on this input. The goal of the generator is to produce realistic samples that not only resemble real data but also satisfy the conditioning information.

3.Discriminator: The discriminator also receives conditional information along with the samples, whether real or generated. It learns to classify samples as real or fake based on both the sample itself and the conditional information.

4.Training: During training, the generator and discriminator are trained in a adversarial manner. The generator tries to produce samples that fool the discriminator into classifying them as real, while the discriminator aims to correctly classify real samples as real and generated samples as fake, considering the conditional information.

5.Loss Function: The loss function used in a cGAN typically consists of two components: the standard GAN loss, which encourages the generator to produce realistic samples, and an additional term that incorporates the conditional information. This additional term penalizes the generator and discriminator if the conditional information is not satisfied.

Conditional GAN

Generator::

•With conditioning, the generator's task becomes more specific. Instead of simply producing random samples, it now has to generate samples that not only resemble real data but also satisfy the provided conditional information.

•The conditional information acts as a guide for the generator, **influencing the features and characteristics of the generated samples**.

•For example, if the conditional information is class labels in an image generation task, the generator learns to produce images corresponding to each class label.

•The generator incorporates the conditional information into its architecture, usually by concatenating or injecting it into the network's input layers.

•This allows the generator to learn the relationship between the conditional information and the generated samples.

Conditional GAN

Discriminator:

•Similar to the generator, the discriminator is also conditioned on additional information during training and inference.

•Conditioning alters the discrimination task by instead of solely judging the realism of samples, the discriminator must consider whether the generated samples match the provided conditional information.
•The discriminator learns to discriminate between real and fake samples while taking the conditional information into account.

•It effectively learns to distinguish between samples that not only look real but also align with the conditioning information.

•The conditional information is typically concatenated with the input samples or injected at various layers of the discriminator network to enable it to consider this information during classification.

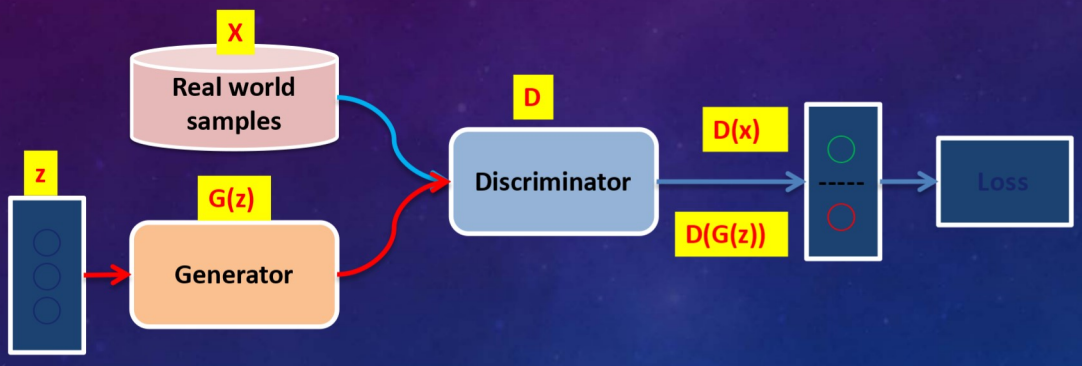
What will happen if we do not give the label to the discriminator??

Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

- **Dr. Chandra Sekhar V**

GAN (Ian Goodfellow et al., NIPS 2014)

Overview



Credit(modified from) https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

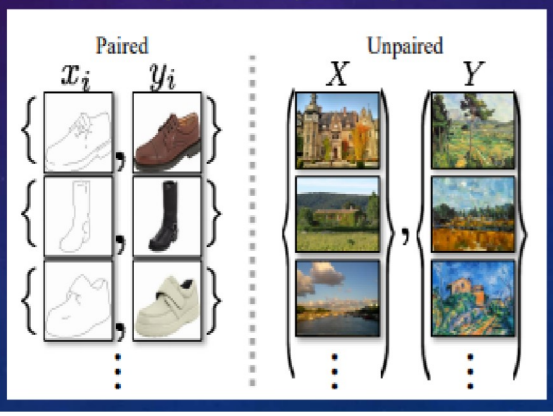
CycleGAN

Cycle GAN is used to transfer characteristic of one image to another or can map the distribution of images to another.

In CycleGAN we treat the problem as an image reconstruction problem. We first take an image input (x) and using the generator G to convert into the reconstructed image.

Then we reverse this process from reconstructed image to original image using a generator F. Then we calculate the mean squared error loss between real and reconstructed image.

The most important feature of this cycle_GAN is that it can do this image translation on an unpaired image where there is no relation exists between the input image and output image.



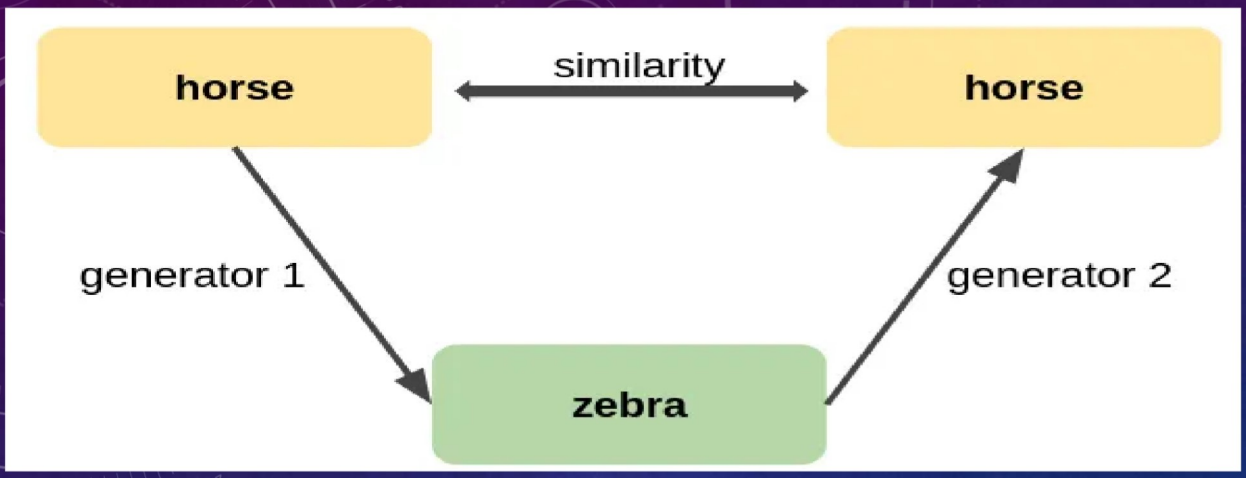
CycleGAN

CycleGAN

Generative Adversarial Models (GANs) are composed of 2 neural networks: a generator and a discriminator. A CycleGAN is composed of 2 GANs, making it a total of 2 generators and 2 discriminators.

- Given 2 sets of different images, horses and zebras for example, one generator transform horses into zebras and the other transform zebras into horses.
- During the training phase, the discriminators are here to check if images computed by generators seem real or fake. Through this process, generators can become better with the feedback of their respective discriminators.
- In the case of CycleGAN, a generator gets an additional feedback from the other generator. This feedback ensure that an image generated by a generator is cycle consistent, meaning that applying consecutively both generators on an image should yield a similar image.

CycleGAN



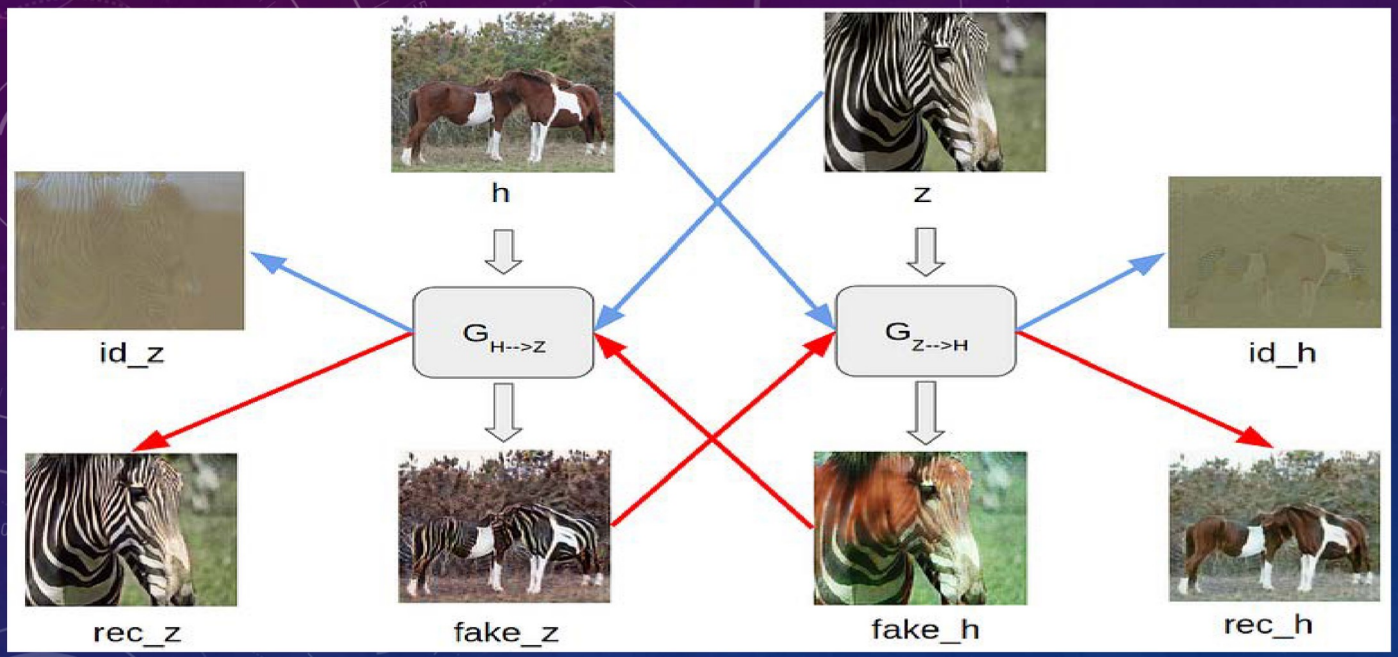
Something very convenient about training a CycleGAN is that we don't need to label its training samples (data are unpaired).

CycleGAN

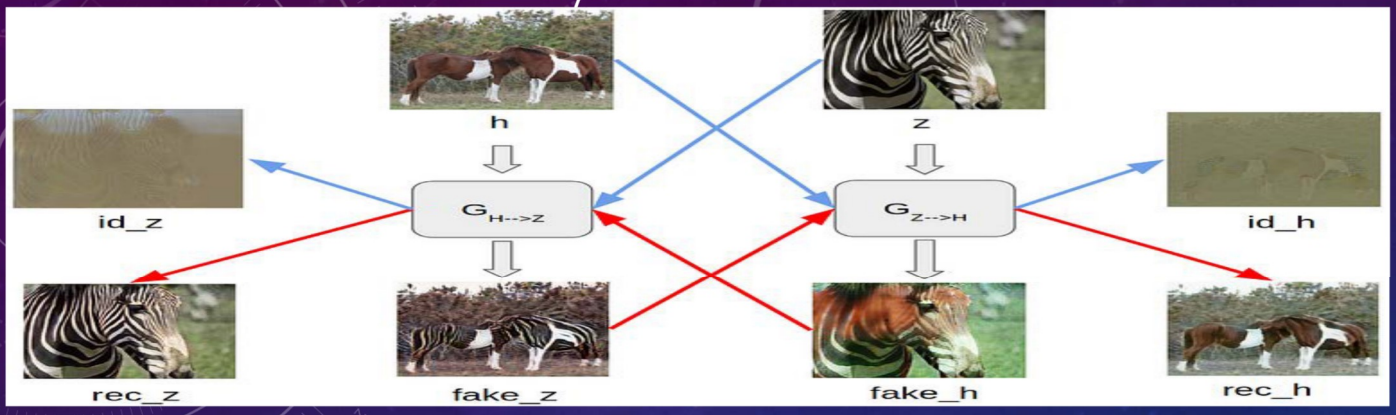
Data:

One training sample of the CycleGAN is formed by a random picture of horse and a random picture of a zebra.

CycleGAN



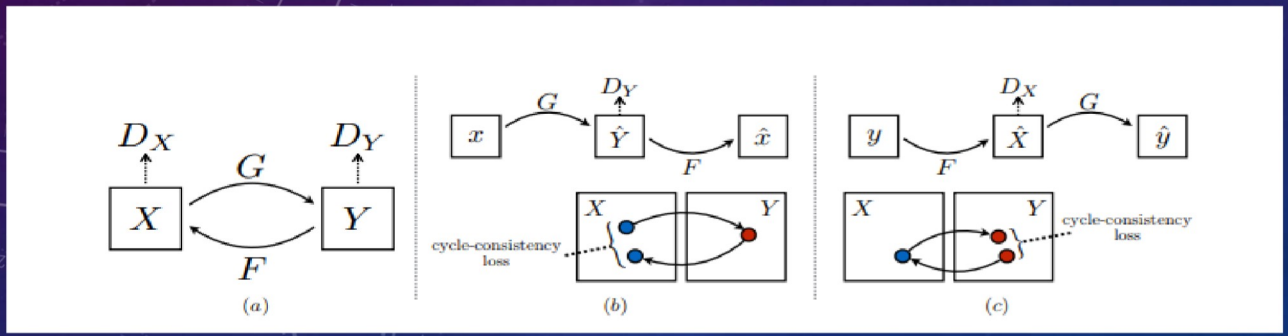
CycleGAN



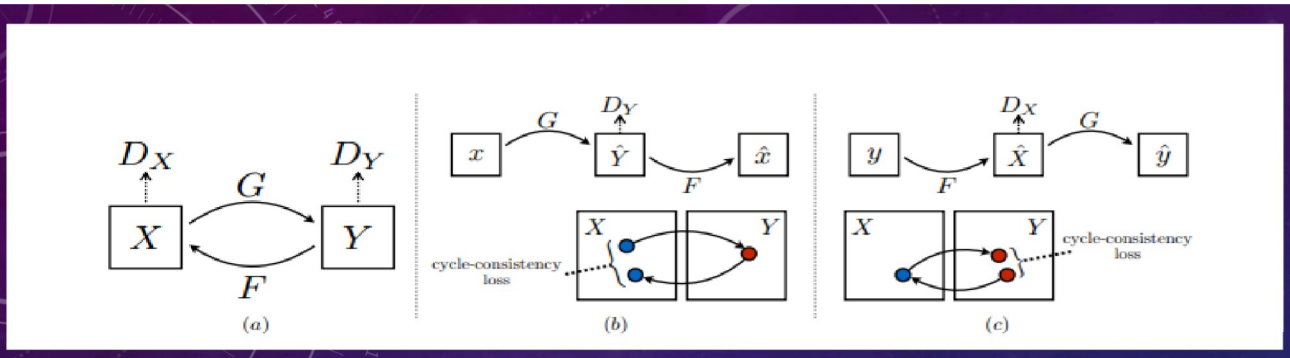
G{H->Z} is the generator that transform horse images into zebra images and G{Z->H} is the generator that transform zebra images into horse images.

Like all the adversarial network CycleGAN also has two parts Generator and Discriminator.

The job of generator to produce the samples from the desired distribution and the job of discriminator is to figure out the sample is from actual distribution (real) or from the one that are generated by generator (fake).



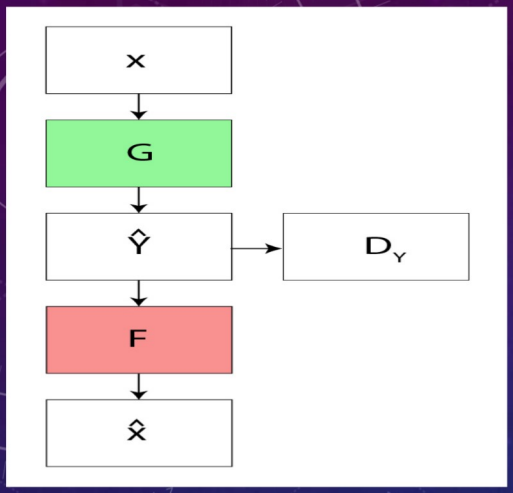
$$G : X \rightarrow Y$$
$$F : Y \rightarrow X$$



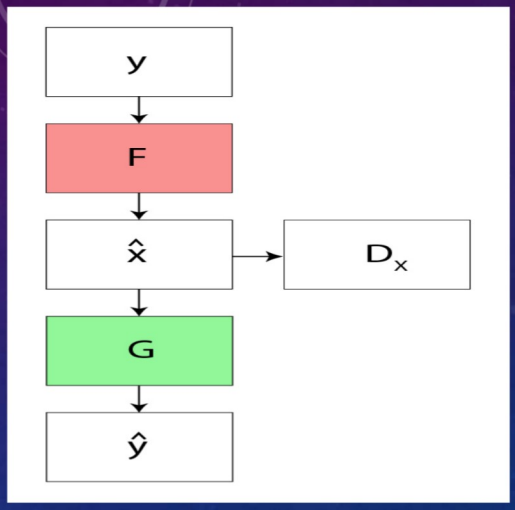
Dx : distinguish G(X)(Generated Output) from Y (real Output)
Dy : distinguish F(Y)(Generated Inverse Output) from X (Input distribution)

To further regularize the mappings, the authors used two more loss function in addition to adversarial loss .
The forward cycle consistency loss and the backward cycle consistency loss .

The forward cycle consistency loss refines the cycle :



The backward cycle consistency loss refines the cycle:



$$x --> G(x) --> F(G(x)) \approx x \qquad y --> F(y) --> G(F(y)) \approx y$$

Generator-A Composite Model (BtoA or Zebra to Horse)

The inputs, transformations, and outputs of the model are as follows:

- **Adversarial Loss: Domain-B** -> Generator-A -> Domain-A -> Discriminator-A -> [real/fake]
- **Identity Loss**: Domain-A -> Generator-A -> Domain-A
- **Forward Cycle Loss**: Domain-B -> Generator-A -> Domain-A -> Generator-B -> Domain-B
- **Backward Cycle Loss**: Domain-A -> Generator-B -> Domain-B -> Generator-A -> Domain-A

We can summarize the inputs and outputs as:

- **Inputs**: Domain-B, Domain-A
- **Outputs**: Real, Domain-A, Domain-B, Domain-A

Generator-B Composite Model (AtoB or Horse to Zebra)

The inputs, transformations, and outputs of the model are as follows:

- **Adversarial Loss**: Domain-A -> Generator-B -> Domain-B -> Discriminator-B -> [real/fake]
- **Identity Loss**: Domain-B -> Generator-B -> Domain-B
- **Forward Cycle Loss**: Domain-A -> Generator-B -> Domain-B -> Generator-A -> Domain-A
- **Backward Cycle Loss**: Domain-B -> Generator-A -> Domain-A -> Generator-B -> Domain-B

We can summarize the inputs and outputs as:

- **Inputs**: Domain-A, Domain-B
- **Outputs**: Real, Domain-B, Domain-A, Domain-B

CycleGAN

Given one training sample (h, z), this is how weights of the CycleGAN is updated by this sample.

- From h, G{H->Z} generates fake_z a fake zebra image. From z, G{H->Z} generates id_z that should be identical to z.
- From z, G{Z->H} generates fake_h a fake horse image. From h, G{Z->H} generates id_h that should be identical to h.
- From fake_z, G{Z->H} generates rec_h a reconstructed image of h that should be similar to h.
- From fake_h, G{H->Z} generates rec_z a reconstructed image of z that should be similar to z.

CycleGAN

Then, weights of the generators are updated by minimizing following losses. In the case of G{*H->Z*}, it can be written as:

$$MSE(D_z(fake_z), 1) = \frac{1}{30}\sum_{i,j}^{30}(D_z(fake_z)(i,j) - 1)^2$$

where D_z(fake_z)(i,j) is the scalar value at coordinate (i,j) of D_z output tensor.

- Mean Squared Error (MSE) between $D\_z(fake\_z)$ and 1. Namely least-squares loss in the original paper. D_z is the discriminator associated to G{*H->Z*}, it outputs 1 if the input — $fake\_z$ here — seems like a real zebra else 0.

CycleGAN

$$MAE(rec_h, h) = \frac{1}{N}\sum_{i,j}^{N}|(rec_h(i,j) - h(i,j)|$$

pixel to pixel absolute difference of the two images.

- Mean Absolute Error (MAE) between rec_h and h. Namely cycle consistency loss in the original paper. This loss is given a lot of importance and updates 10 times as much the generator weights than the MSE loss.

CycleGAN

$$MAE(id_h, h) = \frac{1}{N} \sum_{i,j}^{N} |(id_h(i,j) - h(i,j)|$$

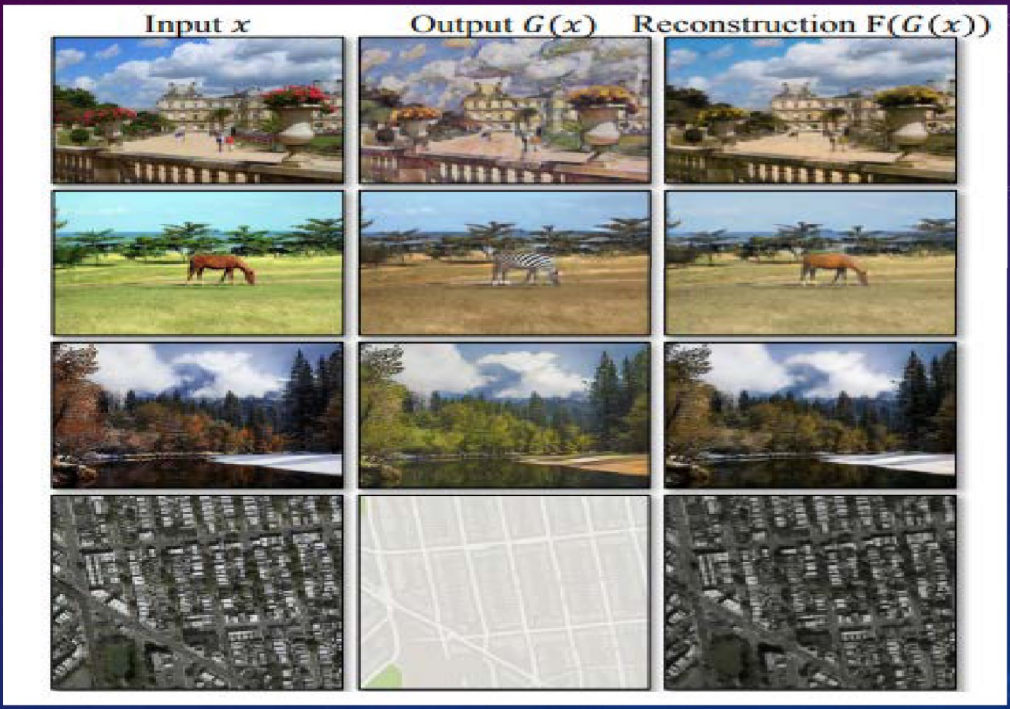
pixel to pixel absolute difference of the two images.

- Mean Absolute Error (MAE) between id_h and h. Namely identity loss in the original paper. It is not necessary to include this loss, but it generally help get better results.

CycleGAN

Afterwards, weights of discriminators are updated by minimizing the following loss. In the case of D_z, it can be written as:

$$\frac{1}{2}[MSE(D_z(fake_z), 0) + MSE(D_z(z), 1)] = \frac{1}{2 * 30}[\sum_{i,j}^{30}(D_z(fake_z)(i,j))^2 + \sum_{i,j}^{30}(D_z(fake_z)(i,j) - 1)^2]$$



Cost Function:

Adversarial Loss: We apply adversarial loss to both our mappings of generators and discriminators. This adversary loss is written as :

$$Loss_{advers}\left(G, D_y, X, Y\right) = \frac{1}{m}\sum\left(1 - D_y\left(G\left(x\right)\right)\right)^2$$

$$Loss_{advers}\left(F, D_x, Y, X\right) = \frac{1}{m}\sum\left(1 - D_x\left(F\left(y\right)\right)\right)^2$$

The model architecture is comprised of two generator models: one generator (Generator-A) for generating images for the first domain (Domain-A) and the second generator (Generator-B) for generating images for the second domain (Domain-B).

Generator-A -> Domain-A
Generator-B -> Domain-B

The generator models perform image translation, meaning that the image generation process is conditional on an input image, specifically an image from the other domain.

Generator-A takes an image from Domain-B as input and Generator-B takes an image from Domain-A as input.

Domain-B -> Generator-A -> Domain-A
Domain-A -> Generator-B -> Domain-B

- Each generator has a corresponding discriminator model. The first discriminator model (Discriminator-A) takes real images from Domain-A and generated images from Generator-A and predicts whether they are real or fake.

- The second discriminator model (Discriminator-B) takes real images from Domain-B and generated images from Generator-B and predicts whether they are real or fake.

  •Domain-A -> Discriminator-A -> [Real/Fake]
  •Domain-B -> Generator-A -> Discriminator-A -> [Real/Fake]
  •Domain-B -> Discriminator-B -> [Real/Fake]
  •Domain-A -> Generator-B -> Discriminator-B -> [Real/Fake]

- The discriminator and generator models are trained in an adversarial zero-sum process, like normal GAN models.
- The generators learn to better fool the discriminators and the discriminator learn to better detect fake images. Together, the models find an equilibrium during the training process.
- Additionally, the generator models are regularized to not just create new images in the target domain, but instead translate more reconstructed versions of the input images from the source domain.
- This is achieved by using generated images as input to the corresponding generator model and comparing the output image to the original images.
- Passing an image through both generators is called a cycle. Together, each pair of generator models are trained to better reproduce the original source image, referred to as cycle consistency.

Domain-B -> Generator-A -> Domain-A -> Generator-B -> Domain-B
Domain-A -> Generator-B -> Domain-B -> Generator-A -> Domain-A

There is one further element to the architecture, referred to as the identity mapping. This is where a generator is provided with images as input from the target domain and is expected to generate the same image without change. This addition to the architecture is optional, although results in a better matching of the color profile of the input image.

Domain-A -> Generator-A -> Domain-A
Domain-B -> Generator-B -> Domain-B

- The discriminator models are trained directly on real and generated images, whereas the generator models are not.
- Instead, the generator models are trained via their related discriminator models. Specifically, they are updated to minimize the loss predicted by the discriminator for generated images marked as “real“, called adversarial loss. As such, they are encouraged to generate images that better fit into the target domain.
- The generator models are also updated based on how effective they are at the regeneration of a source image when used with the other generator model, called cycle loss.
- Finally, a generator model is expected to output an image without translation when provided an example from the target domain, called identity loss.
- Altogether, each generator model is optimized via the combination of four outputs with four loss functions:

- Adversarial loss (L2 or mean squared error).
- Identity loss (L1 or mean absolute error).
- Forward cycle loss (L1 or mean absolute error).
- Backward cycle loss (L1 or mean absolute error).

This can be achieved by defining a composite model used to train each generator model that is responsible for only updating the weights of that generator model, although it is required to share the weights with the related discriminator model and the other generator model.

This is implemented in the define_composite_model() function below that takes a defined generator model (g_model_1) as well as the defined discriminator model for the generator models output (d_model) and the other generator model (g_model_2).

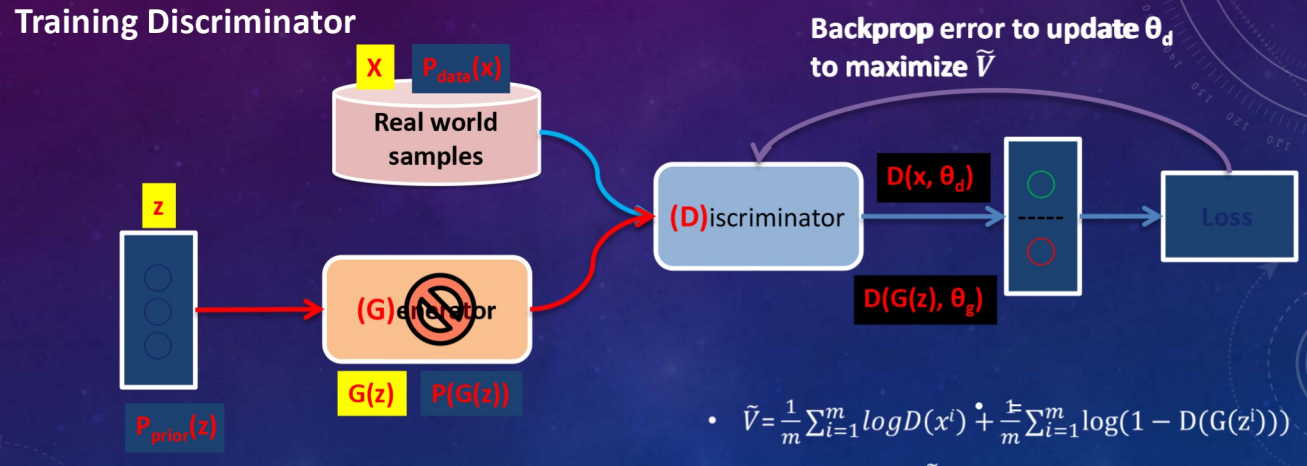
The weights of the other models are marked as not trainable as we are only interested in updating the first generator model, i.e. the focus of this composite model.

The discriminator is connected to the output of the generator in order to classify generated images as real or fake. A second input for the composite model is defined as an image from the target domain (instead of the source domain), which the generator is expected to output without translation for the identity mapping.

Next, forward cycle loss involves connecting the output of the generator to the other generator, which will reconstruct the source image.

- Finally, the backward cycle loss involves the image from the target domain used for the identity mapping that is also passed through the other generator whose output is connected to our main generator as input and outputs a reconstructed version of that image from the target domain.

GAN (Ian Goodfellow et al., NIPS 2014)



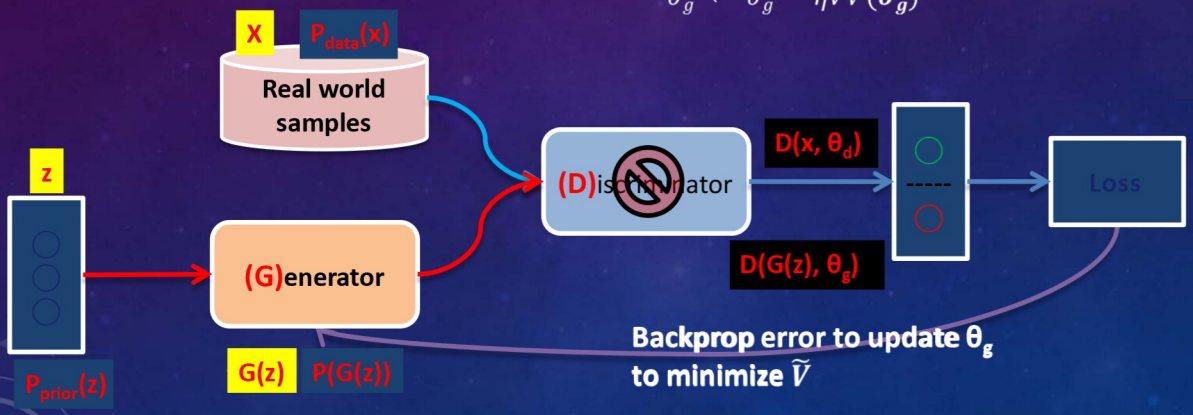
- $\tilde{V} = \frac{1}{m}\sum_{i=1}^{m} logD(x^i) + \frac{1}{m}\sum_{i=1}^{m}\log(1 - D(G(z^i)))$
- $\theta_d \leftarrow \theta_d + \eta\nabla\tilde{V}(\boldsymbol{\theta_d})$

Credit(modified from) https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

GAN (Ian Goodfellow et al., NIPS 2014)

Training Generator

- $\tilde{V} = \frac{1}{m}\sum_{i=1}^{m} logD(x^i) + \frac{1}{m}\sum_{i=1}^{m}\log(1 - D(G(z^i)))$
- $\theta_g \leftarrow \theta_g - \eta\nabla\tilde{V}(\boldsymbol{\theta_g})$



Credit(modified from) https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

GAN (Ian Goodfellow et al., NIPS 2014)

- Objective

Maximize D — Value of — Expectation — fake

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim Pdata(x)}[\log D(x)] + \mathbb{E}_{z\sim pz(z)}[\log\left(1 - D\left(G(z)\right)\right)]$$

Minimize G — Distribution of real data — Prob. Of D(*real*) — Distribution of generated samples — Prob. Of D(*fake*)

CycleGAN – Objective Function

- Adversarial loss (for G: X -> Y)
- $$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))].$$

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].$$

- **Full objective**

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F),$$

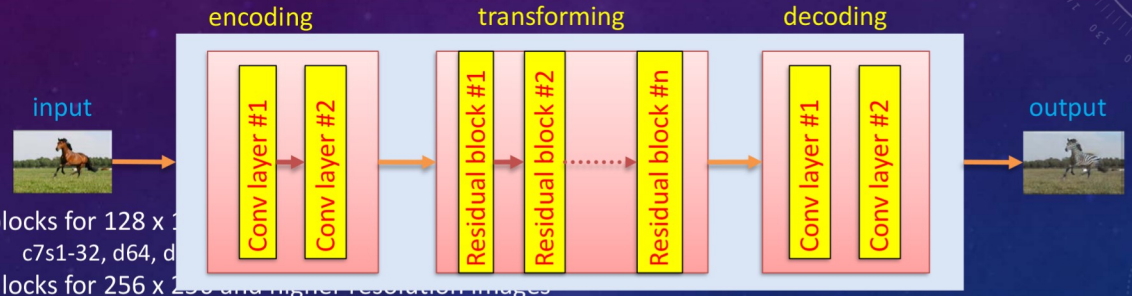
- **Aim to solve**

$$G^*, F^* = \arg\min_{G,F} \max_{D_x, D_Y} \mathcal{L}(G, F, D_X, D_Y).$$

Credit: https://arxiv.org/pdf/1703.10593.pdf

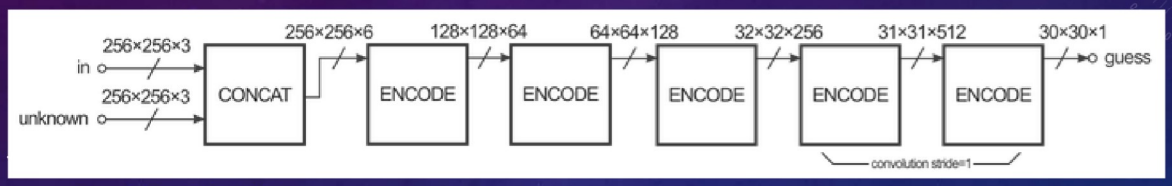
CycleGAN – Implement

Generator network



- 6 blocks for 128 x 1
  - c7s1-32, d64, d
- 9 blocks for 256 x 2[illegible]
  - c7s1-32, d64, d128, R128, R128, R128, R128, R128, R128, R128, R128, R128, u64, u32, c7s1-3

CycleGAN – Implementation

- **Discriminator network**
- 

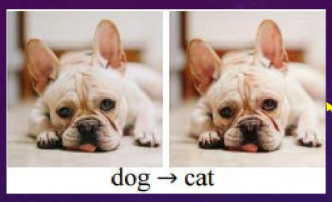
  - c64, c128, c256, c512

Credit: https://affinelayer.com/pix2pix/

CycleGAN – Results

- Applications
- Analysis

CycleGAN – Limitations



Changes in distribution characteristics
(test set vs. training set)

Tasks require geometric changes

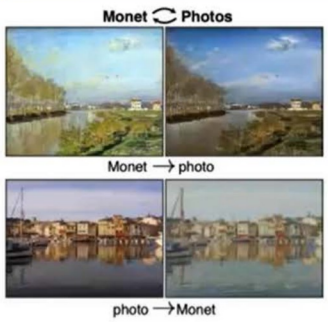
A lingering gap between the results achievable from pix2pix and that achieved by CycleGAN.

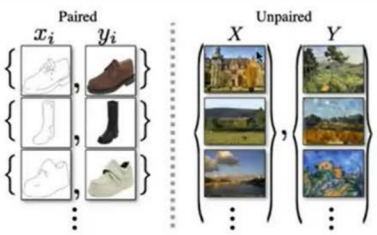
https://qiita.com/itok_msi/items/b6b615bc28b1a720afd7#%E8%BF%BD%E5%8A%A0%E5%AE%9F%E9%A8%93%E7%B5%90%E6%9E%9C20170614%E8%BF%BD%E8%A8%987

Boulder

Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

YouTube Video





$G : X \to Y$

$F : Y \to X$

$F(G(x)) \approx x$ and $G(F(y)) \approx y$

$D_X \to$ aims to discriminate between $\{x\}$ and $\{F(y)\}$

$D_Y \to$ aims to discriminate between $\{y\}$ and $\{G(x)\}$

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))],$$

$\min_G \max_{D_Y} \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y)$

$\min_F \max_{D_X} \mathcal{L}_{\text{GAN}}(F, D_X, Y, X).$

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].$$

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F),$$

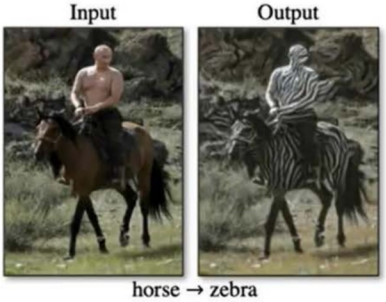
$$G^*, F^* = \arg \min_{G,F} \max_{D_x, D_Y} \mathcal{L}(G, F, D_X, D_Y).$$

Least-squares Loss

$$\min_G \mathbb{E}_{x \sim p_{\text{data}}(x)}[(D(G(x)) - 1)^2]$$

$$\min_D \mathbb{E}_{y \sim p_{\text{data}}(y)}[(D(y) - 1)^2] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[D(G(x))^2]$$

update the discriminators using a history of generated images



Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." *Proceedings of the IEEE international conference on computer vision*. 2017.

Unsupervised Image-to-Image Translation Networks

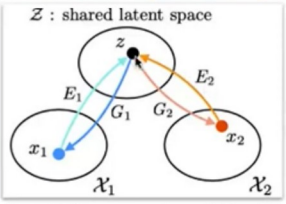
Boulder

UNIT: UNsupervised Image-to-image Translation Networks

Goal: To learn a joint distribution of images in different domains by using images from the marginal distributions in individual domains.

Since an infinite set of possible joint distributions can yield the given marginal distributions, we could infer nothing about the joint distribution from the marginal samples without additional assumptions.

Shared Latent Space Assumption



$z = E_1(x_1) = E_2(x_2)$

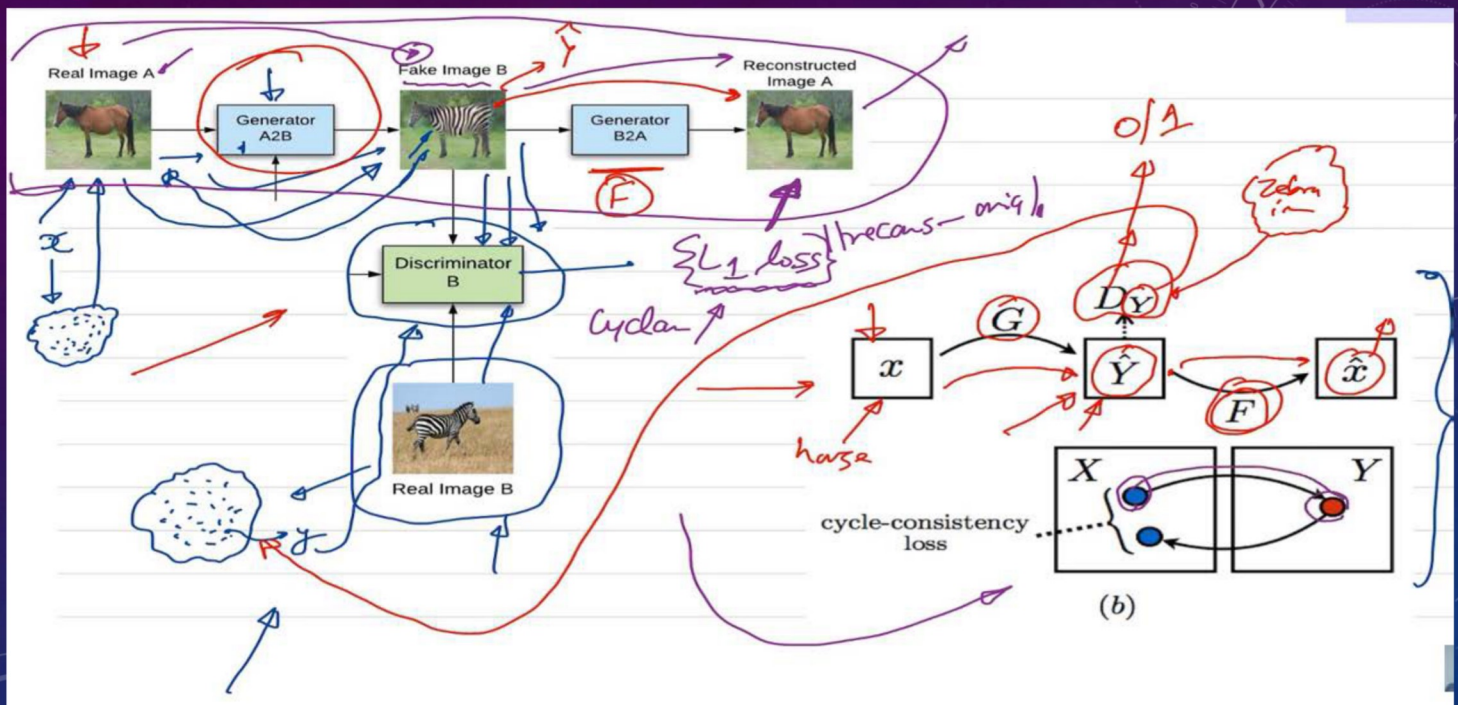
$x_1 = G_1(z) \quad x_2 = G_2(z)$

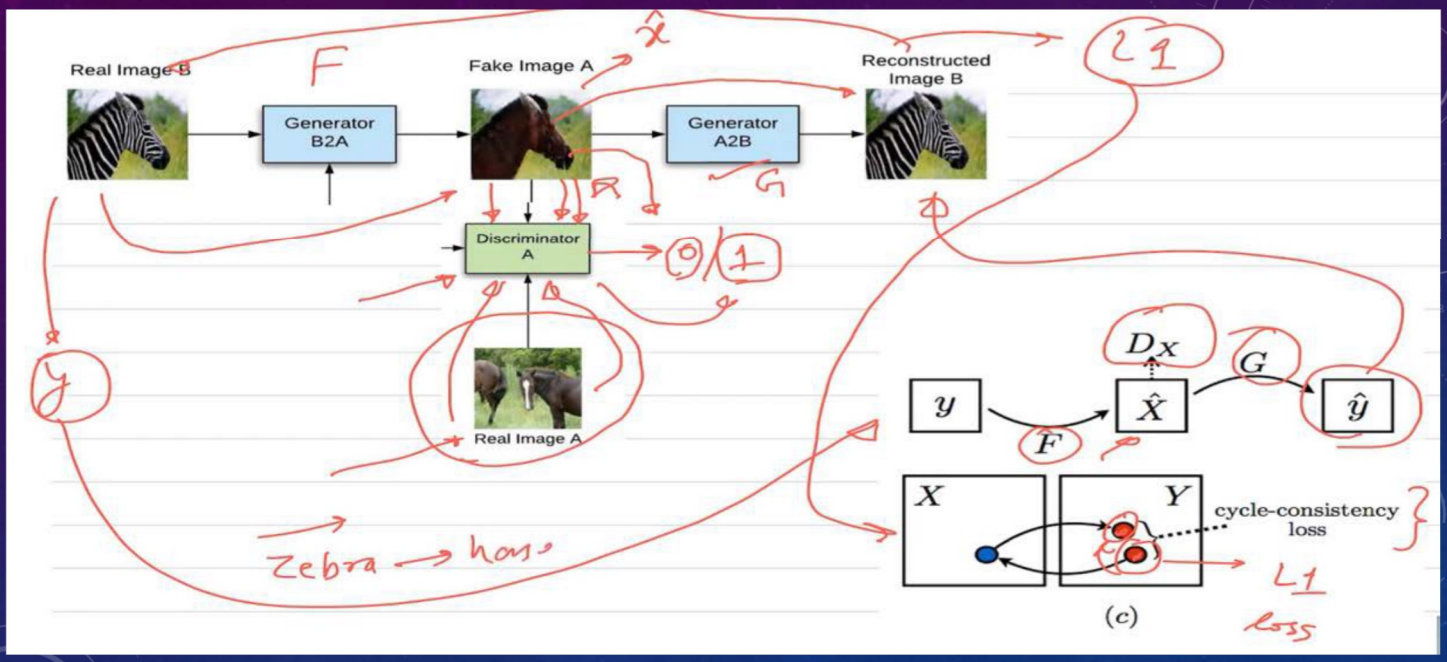
Shared latent space constraint implies cycle-consistency!

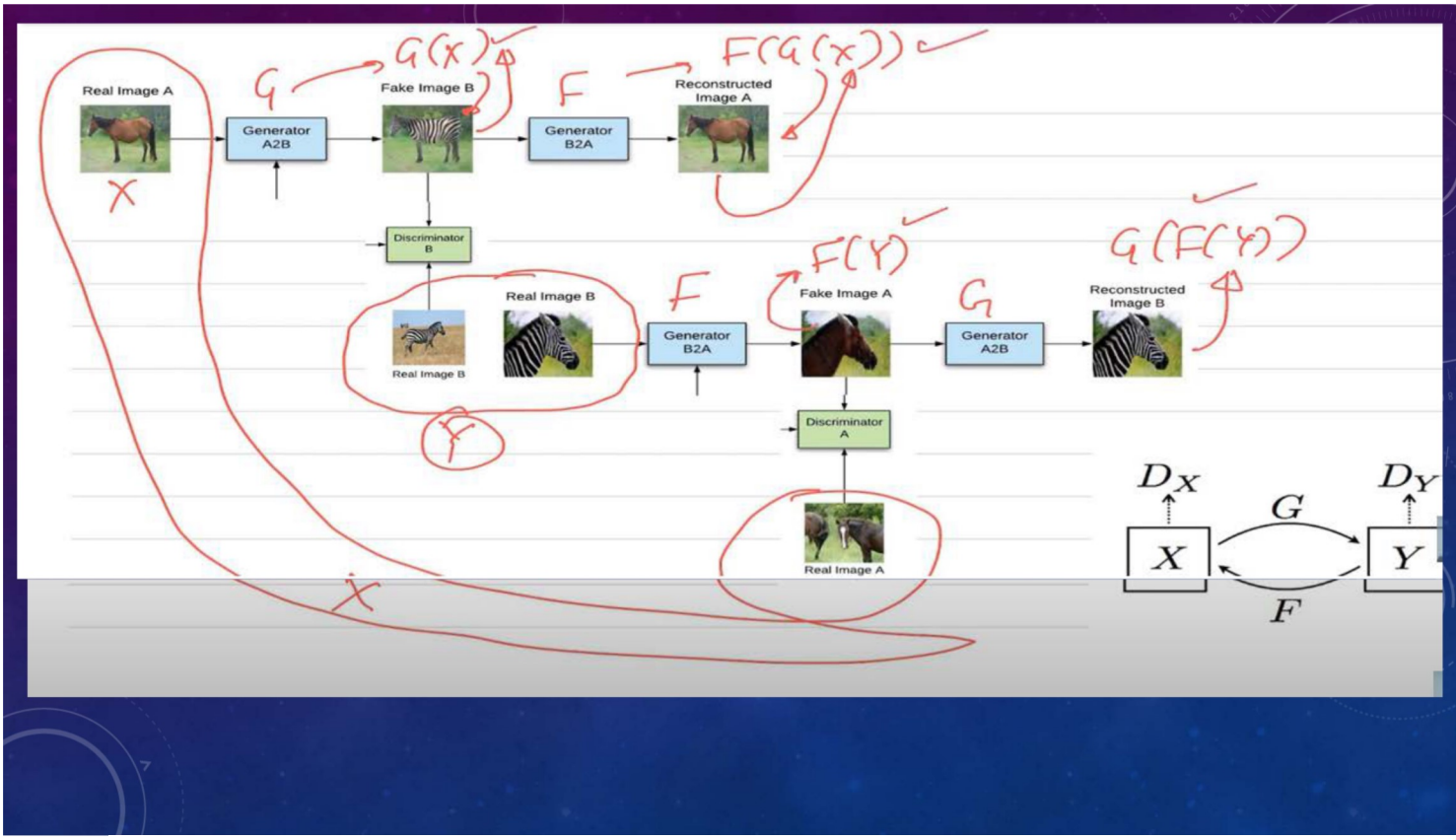
$F^{1\to 2} := G_2 \circ E_1 \qquad F^{2\to 1} := G_1 \circ E_2$

$\{E_1, E_2\} \to$ encoders, $\{G_1, G_2\} \to$ generators, $\{D_1, D_2\} \to$ discriminators

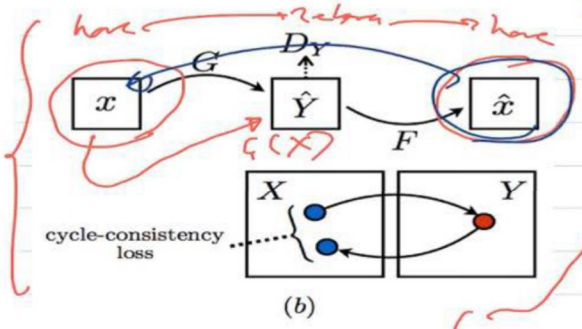
Liu, Ming-Yu, Thomas Breuel, and Jan Kautz. "Unsupervised image-to-image translation networks." *Advances in neural information processing systems*. 2017.




Real Image B
Generator B2A
Fake Image A
Generator A2B
Reconstructed Image B
Discriminator A
Real Image A
D_X
G
y
F
$\hat{X}$
$\hat{y}$
X
Y
cycle-consistency loss
(c)


Real Image A
Generator A2B
Fake Image B
Generator B2A
Reconstructed Image A
Discriminator B
Real Image B
Real Image B
Generator B2A
Fake Image A
Generator A2B
Reconstructed Image B
Discriminator A
Real Image A
D_X
D_Y
G
X
Y
F

G, D horse → zebra GAN



(b)

GAN

{ Adversarial Loss }

GAN

fake sechan

$$L_{GAN_1}(G, D_Y, X, Y) = \{ E_{y \sim p_{data}(y)} \log [D_Y(y)] + E_{x \sim p_{data}(x)} [\log(1 - D_Y(G(x)))] \}$$

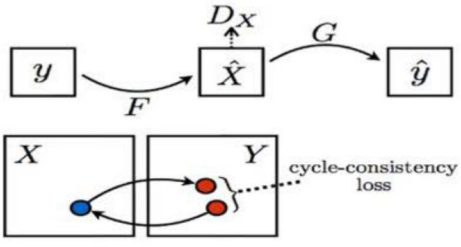
$y \sim p_{data}(y)$: zebra — real sample.

D_Y : zebra

$G(X)$

{ Cycle consistency loss: }

$$L_{cyc_2}(G, F) = E_{x \sim p_{data}(x)} [\| F(G(x)) - x \|_1]$$



(c)

Zebra → horse.

Adversarial Loss ✓

$$L_{GAN_2}(F, D_X, Y, X) = E_{x \sim p_{data}(x)} \log [D_X(x)] + E_{y \sim p_{data}(y)} [\log(1 - D_X(F(y)))]$$

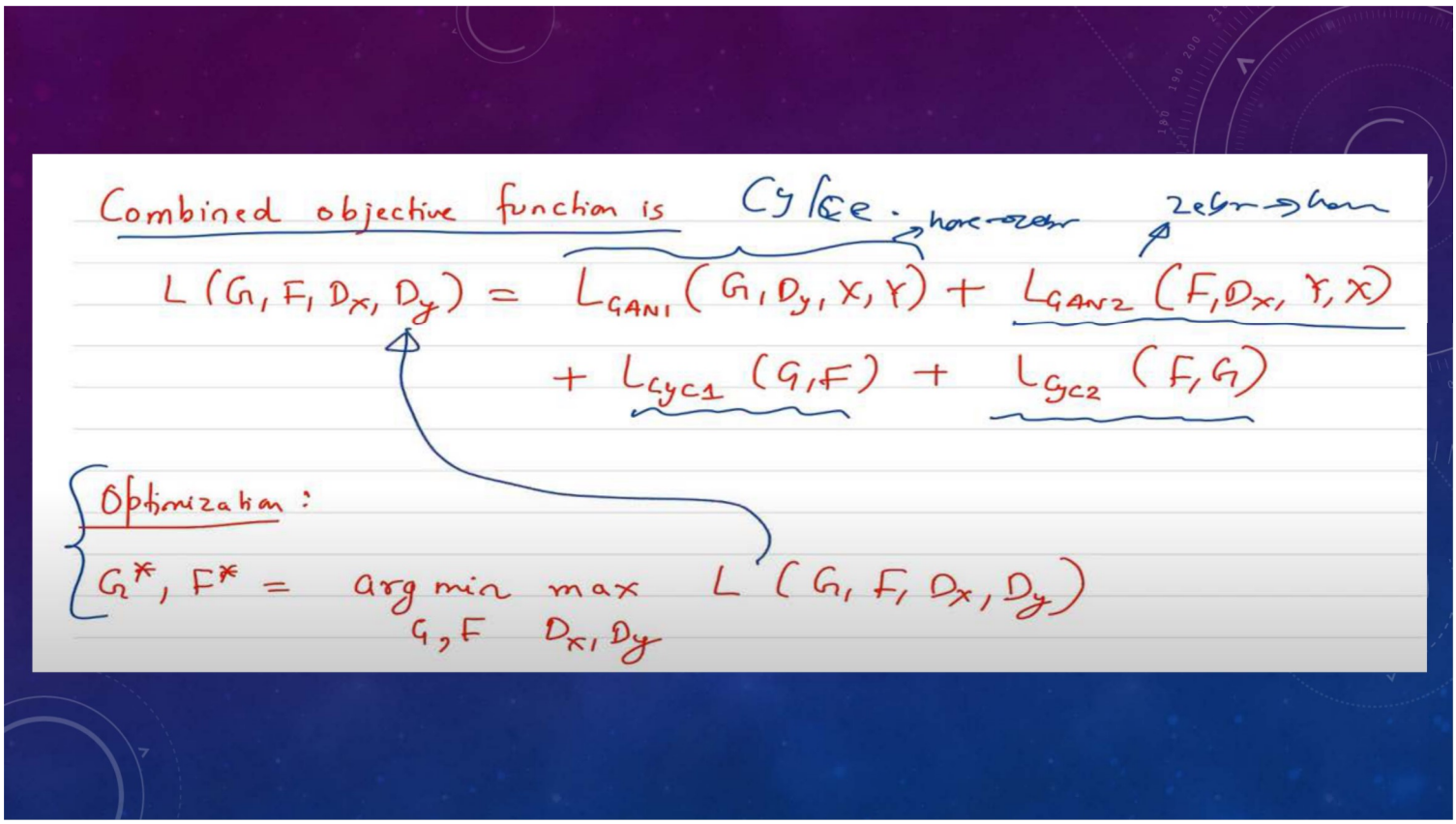
x : horse

F(y) : generated horse

Cycle consistency Loss: ✓

$$L_{cyc_2}(F, G) = E_{y \sim p_{data}(y)} [\| G(F(y)) - y \|_1]$$

G(F(y)) : estimated zebra image



Information Maximizing Generative Adversarial Networks- InfoGAN

Dr. Chandra Sekhar V

Why we need InfoGAN

- For instance, a GAN that is trained to produce fake hand-written digit images may be able to generate very real hand-written digit images, but we have no control over which number it generates.
- InfoGAN solved this problem: the network can learn to produce images with specific categorical features (such as digits 0 to 9) and continuous features (such as the rotational angle of the digits), in an unsupervised manner.
- In addition, because the learning is unsupervised, it is able to find the patterns hidden among the images, and generate images that follow these hidden patterns.

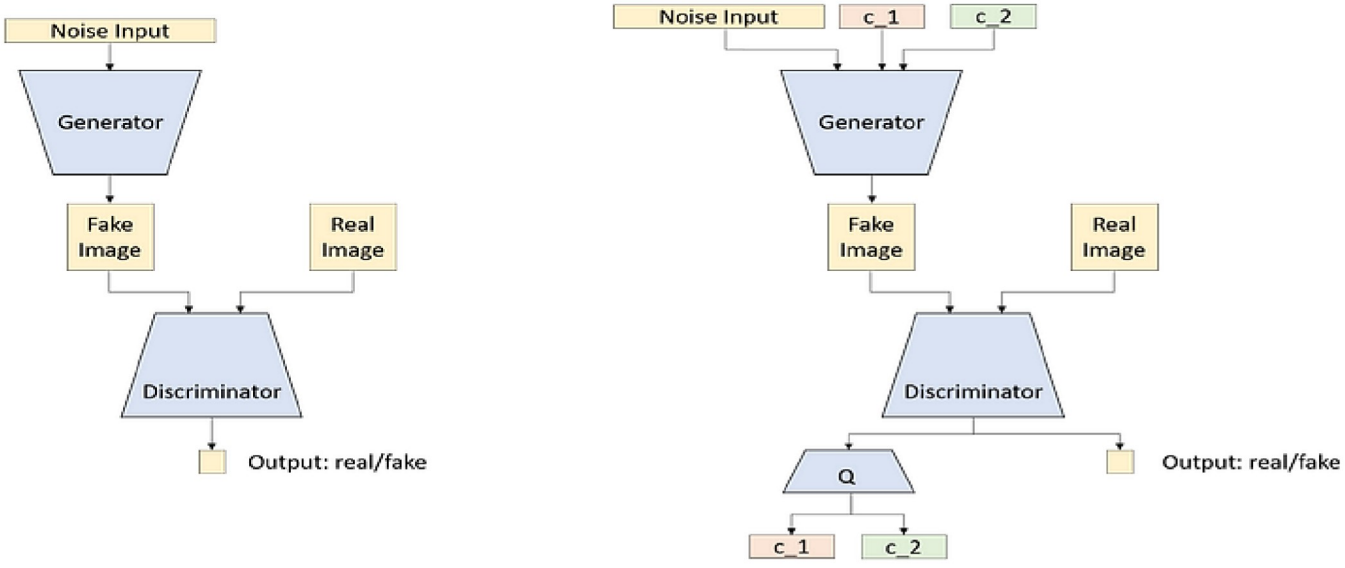
Why we need InfoGAN

- In InfoGAN, to control the types of images produced, we need to feed additional information on the top of random noises to the generator and force it to use the information when making the fake images.
- The additional information we feed should relate to the types of features we want the images to have.
- For example, if we want to produce specific MNIST digits, we need to feed a categorical vector containing integers from 0 to 9; if we want to produce MNIST digits with different rotational angles, we may want to feed float numbers randomly selected between -1 to 1.

Why we need InfoGAN : to produce feature-specific MNIST digits

- Feeding additional information is easy, as we just need to add extra inputs to the generator model.
- But how can we ensure that the generator will use the information instead of completely ignoring it? If we still train the generator simply based on the response of the discriminator, the generator won't use the additional information as the additional information will not help the generator to create more realistic images (it is only helpful to generate specific features of the images).
- Thus, we need to apply extra "penalties" on the generator if it does not use the additional information.
- One way is to add an additional network (often named auxiliary network and denoted as Q) that takes fake images and reproduces the additional information we fed into the generator.
- In this way, the generator is forced to use the additional information as if it doesn't, there is no way that the auxiliary network can correctly reproduce the additional information, and the generator will be "penalized".

Why we need InfoGAN



- InfoGAN is designed to maximize the mutual information between a small subset of the latent variables and the observation.
- A lower bound of the mutual information objective is derived that can be optimized efficiently.
- By doing so, InfoGAN successfully disentangles writing styles from digit shapes on MNIST dataset, and disentangles the visual concepts that include hair styles, presence/absence of eyeglasses, and emotions on the CelebA face dataset.

1.1. MinMax Game Using Mutual Information

In information theory, mutual information between X and Y, I(X, Y), measures the "amount of information" learned from knowledge of random variable Y about the other random variable X.

The mutual information can be expressed as the difference of two entropy terms:

I(X; Y) is the reduction of uncertainty in X when Y is observed.
If X and Y are independent, then I(X; Y) = 0, because knowing one variable reveals nothing about the other.
By contrast, if X and Y are related by a deterministic, invertible function, then maximal mutual information is attained.ac

$$I(X;Y) = H(X) + H(Y) - H(X,Y)$$

Where:

- $I(X;Y)$ represents the mutual information between X and Y.
- $H(X)$ is the entropy of X, representing the amount of uncertainty or randomness in X.
- $H(Y)$ is the entropy of Y, representing the amount of uncertainty or randomness in Y.
- $H(X,Y)$ is the joint entropy of X and Y, representing the amount of uncertainty or randomness in the joint distribution of X and Y.

- It is essentially a measure of the dependence or association between the two variables: higher mutual information indicates greater dependence or shared information, while lower mutual information indicates less dependence or shared information.

- Given any x~P_G(x), we want P_G(c|x) to have a _____ entropy. In other words, the information in the latent code c should not be lost in the generation process.

The adversarial loss in InfoGAN is formulated as:

$$\mathcal{L}_{\text{adv}}(G, D) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

- The first term $\mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)]$ represents the expectation over real data samples x drawn from the true data distribution $p_{\text{data}}(x)$. The discriminator $D(x)$ outputs the probability that x is real. The goal of the generator is to maximize this term, making the discriminator classify real samples as real with high probability.
- The second term $\mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$ represents the expectation over noise samples z drawn from a prior distribution $p_z(z)$. The generator $G(z)$ produces fake samples, and the discriminator $D(G(z))$ outputs the probability that $G(z)$ is real. The goal of the generator is to minimize this term, making the discriminator classify fake samples as real with low probability.

The overall objective is to minimize the discriminative ability of D while maximizing the generative ability of G, resulting in realistic generated samples.

1. **Adversarial Loss**:
   The adversarial loss in InfoGAN is formulated as:

$$\mathcal{L}_{\text{adv}}(G, D) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

   - The first term $\mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)]$ represents the expectation over real data samples x drawn from the true data distribution $p_{\text{data}}(x)$. The discriminator $D(x)$ outputs the probability that x is real. The goal of the generator is to maximize this term, making the discriminator classify real samples as real with high probability.
   - The second term $\mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$ represents the expectation over noise samples z drawn from a prior distribution $p_z(z)$. The generator $G(z)$ produces fake samples, and the discriminator $D(G(z))$ outputs the probability that $G(z)$ is real. The goal of the generator is to minimize this term, making the discriminator classify fake samples as real with low probability.

   The overall objective is to minimize the discriminative ability of D while maximizing the generative ability of G, resulting in realistic generated samples.

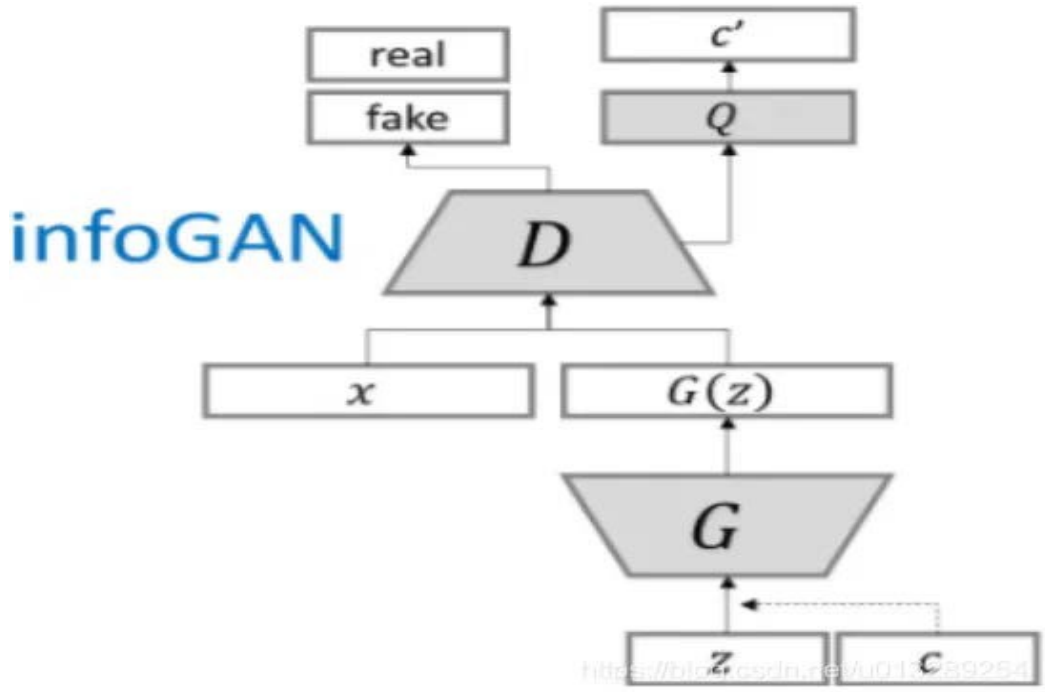
2. **Mutual Information Regularization Term**:
   The mutual information regularization term in InfoGAN is formulated as:

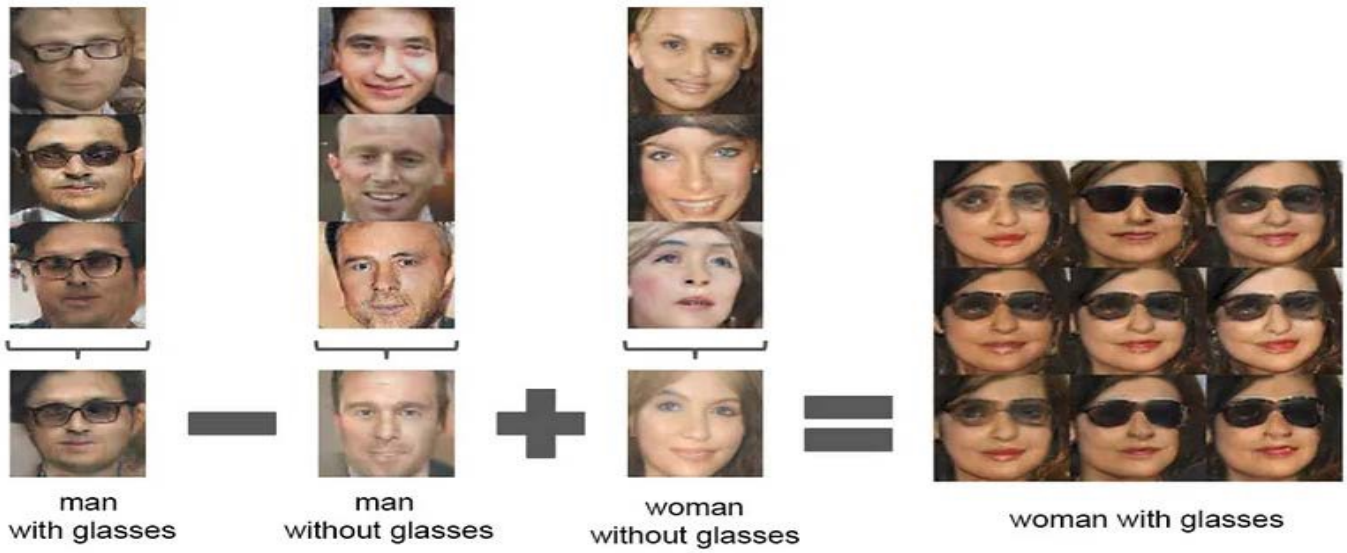
$$\mathcal{L}_I(G, Q) = -\mathbb{E}_{c \sim Q(c|G(z,c))}[\log Q(c|G(z, c))] + \mathbb{E}_{c \sim P(c)}[\log P(c)]$$

   - The first term $-\mathbb{E}_{c \sim Q(c|G(z,c))}[\log Q(c|G(z, c))]$ represents the negative expectation over the latent variables c given the generated samples $G(z, c)$. This term encourages the distribution $Q(c|G(z, c))$ to match the prior distribution of the latent variables $P(c)$. In other words, it encourages the latent variables to be informative about the generated samples.
   - The second term $\mathbb{E}_{c \sim P(c)}[\log P(c)]$ represents the expectation over the prior distribution of the latent variables $P(c)$. This term provides a regularization force, encouraging the latent variables to follow a certain prior distribution.

   By combining these two terms, the mutual information regularization term encourages the learned latent representations to capture meaningful and disentangled information about specific attributes or features of the data.

Overall, the total loss function of InfoGAN combines the adversarial loss and the mutual information regularization term, allowing the model to simultaneously generate realistic samples and learn disentangled representations. Adjusting the hyperparameters (such as λ in the total loss function) allows researchers to control the trade-off between these objectives.





This shows that there are structures in the noise vectors that have meaningful and consistent effects on the generator output

- However, there's no systematic way to find these structures. The process is very manual: 1) generate a bunch of images, 2) find images that have the characteristic you want, 3) average together their noise vectors and hope that it captures the structure of interest.
- In the context of Generative Adversarial Networks (GANs), the idea that representation is entangled refers to the complex relationship between the learned latent space and the generated output.
- InfoGAN tries to solve this problem and provide a disentangled representation.

The idea is to provide a latent code, which has meaningful and consistent effects on the output. For instance, let's say you're working with the MNIST hand-written digit dataset. You know there are 10 digits, so it would be nice if you could use this structure by assigning part of the input to a 10-state discrete variable. The hope is that if you keep the code the same and randomly change the noise, you get variations of the same digit.

InfoGAN

The way InfoGAN approaches this problem is by splitting the Generator input into two parts: the traditional noise vector and a new "latent code" vector. The codes are then made meaningful by maximizing the Mutual Information between the code and the generator output.

This framework is implemented by merely adding a regularization term (red box) to the the original GAN's objective function.

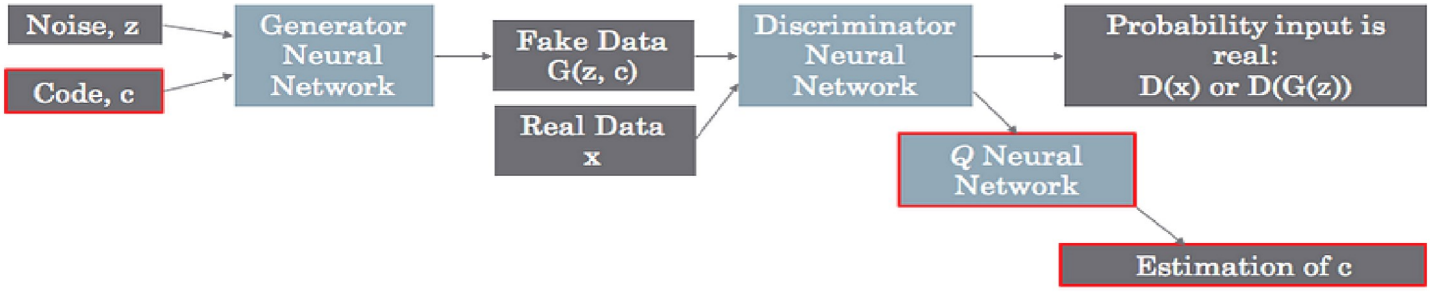
$$\min_G \max_D V_I(D,G) = V(D,G) - \lambda I(c; G(z,c))$$

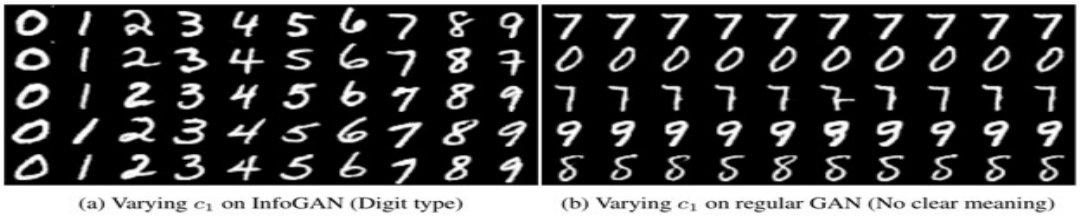
The I(c;G(z,c)) term is the mutual information between the latent code c and the generator output G(z,c).

- The regularizer term above translates to the following process: Sample a value for the latent code c from a prior of your choice; Sample a value for the noise z from a prior of your choice; Generate x = G(c,z); Calculate Q(c|x=G(c,z)).
- The final form of the objective function is then given by this lower-bound approximation to the Mutual Information:

$$\min_{G,Q} \max_{D} V_{\text{InfoGAN}}(D, G, Q) = V(D, G) - \lambda L_I(G, Q)$$

- As mentioned above, there's now a second input to the generator: the latent code.
- The auxiliary distribution introduced in the theory section is modeled by another neural network, which really is just a fully connected layer tacked onto the last representation layer of the discriminator.
- The Q network is essentially trying to predict what the code is. This is only used when feeding in fake input, since that's the only time the code is known.





(a) Varying c_1 on InfoGAN (Digit type)

(b) Varying c_1 on regular GAN (No clear meaning)

(c) Varying c_2 from −2 to 2 on InfoGAN (Rotation)

(d) Varying c_3 from −2 to 2 on InfoGAN (Width)

InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets

Boulder

Disentangled Representations
- writing styles from digit shapes on MNIST
- pose from lighting of 3D rendered images
- background digits from the central digit on SVHN
- hair styles, presence/absence of eyeglasses and emotions on CelebA

Regular GAN

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim P_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim \text{noise}}[\log(1 - D(G(z)))]$$

InfoGAN

$$\min_G \max_D V_I(D,G) = V(D,G) - \lambda I(c; G(z,c))$$

$z \rightarrow$ source of incompressible noise

$c = (c_1, \ldots, c_L) \rightarrow$ latent code (semantic features of data)

In information theory, mutual information $I(X;Y)$ between X and Y, measures the "amount of information" learned from knowledge of random variable Y about the other random variable X.

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

Computing $I(c; G(z,c))$ requires access to the posterior $P(c|x)$

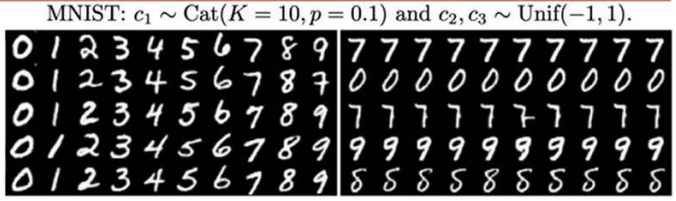
$Q(c|x) \rightarrow$ auxiliary distribution to approximate $P(c|x)$

$$L_I(G,Q) = E_{c\sim P(c), x\sim G(z,c)}[\log Q(c|x)] + H(c)$$
$$\leq I(c; G(z,c))$$

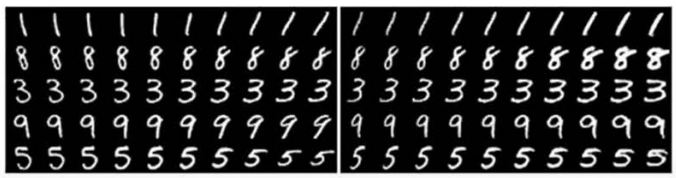
$L_I(G,Q) \rightarrow$ Variational Lower Bound

$$\min_{G,Q} \max_D V_{\text{InfoGAN}}(D,G,Q) = V(D,G) - \lambda L_I(G,Q)$$

MNIST: $c_1 \sim \text{Cat}(K=10, p=0.1)$ and $c_2, c_3 \sim \text{Unif}(-1,1)$.



(a) Varying c_1 on InfoGAN (Digit type) (b) Varying c_1 on regular GAN (No clear meaning)

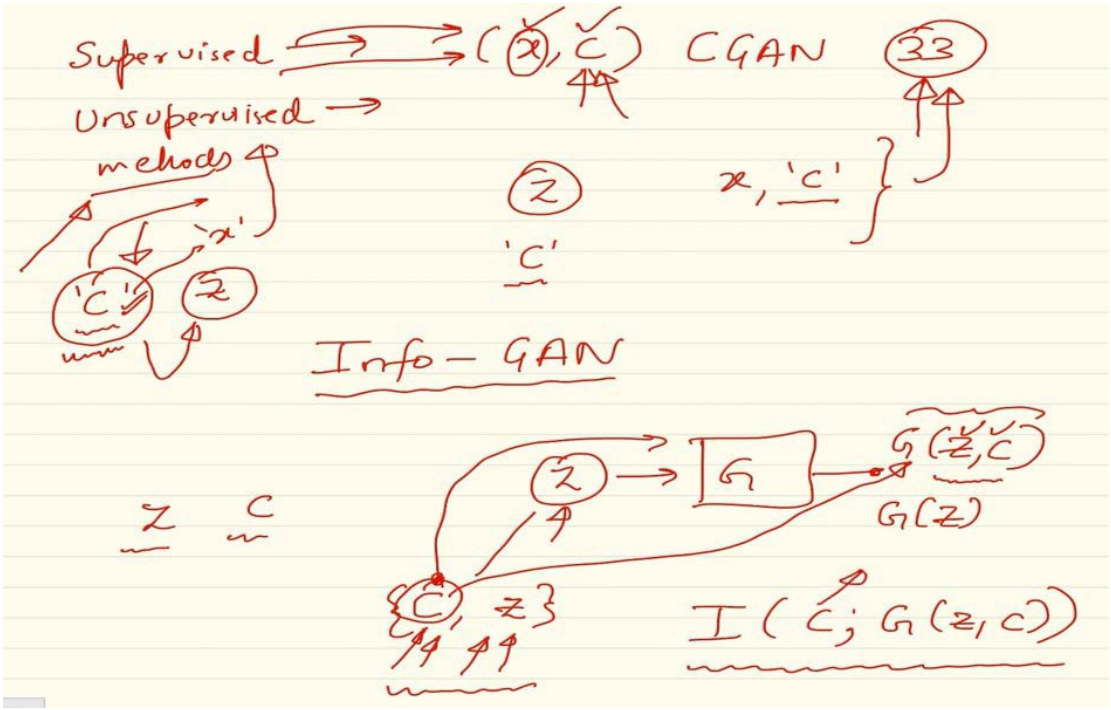


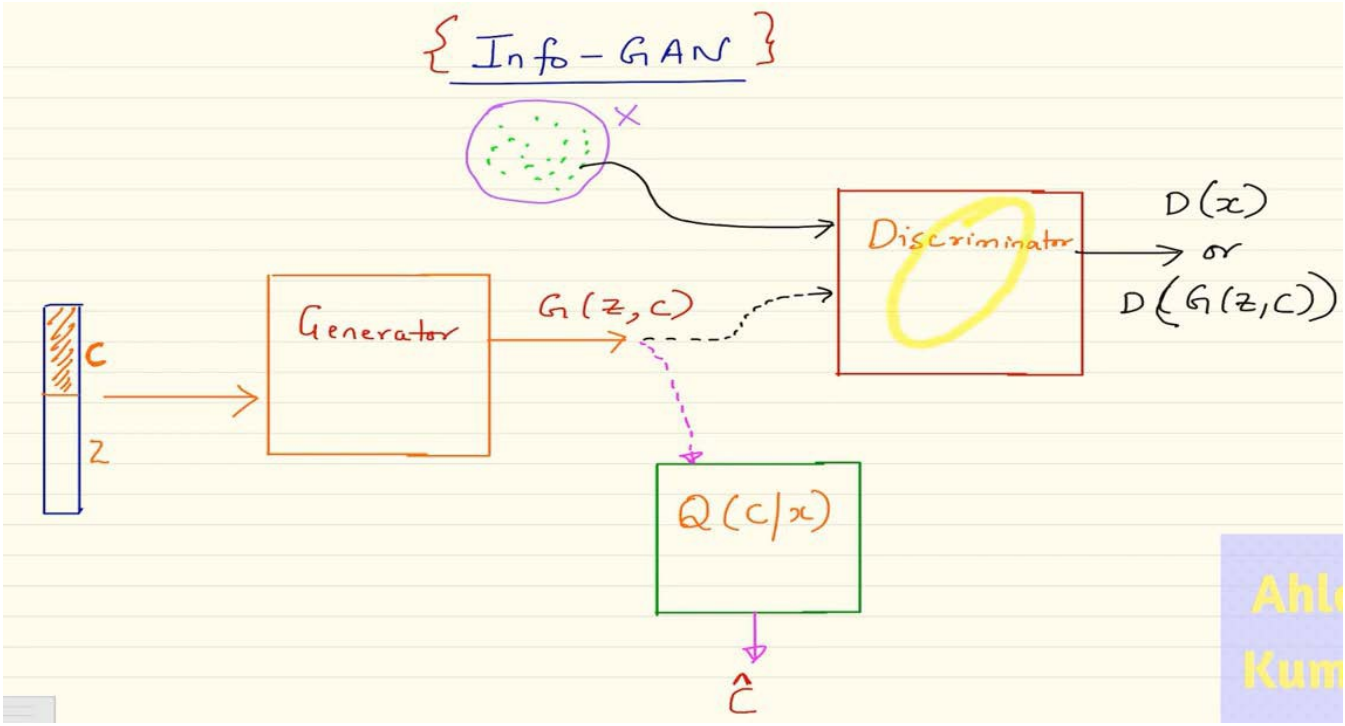
(c) Varying c_2 from -2 to 2 on InfoGAN (Rotation) (d) Varying c_3 from -2 to 2 on InfoGAN (Width)

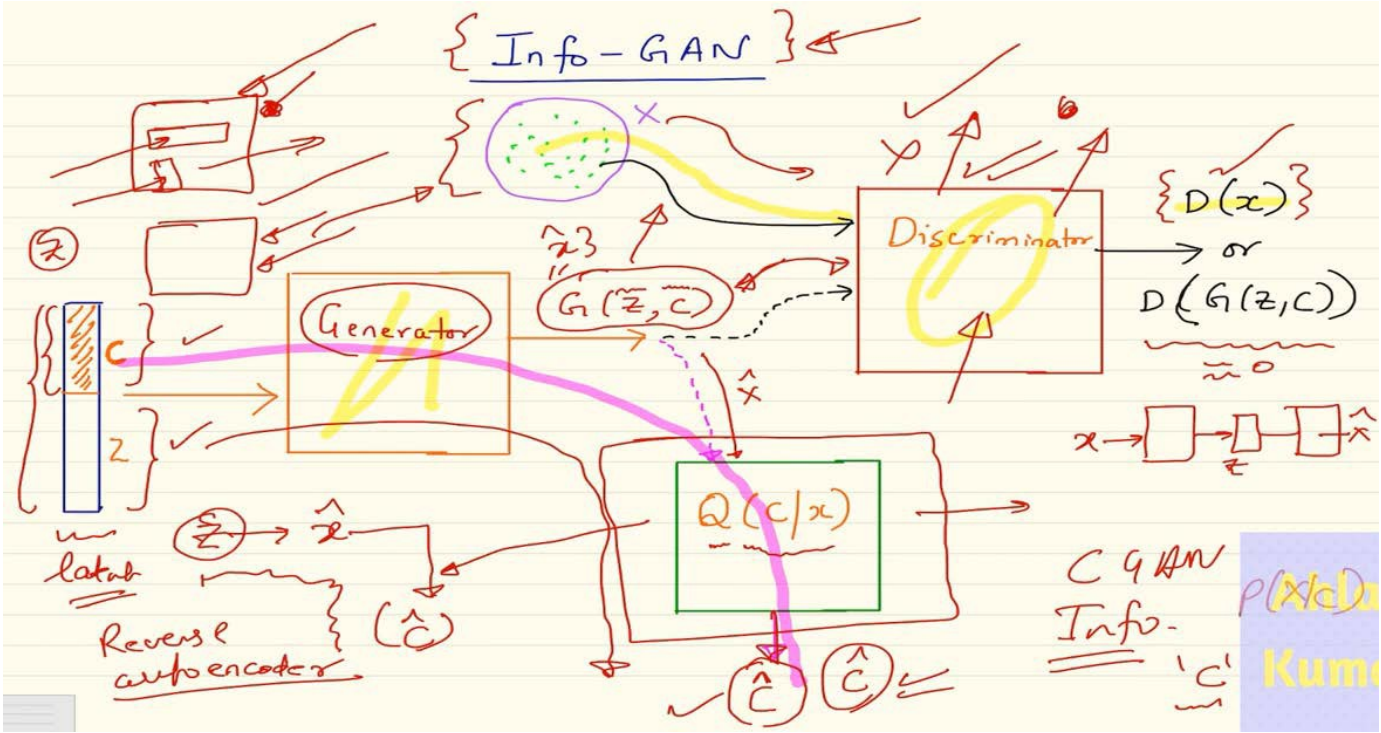
(a) Rotation (b) Width

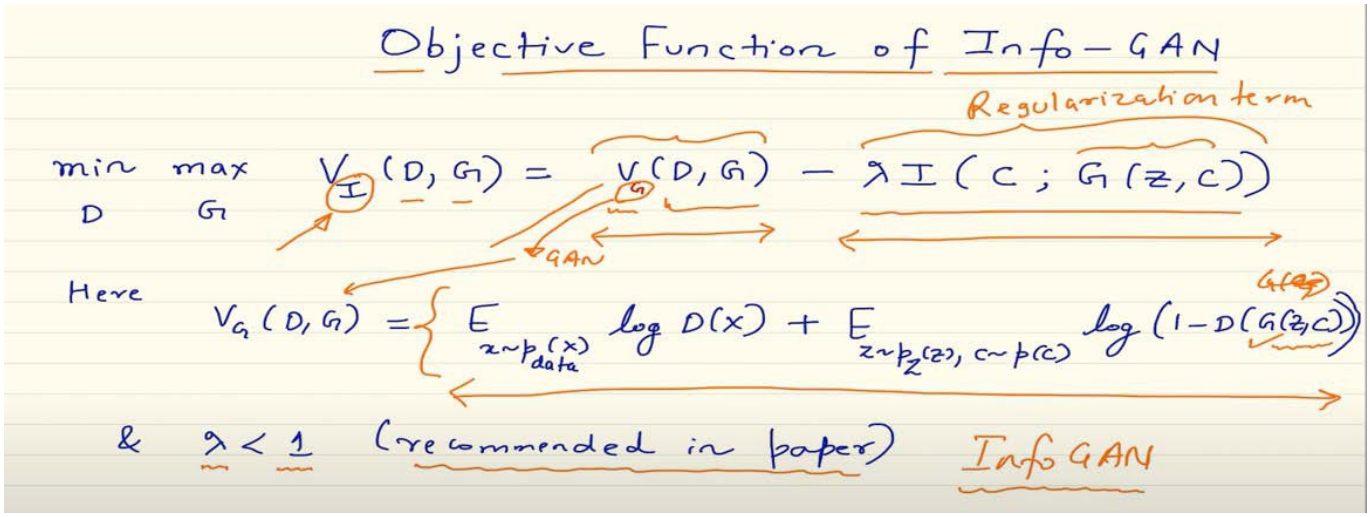
(c) Hair style (d) Emotion

Chen, Xi, et al. "Infogan: Interpretable representation learning by information maximizing generative adversarial nets." *arXiv preprint arXiv:1606.03657* (2016).




{ Info - GAN }
X
Discriminator
D(x)
or
D(G(z,c))
Generator
G(z,c)
c
z
Q(c|x)
Ĉ


{ Info - GAN }
X
Discriminator
D(x)
or
D(G(z,c))
Generator
G(z̃,c)
c
z
Q(c|x)
Reverse
auto encoder
Info-
'c'



Attention Mechanism

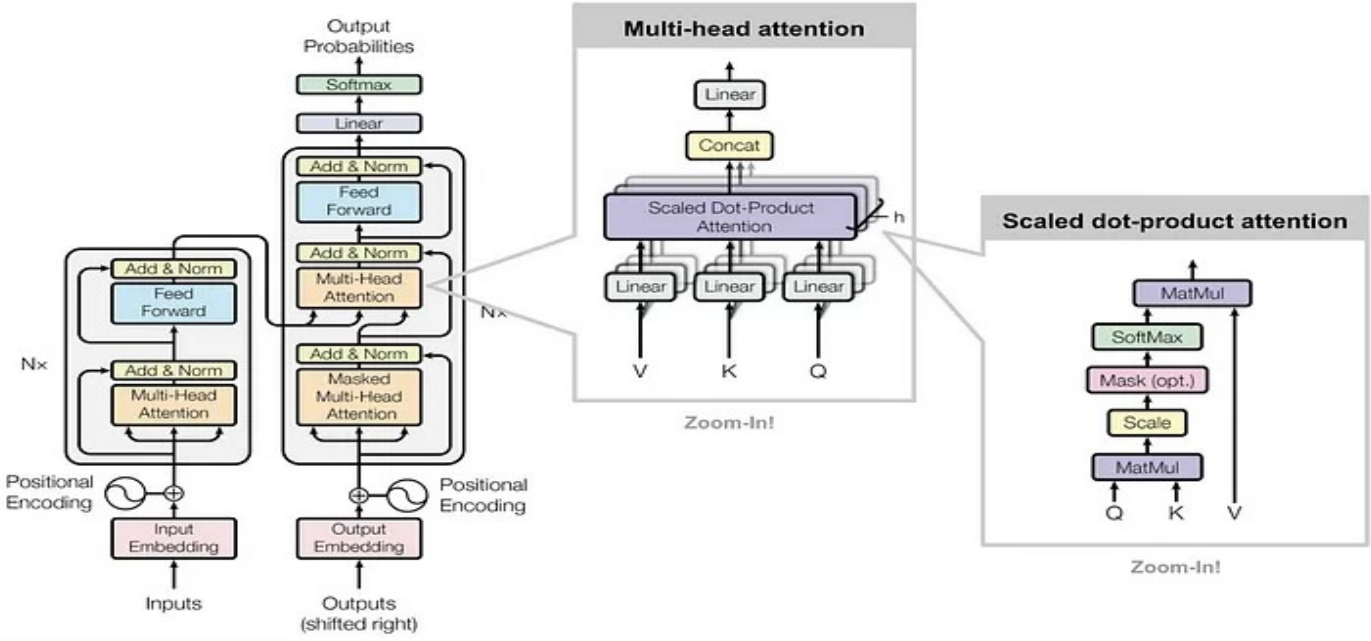
Dr. Chandra Sekhar V

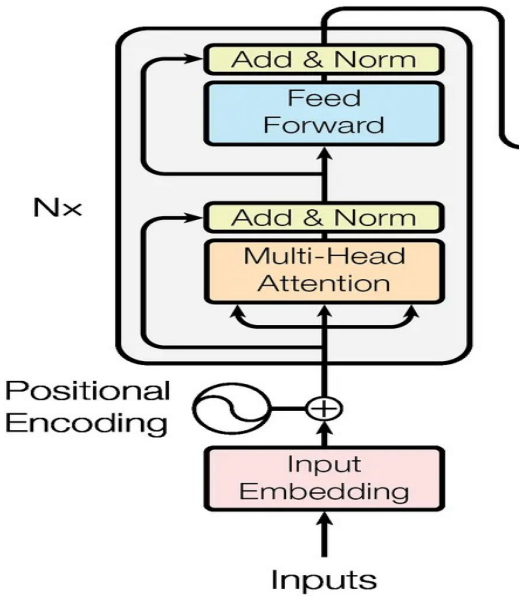
What was the need for Transformers?

- Recurrent neural networks (RNN) are capable of looking at previous inputs to predict the next possible word.
- But RNN's curse of the shorter window of reference, resulting in Vanishing Gradient, makes it difficult to capture the context of a story when the story gets longer.
- This is still true for Gated Recurrent Units (GRU's) and Long-short Term Memory (LSTM's) networks, although they do have a bigger capacity to achieve longer-term memory compared to RNN.

What was the need for Transformers?

Not only that, RNN is slow to train. Such a recurrent process does not make use of modern graphics processing units (GPUs), which were designed for parallel computation. But what's even worse is that LSTM is even slower to train.

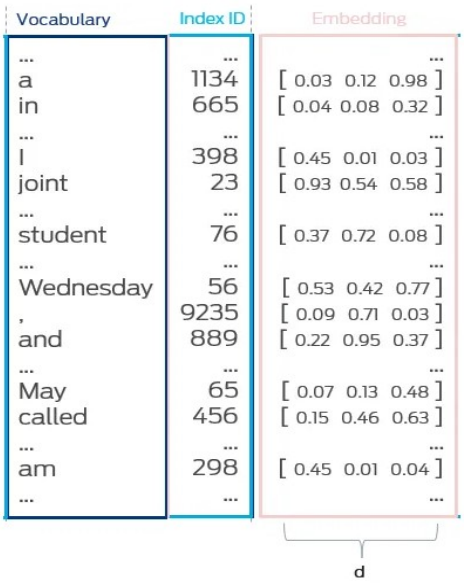

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)
Multi-head attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q
Zoom-In!
Scaled dot-product attention
MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q
K
V
Zoom-In!


Add & Norm
Feed Forward
Nx
Add & Norm
Multi-Head Attention
Positional Encoding
Input Embedding
Inputs

Inputs and Input Embedding

This model is trained from corpus of ~30,000 unique words. Each of these words have a unique ID, known as vocabulary index.

The next step is to convert the input word into it's corresponding word embedding. Word embedding is the vector representation of each words in the vocabulary.



In reality, the dimension is 512, 768 or even 1024. The more, the better.

- Each of these dimensions captures "some" linguistic feature about that word. Since the model decides these features itself during the training, it can be non-trivial to find out what exactly each of the dimensions represents.
- These vectors are randomly initialized and IT IS THESE that will get fine-tuned during the model training, and will ultimately generate the contextual representation of the words to be leveraged during the inference time.

Positional Encoding

- In recurrent networks like LSTMs and GRUs, the network processes the input sequentially, token after token. The hidden state at position t+1 depends on the hidden state from position t.
- This way, the network has a reference to identify the relative positions of each token by accumulating information.
- However, Transformers has no notion of word order. That's why it is faster but we do need the information of word's positions.

Even though she did not win the award, she was satisfied.

Even though she did win the award, she was not satisfied.

Hence the requirement of positional encoding. So how do we do it?

Strategy 1: Add vector of positions IDs (0,1,2,...,(N-1)) with the Word Vectors.

Positional Encoding

Implemented Strategy

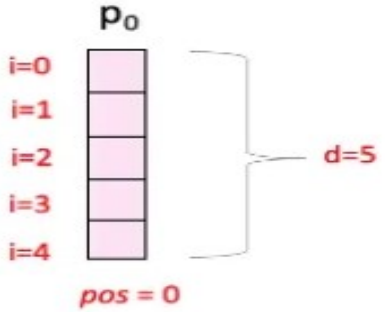
Hence, the authors propose a cyclic (dynamic) solution where sine and cosine function with different frequencies is added to each word embedding. The formula goes like this:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

Positional Encoding

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

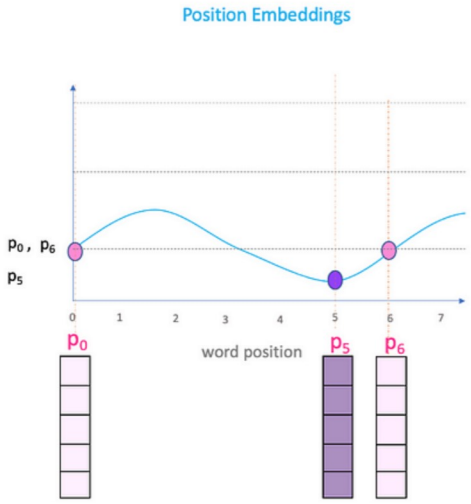


Here pos refers to the position of the word in the sequence. P0 refers to the position embedding of the first word, d means the size of the word/token embedding (here it is d=5). Finally, i refers to each of the 5 individual dimensions of the embedding (i.e. 0, 1,2,3,4).

Positional Encoding

While d is fixed, pos and i vary. Let us try understanding the latter two.

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

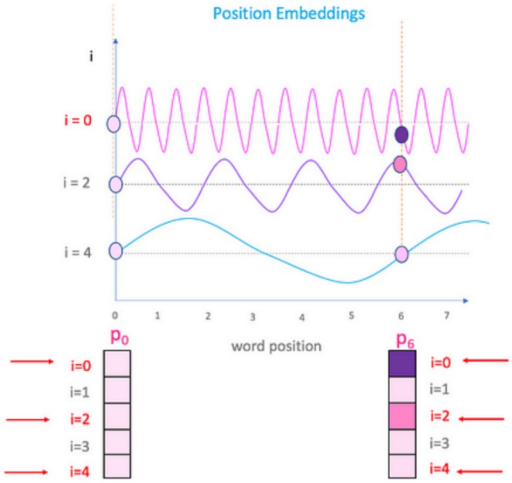


If we plot a sin curve and vary pos (on the x-axis), you will land up with different position values on the y-axis. Therefore, words with different positions will have different values position embeddings values.

Positional Encoding

There is a problem though. Since sin curve repeat in intervals, you can see in the figure above that P0 and P6 have the same position embedding values, despite being at two very different (word) positions. This is where the i part in the equation comes into play.

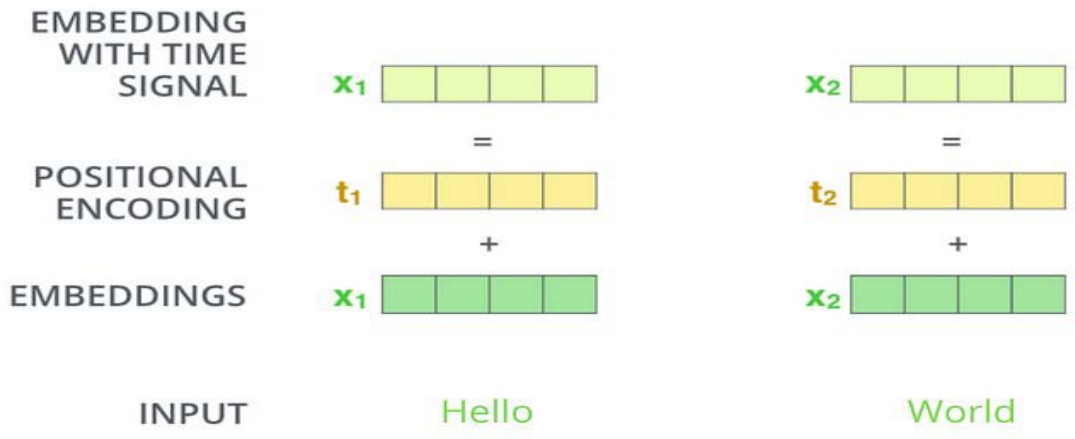
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$



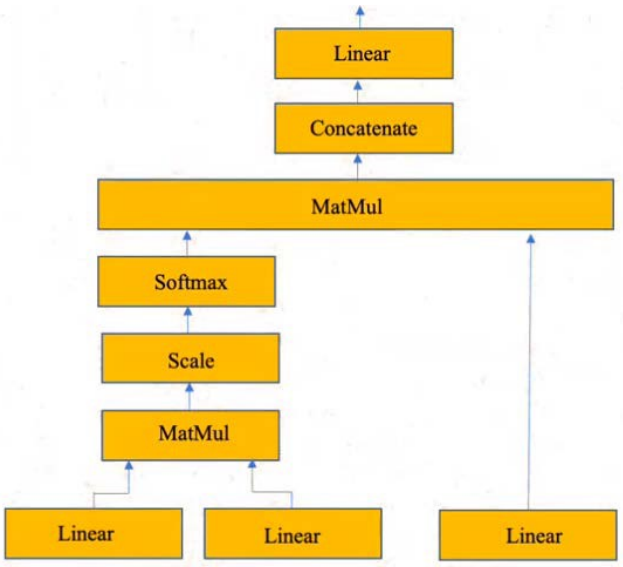
Positional Encoding

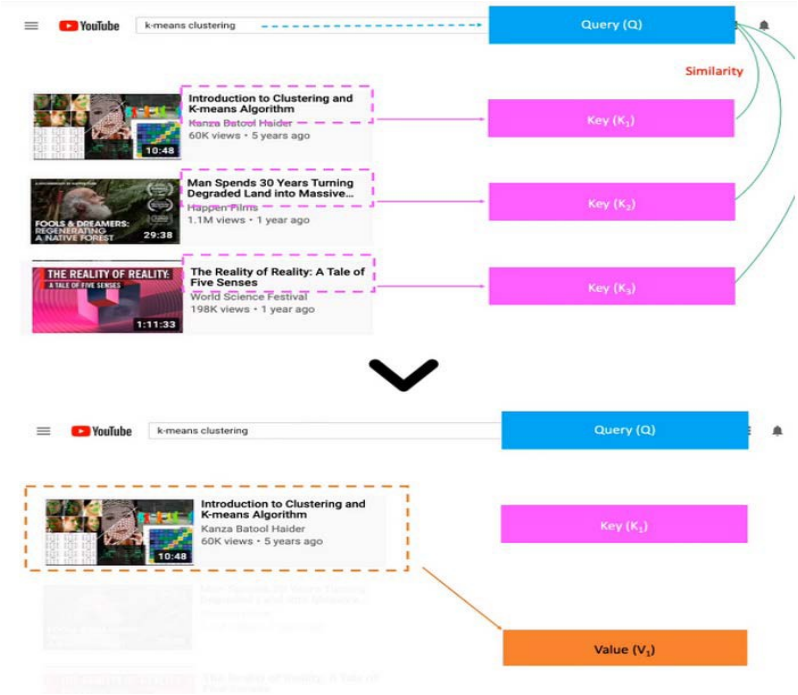
- The position encoding in transformers alternates between sine and cosine functions for each dimension of the position vector, which helps the model determine the position of each word in the sequence.
- By applying sine to even indices and cosine to odd indices of the position vector, the model creates a unique position embedding for each word.
- This method allows the model to capture relative positions between words since the functions' outputs vary predictably with position.
- The varying frequencies of these sine and cosine functions across different dimensions enable the model to encode positional information in a high-dimensional space, facilitating the understanding of word order and the syntactic structure of sentences.

Positional Encoding

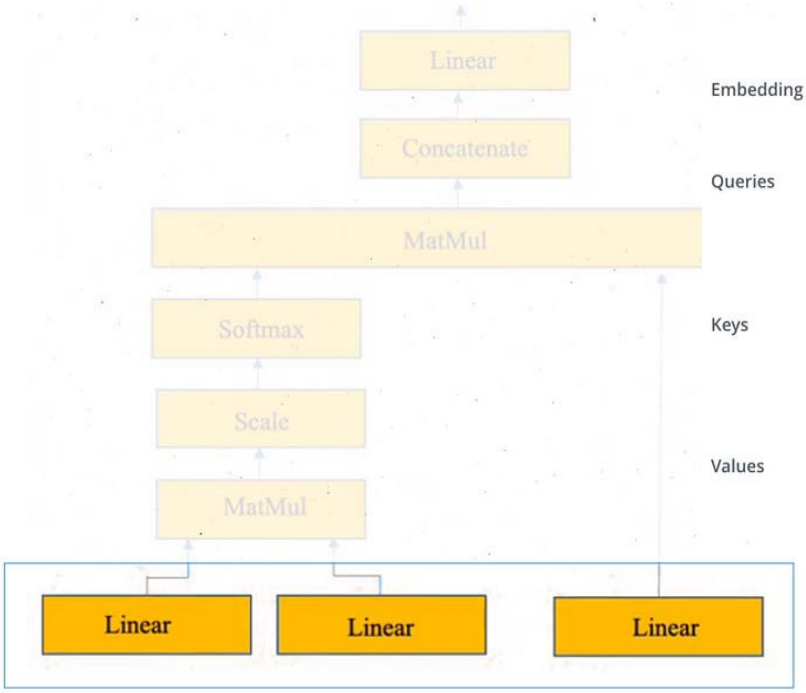


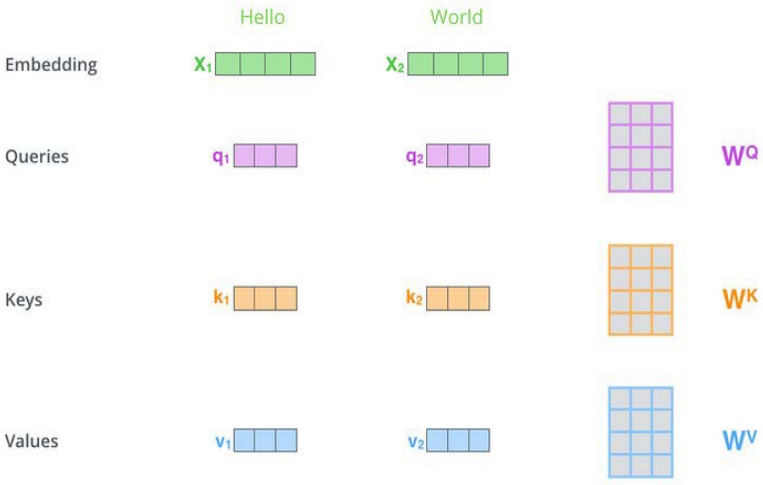
Positional Encoding



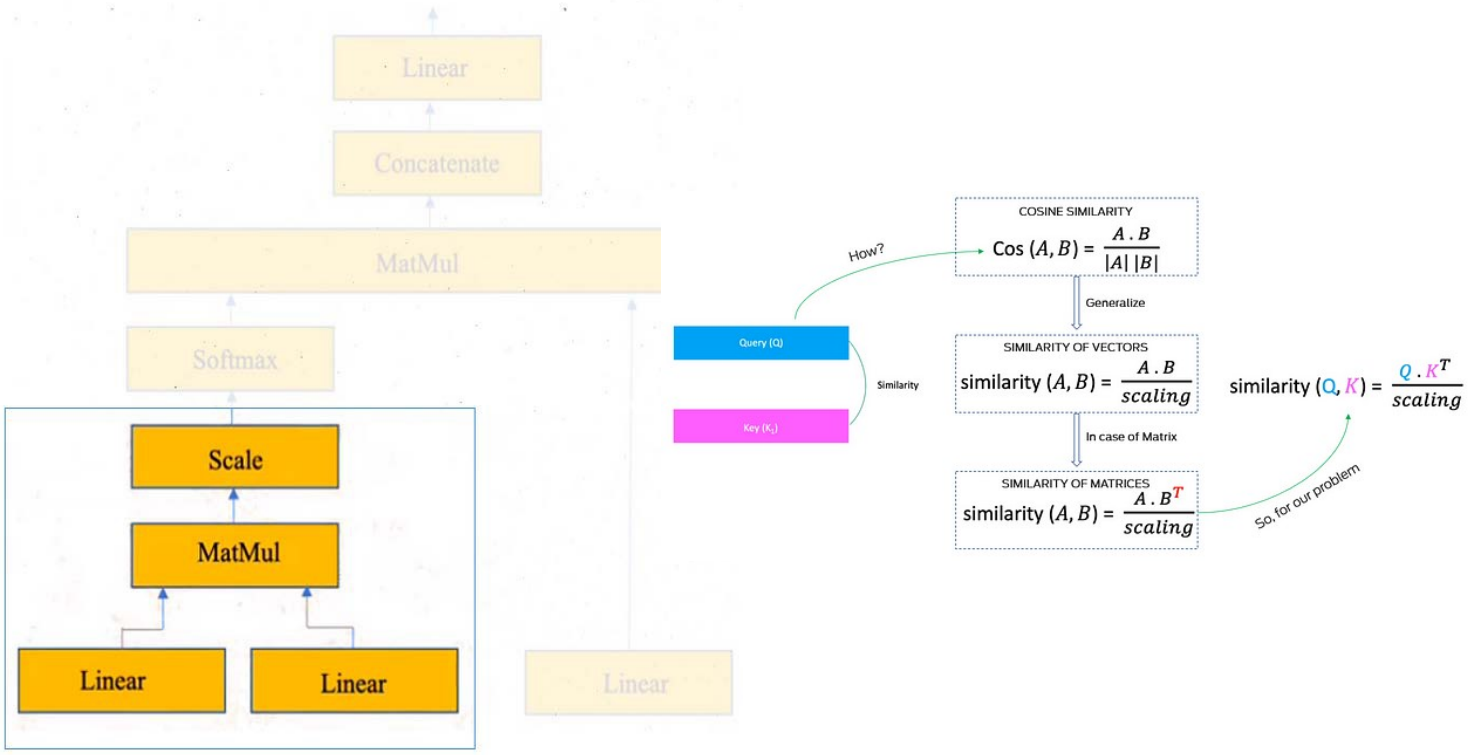


Positional Encoding



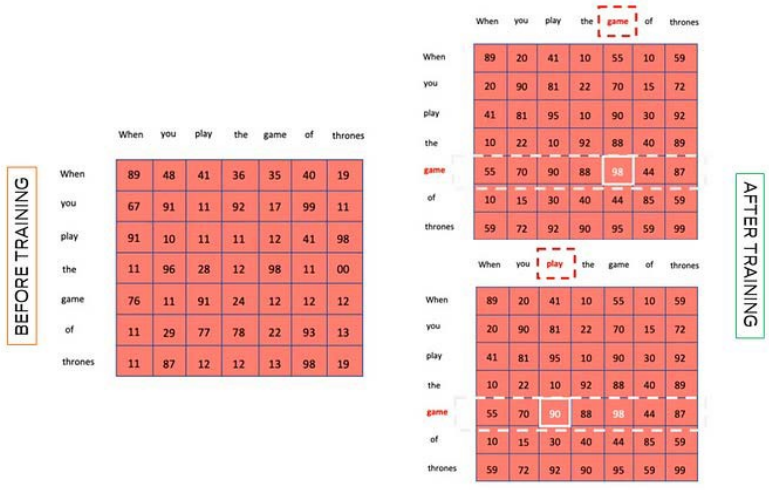


Positional Encoding



Positional Encoding

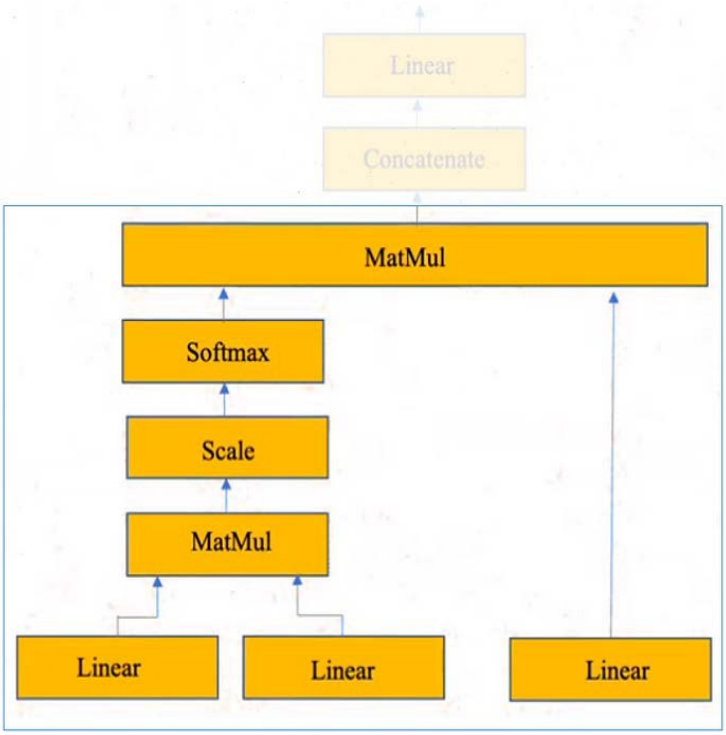
Now, Query and transpose of Key undergoes dot product matrix multiplication to generate a score matrix, which determines how much focus should a word be put on other words. Higher score means more focus. This is how the queries are mapped to the keys.



Positional Encoding

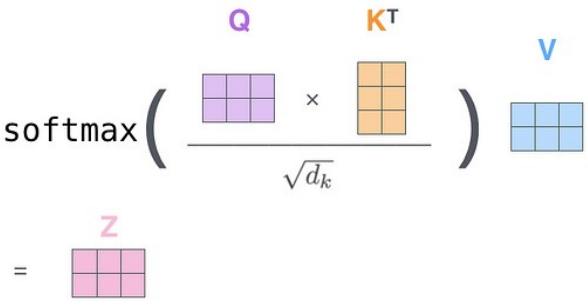
- Then, the scores get scaled down by getting divided by the square root of the dimension of key vector. This is to allow for more stable gradients, as multiplying values can have exploding effects.
- Next, you take the softmax of the scaled score to get the attention weights (or filters), which gives you probability values between 0 and 1.
- By doing a softmax the higher scores get enhanced, and lower scores are depressed. This allows the model to be more confident about which words to attend.

Mapping the Attention Weight with Original Matrix

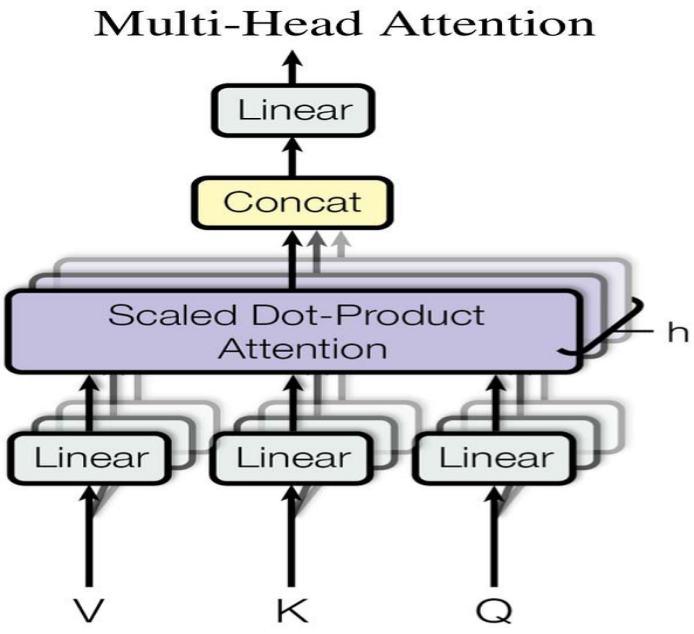


- Then you take the attention weights and multiply it by your Value vector to get an output matrix Z.
- The higher softmax scores will keep the value of words the model learns is more important. The lower scores will drown out the irrelevant words.
- Why is this multiplication done? The best way to explain the reason for implementing this technique is in the context of computer vision.
- Imagine encountering Yahiko from the 6 path of Pain. In reality, the entire view is like this.

Final Step/Summary

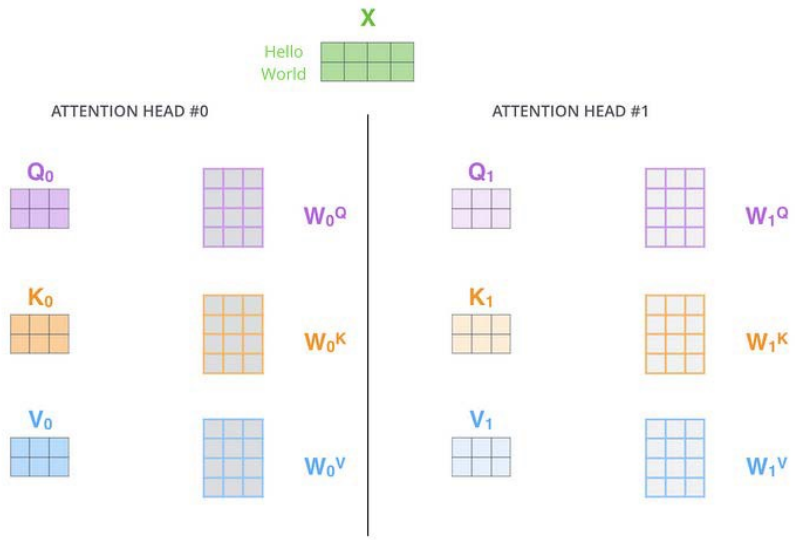


Final Step/Summary

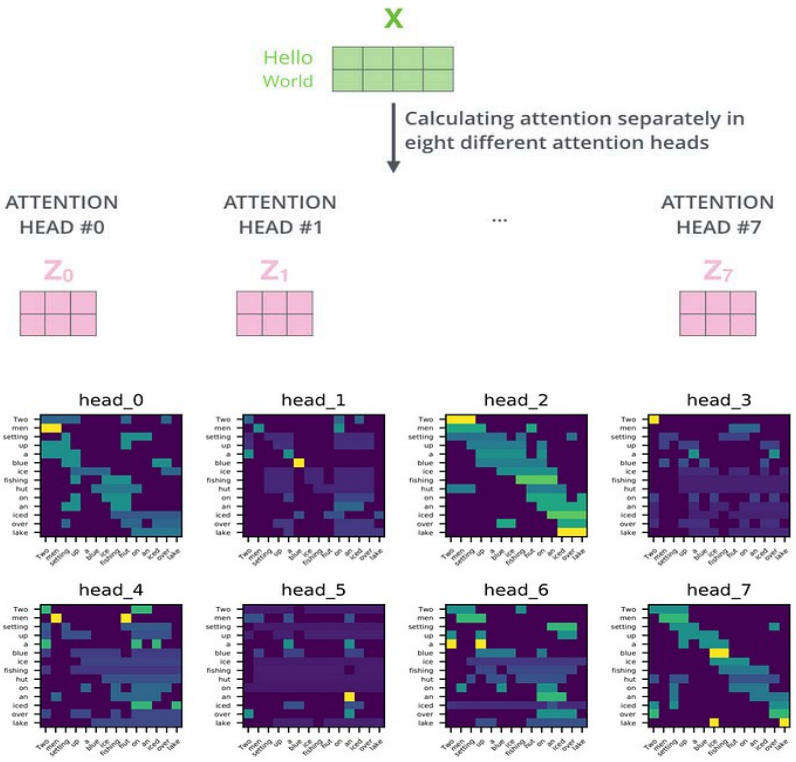


Final Step/Summary

Final Step/Summary

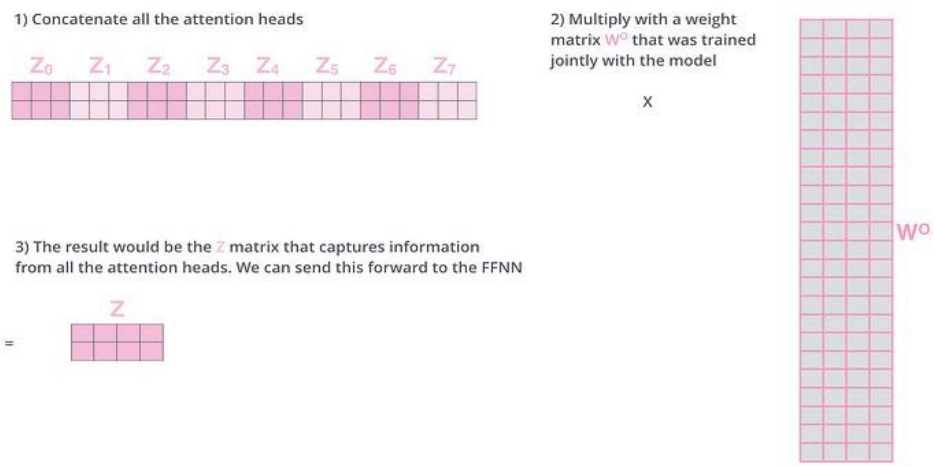


Final Step/Summary

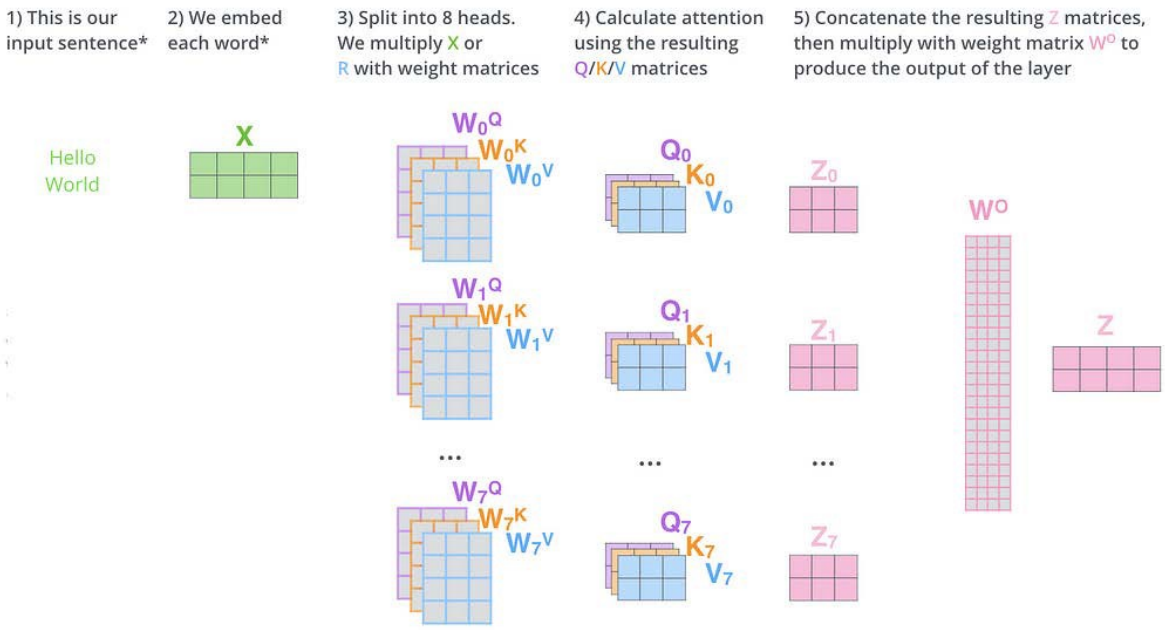


Final Step/Summary

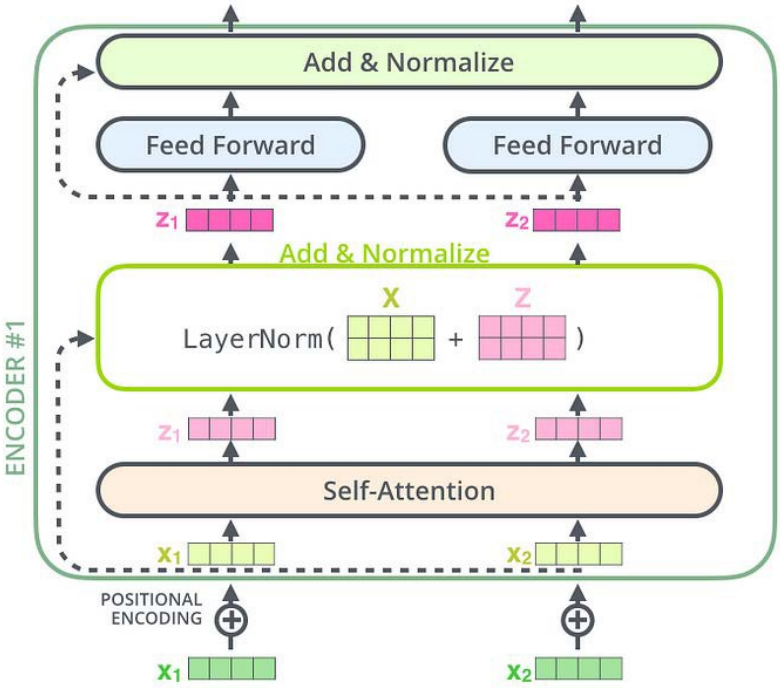
- However, this leaves us with a bit of a challenge. The upcoming feed-forward neural network (FFNN) layer is not expecting 8 matrices — it's expecting a single matrix (a vector for each word).
- Hence, we concatenate the matrices, then pass it through another linear layer (again comprising of weights matrices) WO and get the original vector dimension back.



Final Step/Summary



Final Step/Summary



The multi-headed attention output vector is added to the original positional input embedding. This is called a residual connection. The purpose of this component is to preserve the original context, thereby tackling the Vanishing Gradient problem.