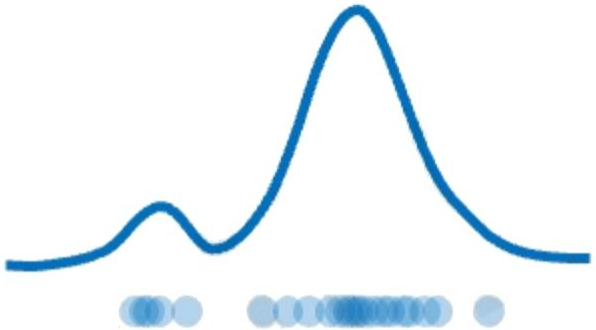
Deep Generative Modeling

(unsupervised learning models)

Generative Modeling

Goal: Take as input training samples from some distribution and learn a model that represents that distribution

Density Estimation



samples

Sample Generation

Input samples

Training data ~ $P_{data}(x)$

Generated samples

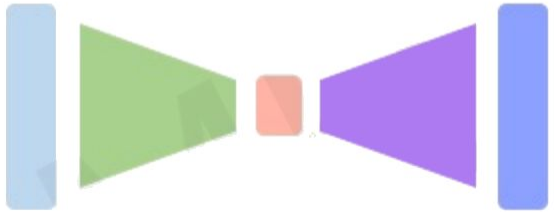
Generated ~ $P_{model}(x)$

How can we learn $P_{model}(x)$ similar to $P_{data}(x)$?

Deep Generative Models

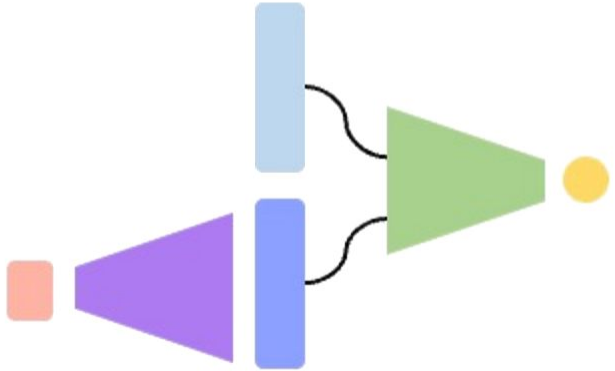
Autoencoders and Variational Autoencoders (VAEs)

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



Generative Adversarial Networks (GANs)

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



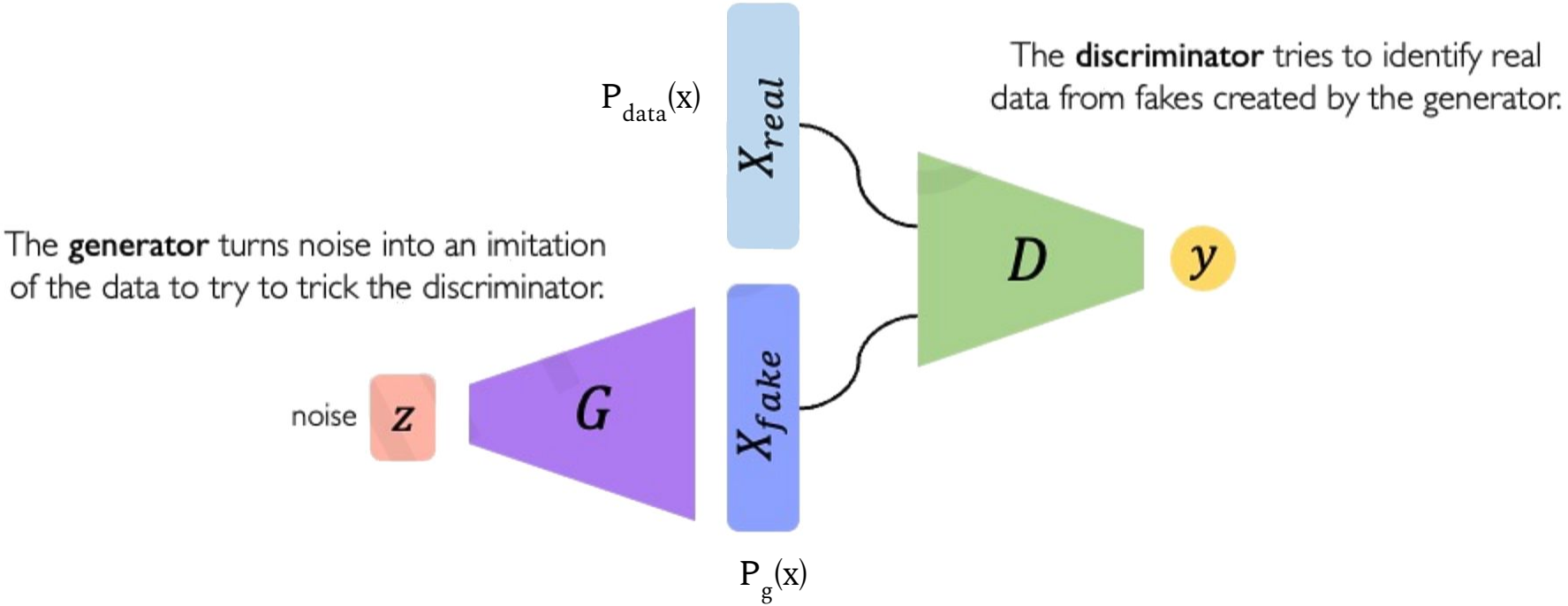
What Makes a Good Generative Model?

- Efficient Sampling: Quick and cost-effective generation of samples.
- High-Quality Sampling: Generated samples are indistinguishable from the training data.
- Coverage: Ability to represent the entire training data distribution.

What Makes a Good Generative Model?

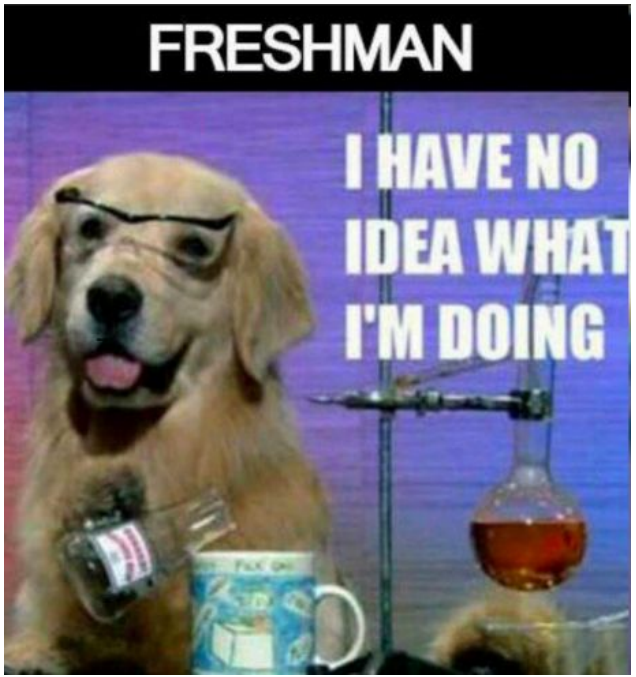
- Well-Behaved Latent Space: Smooth changes in latent variables result in smooth changes in data examples.
- Disentangled Latent Space: Each latent dimension influences interpretable and distinct aspects of the data.

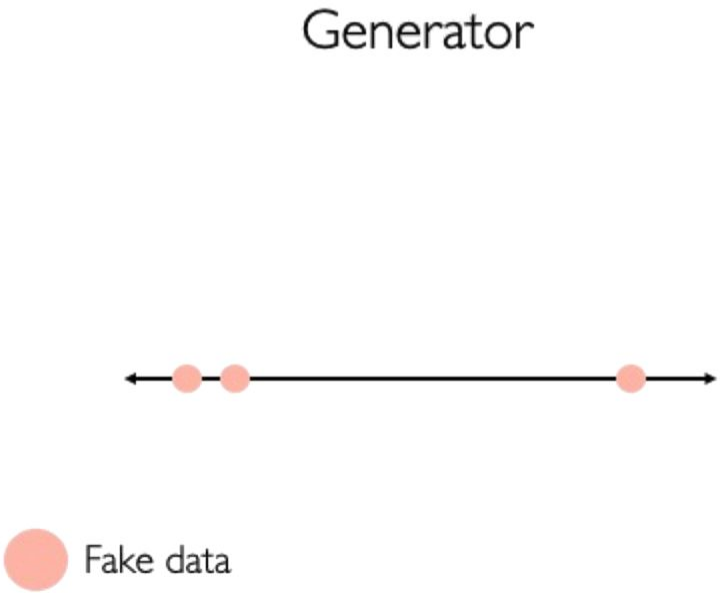
Generative Adversarial Networks (GANs) are a way to make a generative model by having two neural networks compete with each other.



Intuition behind GAN

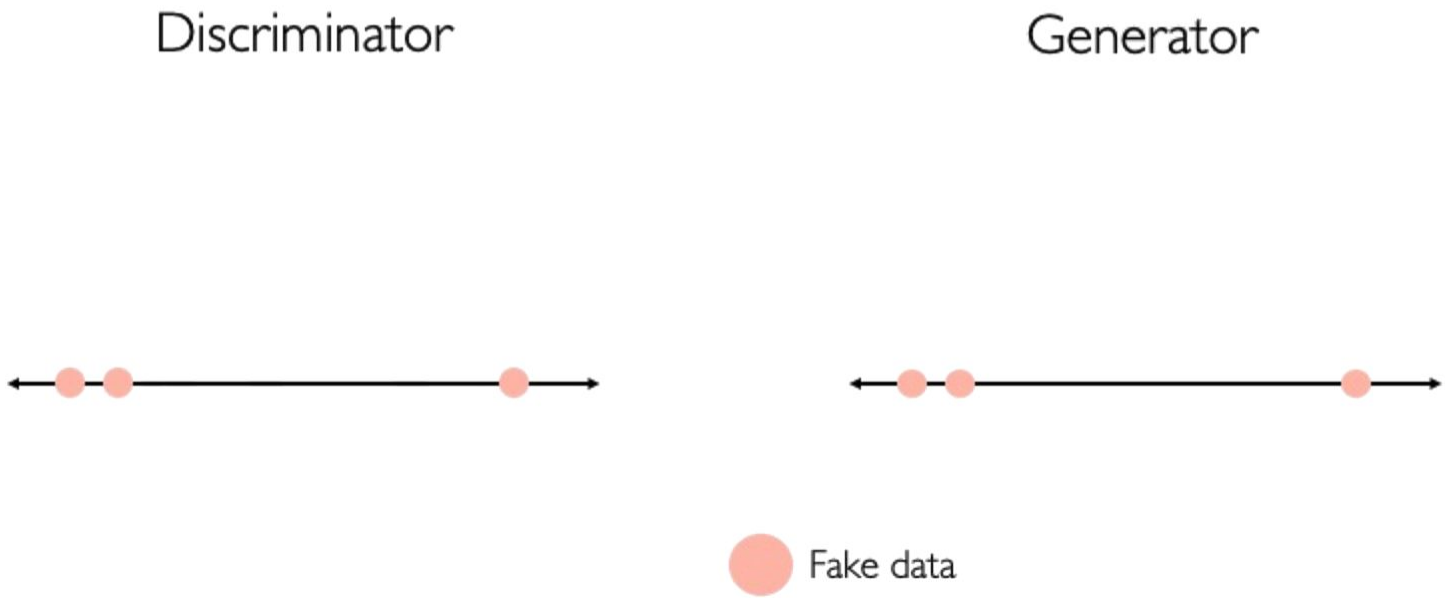
Generator starts from noise to try to create an imitation of the data.





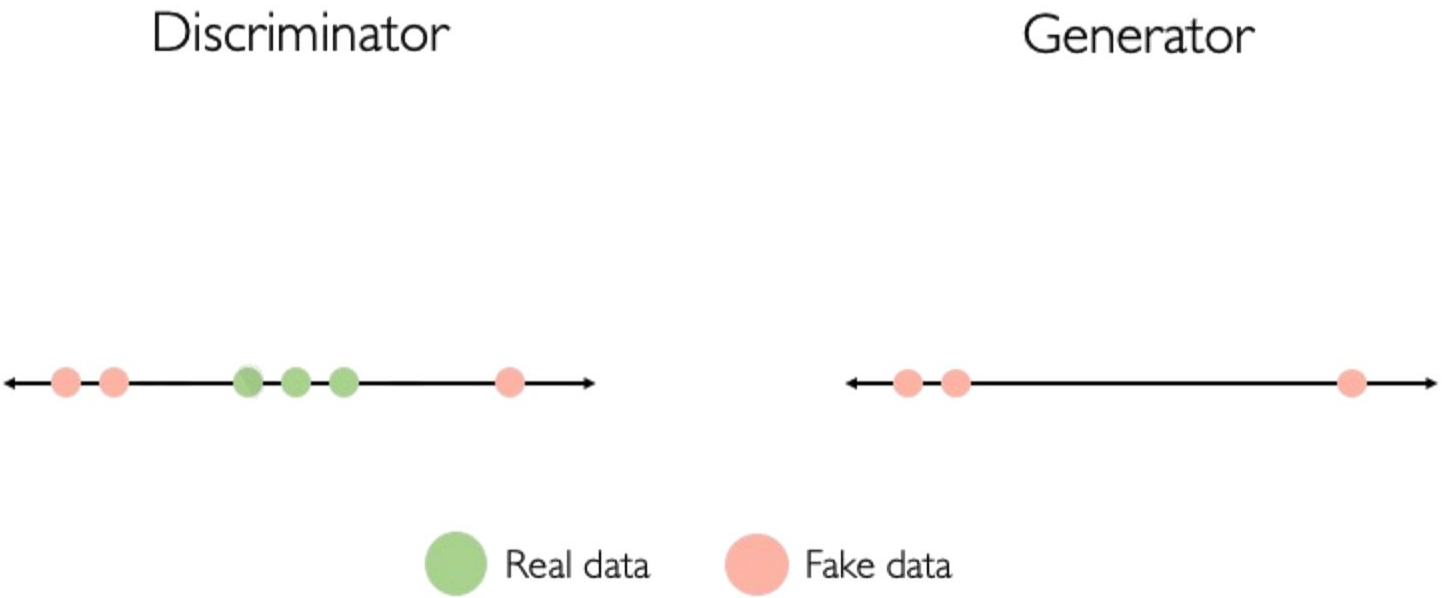
Intuition behind GAN

Discriminator looks at both real data and fake data created by the generator.

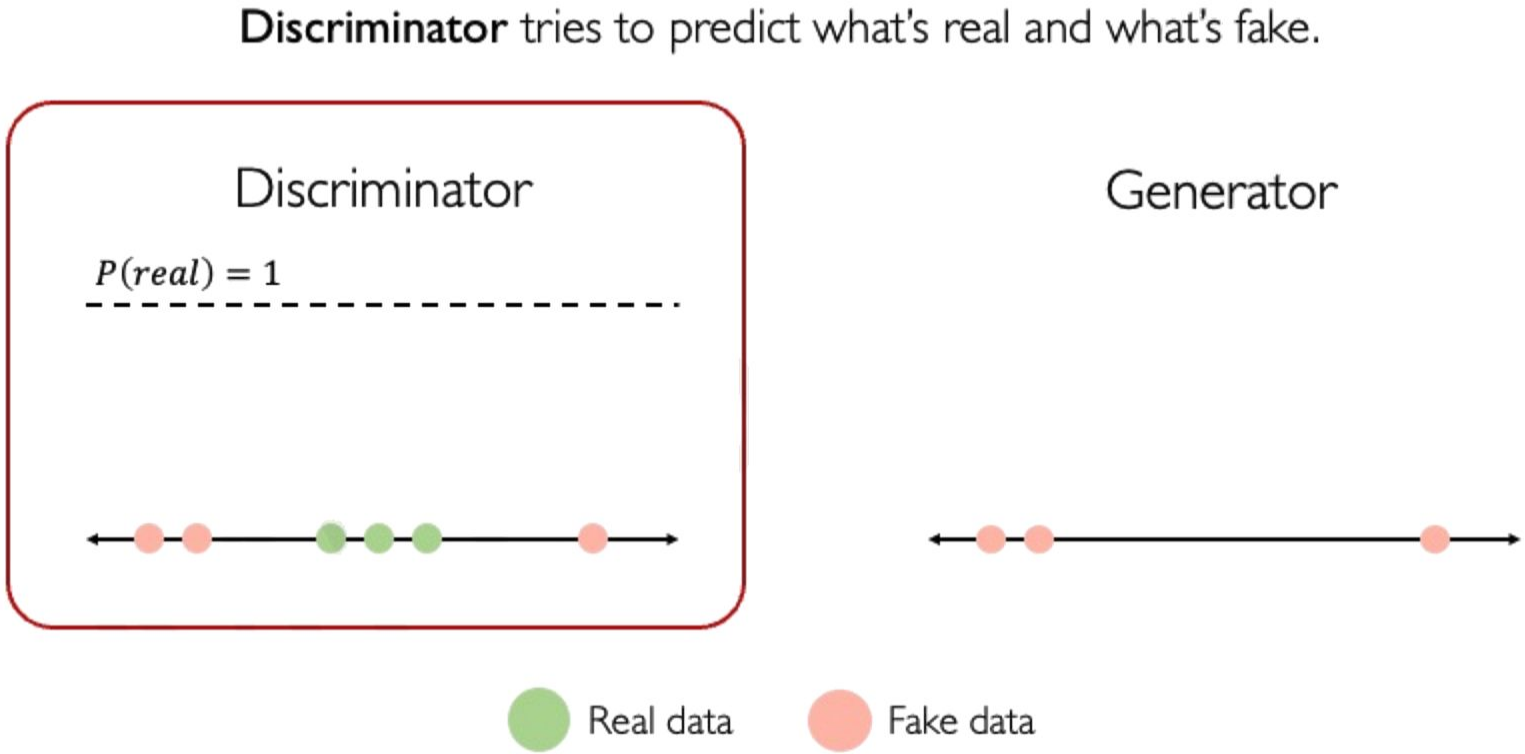


Intuition behind GAN

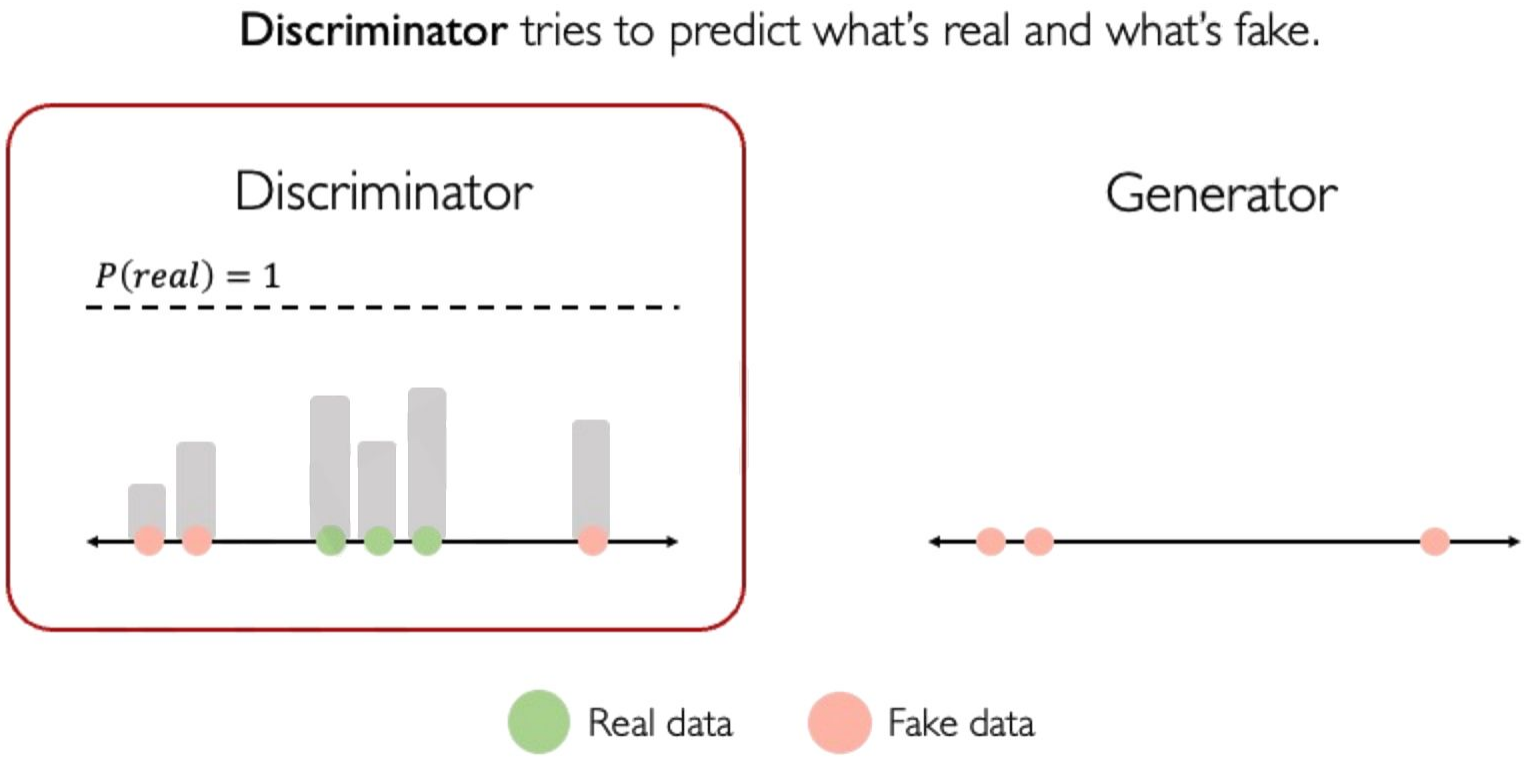
Discriminator looks at both real data and fake data created by the generator.



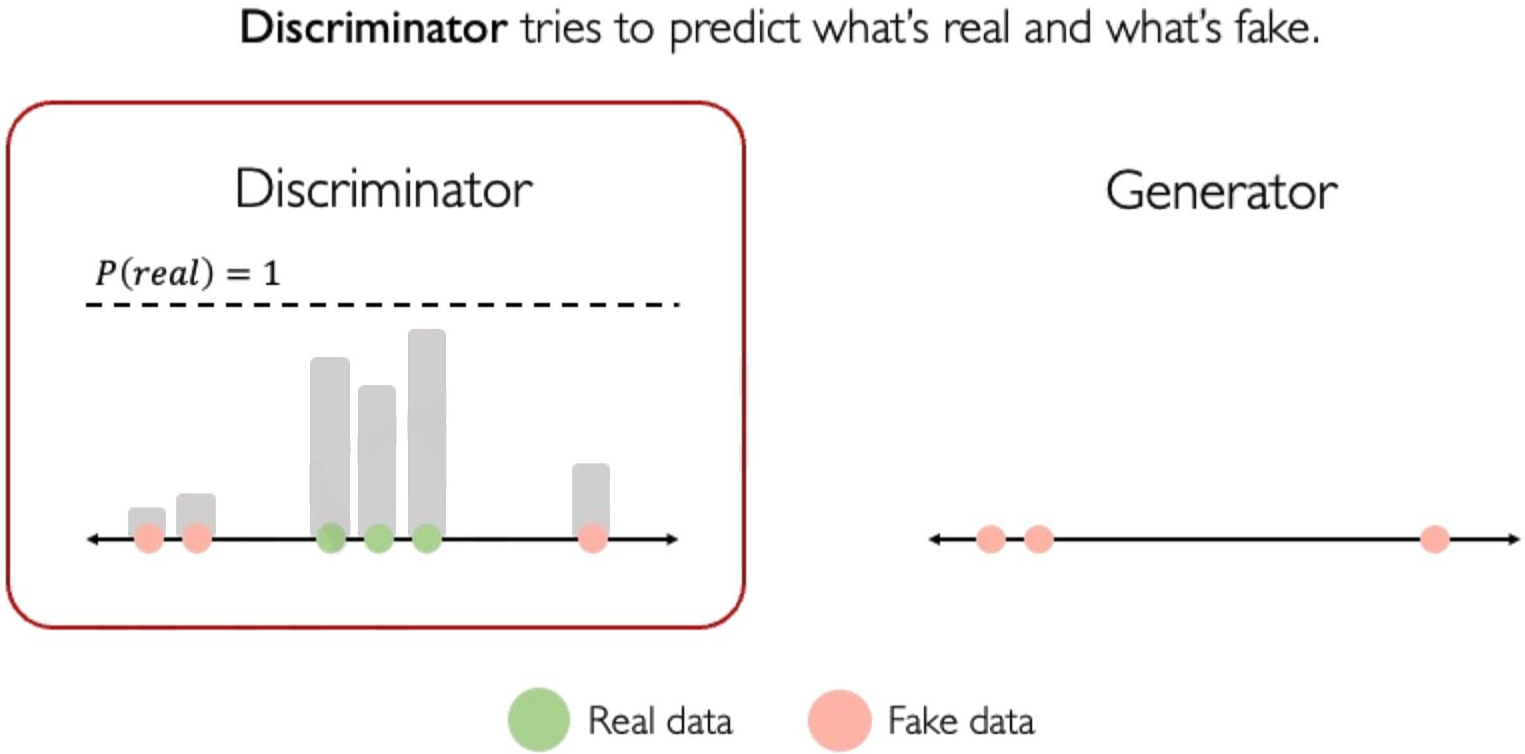
Intuition behind GAN



Intuition behind GAN

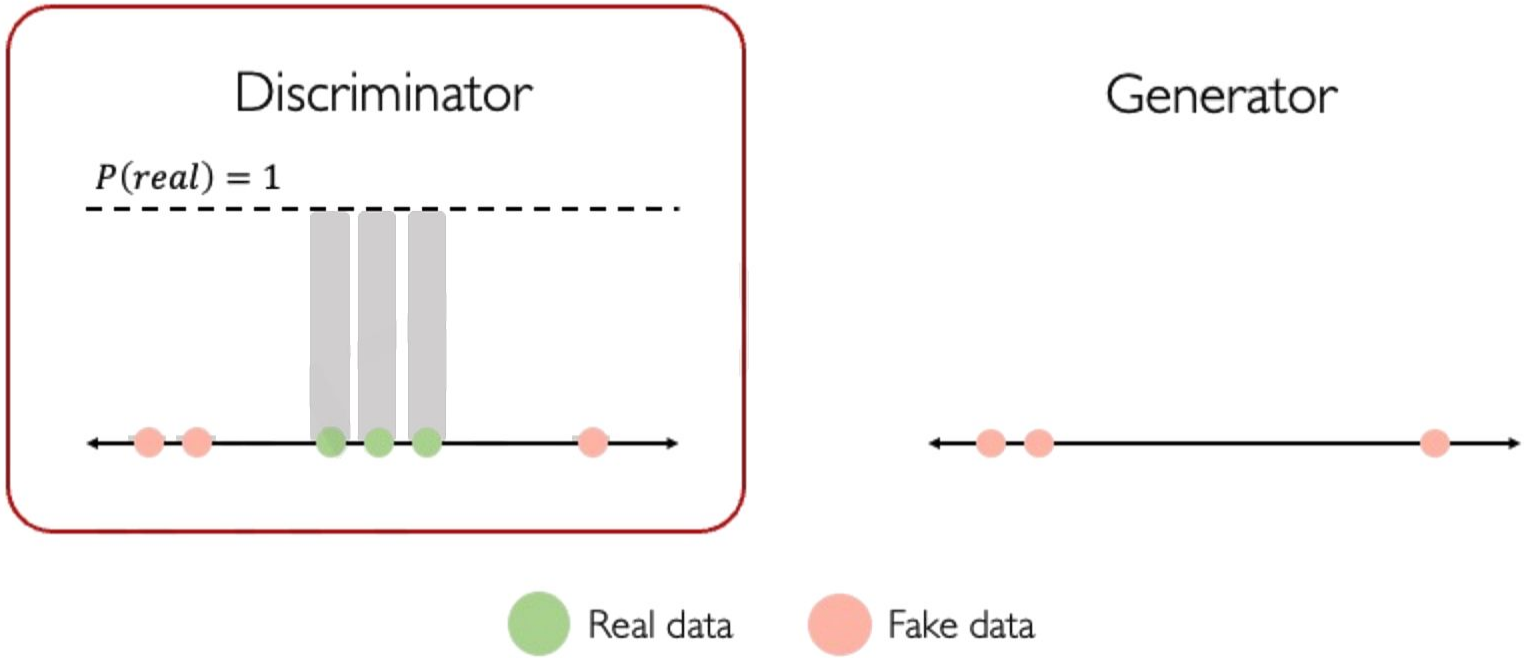


Intuition behind GAN



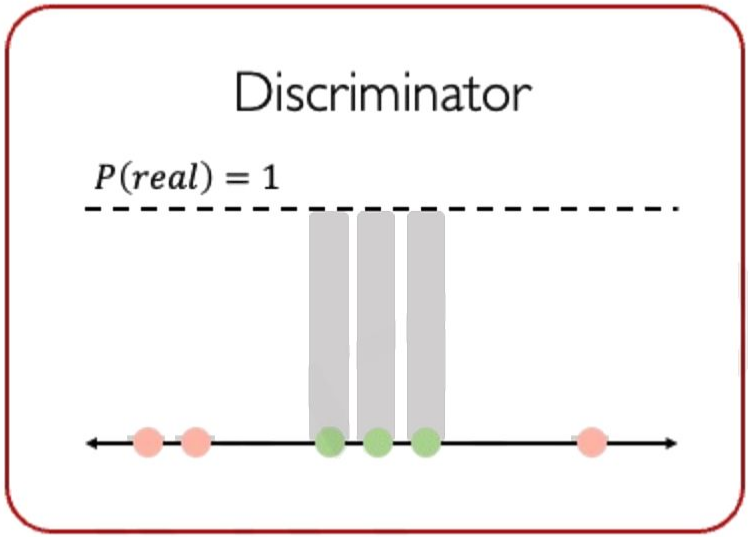
Intuition behind GAN

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



Intuition behind GAN

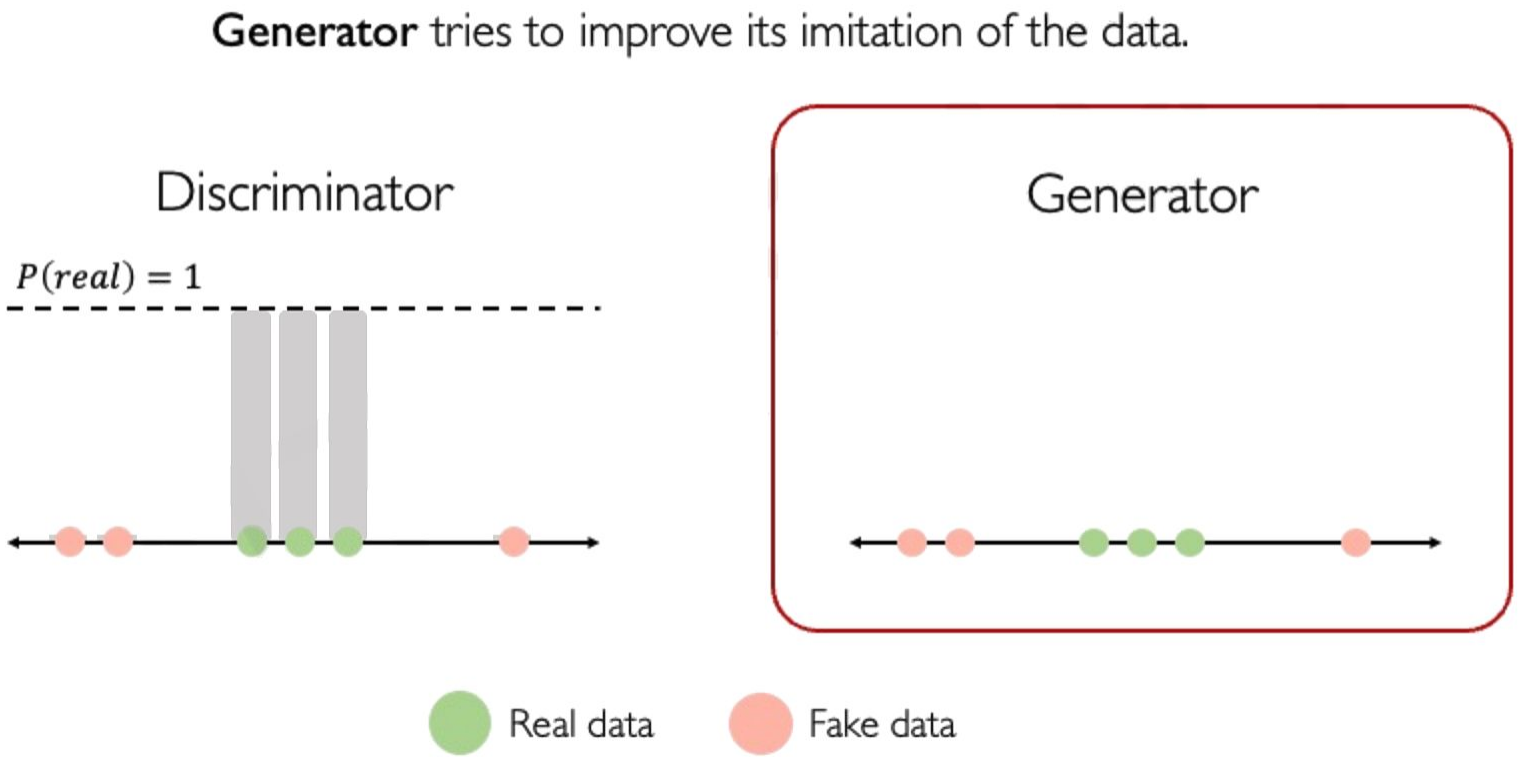
Discriminator tries to predict what's real and what's fake.



Real data

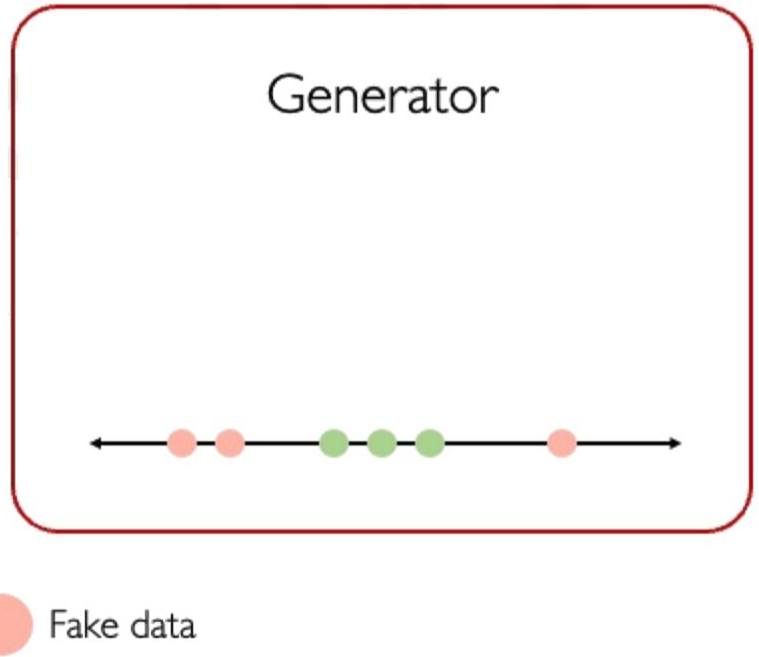
Fake

Intuition behind GAN



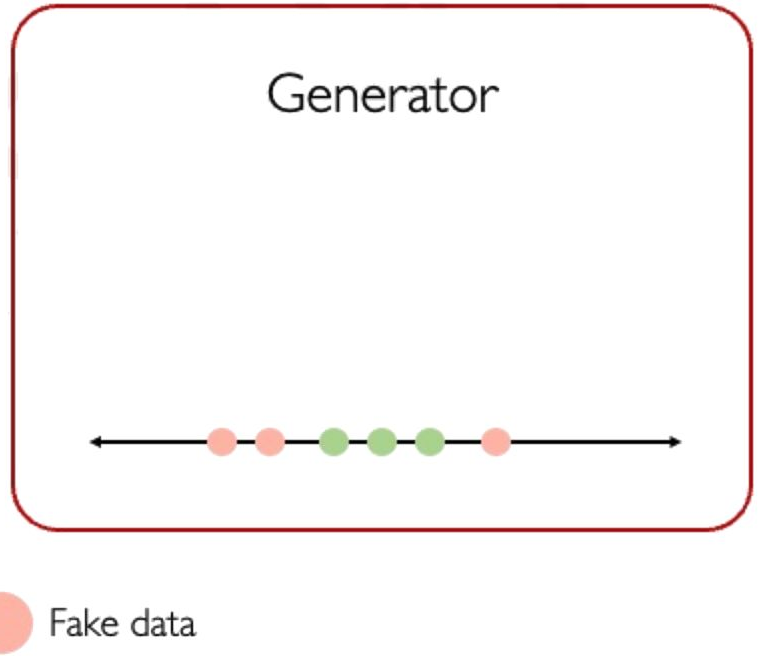
Intuition behind GAN

Generator tries to improve its imitation of the data.

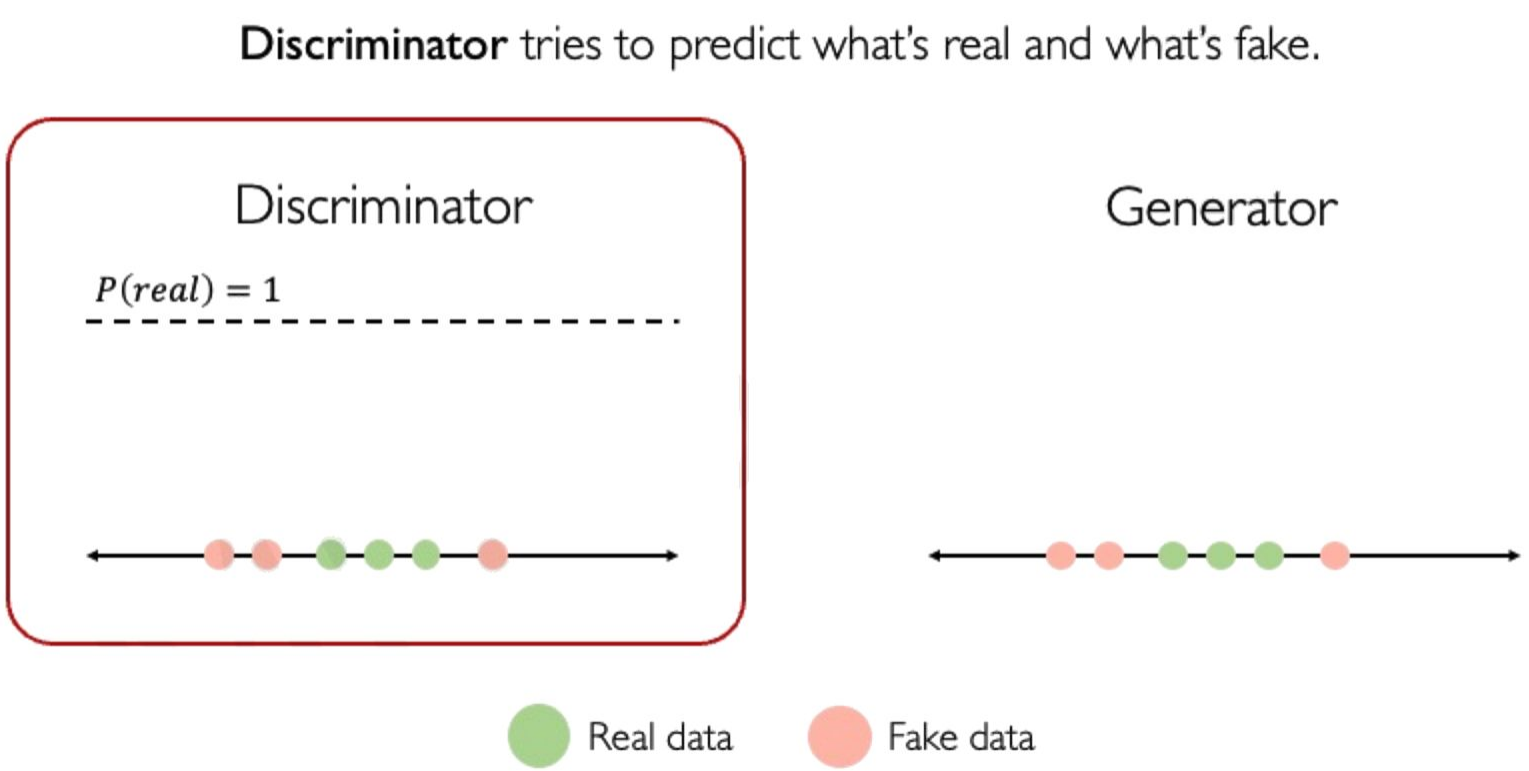


Intuition behind GAN

Generator tries to improve its imitation of the data.

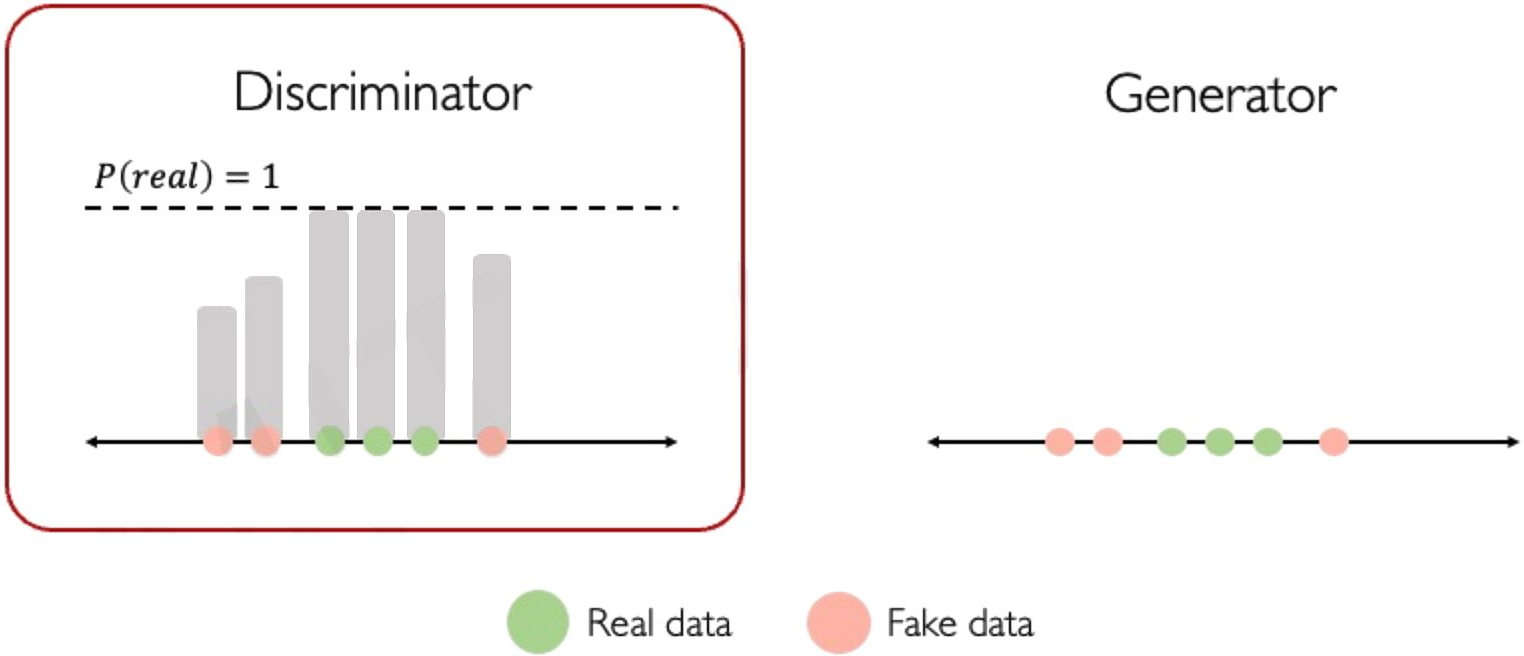


Intuition behind GAN



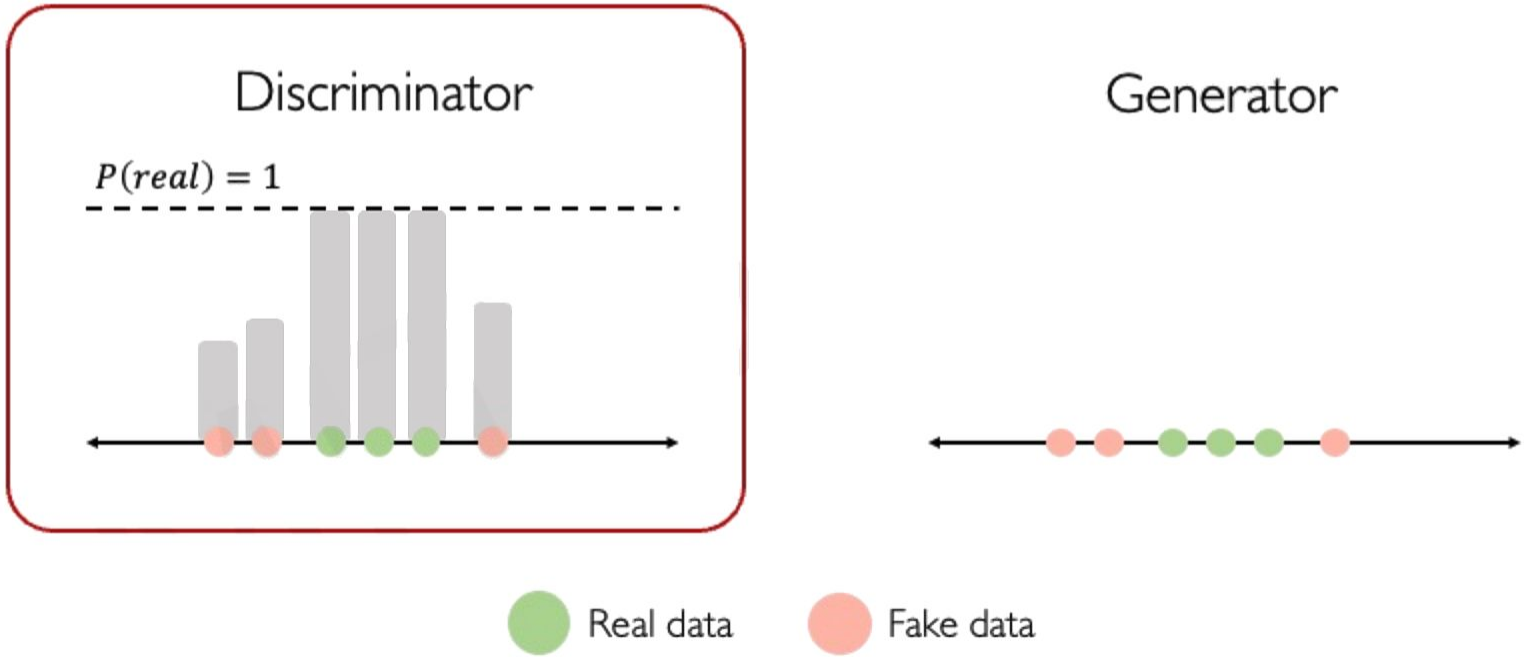
Intuition behind GAN

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



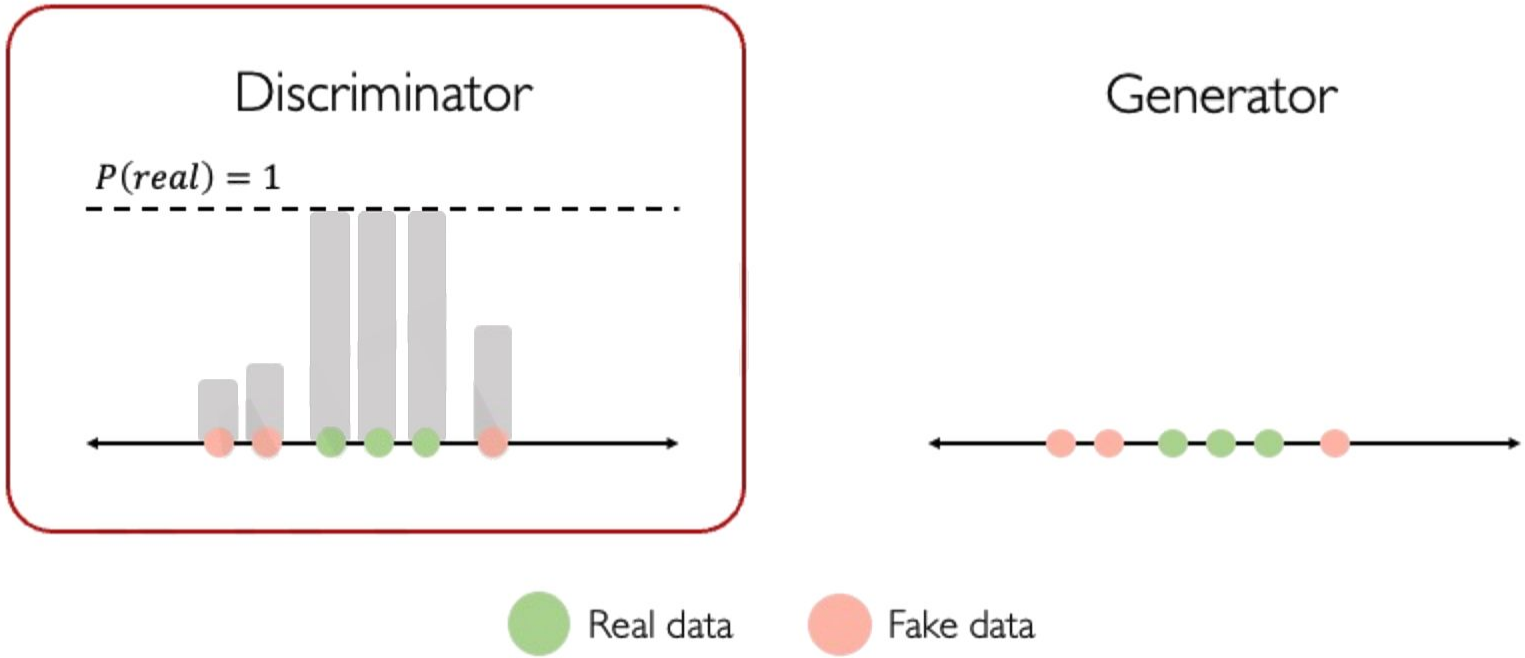
Intuition behind GAN

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



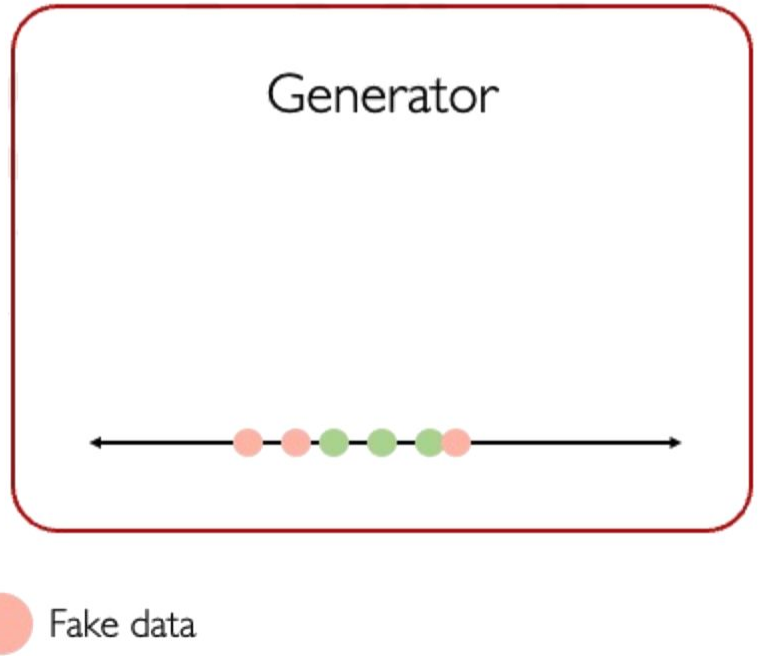
Intuition behind GAN

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



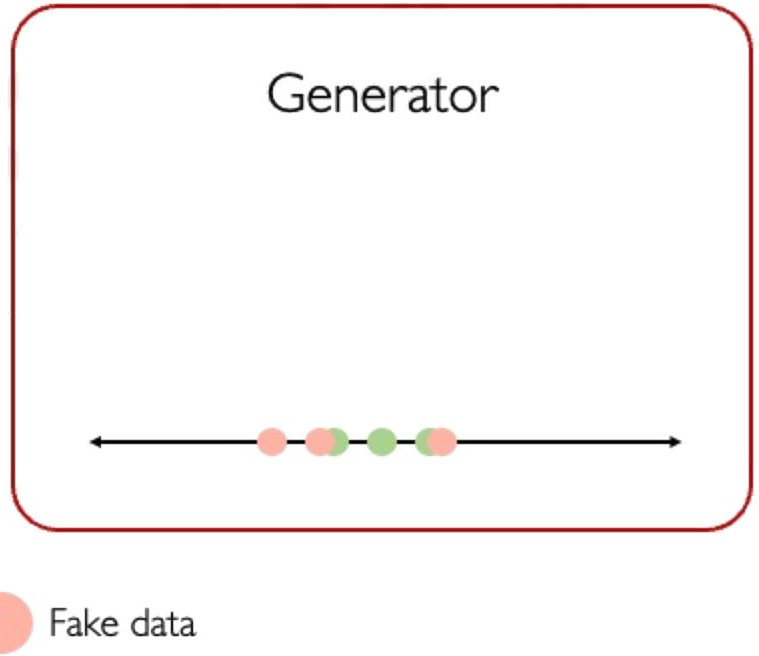
Intuition behind GAN

Generator tries to improve its imitation of the data.



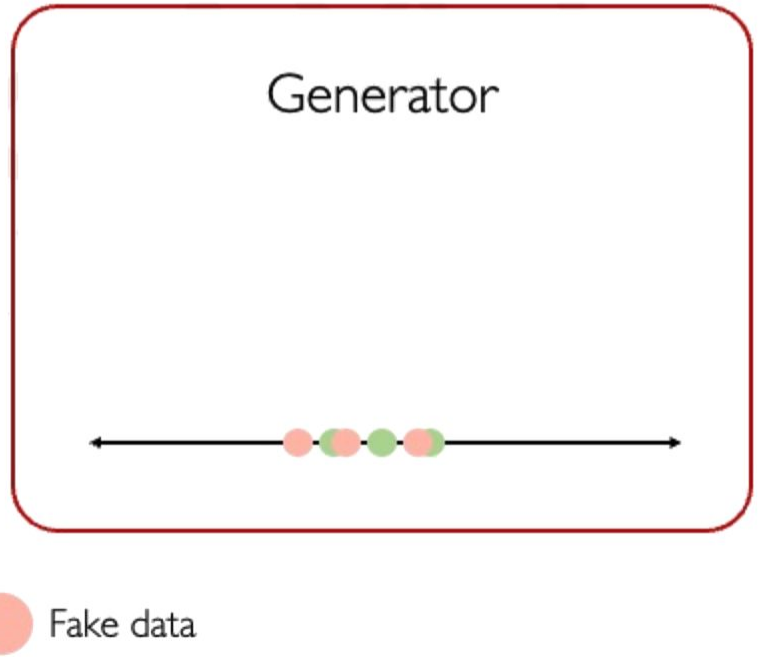
Intuition behind GAN

Generator tries to improve its imitation of the data.



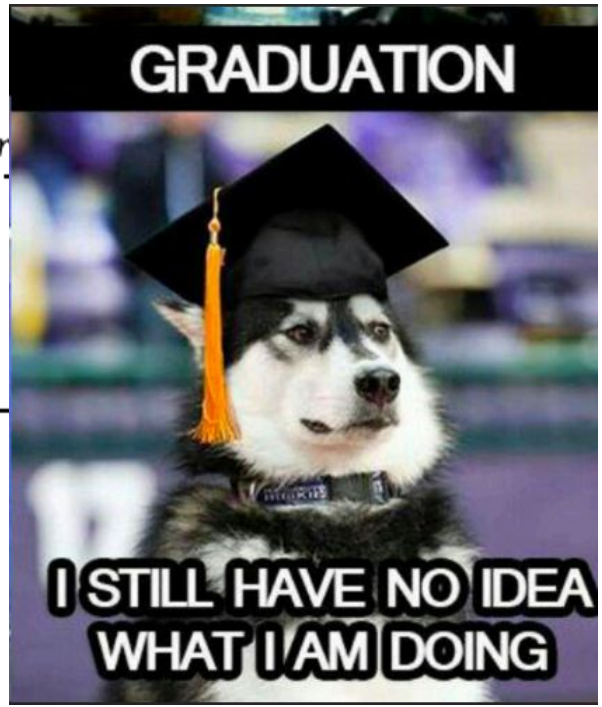
Intuition behind GAN

Generator tries to improve its imitation of the data.



Intuition behind GAN

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

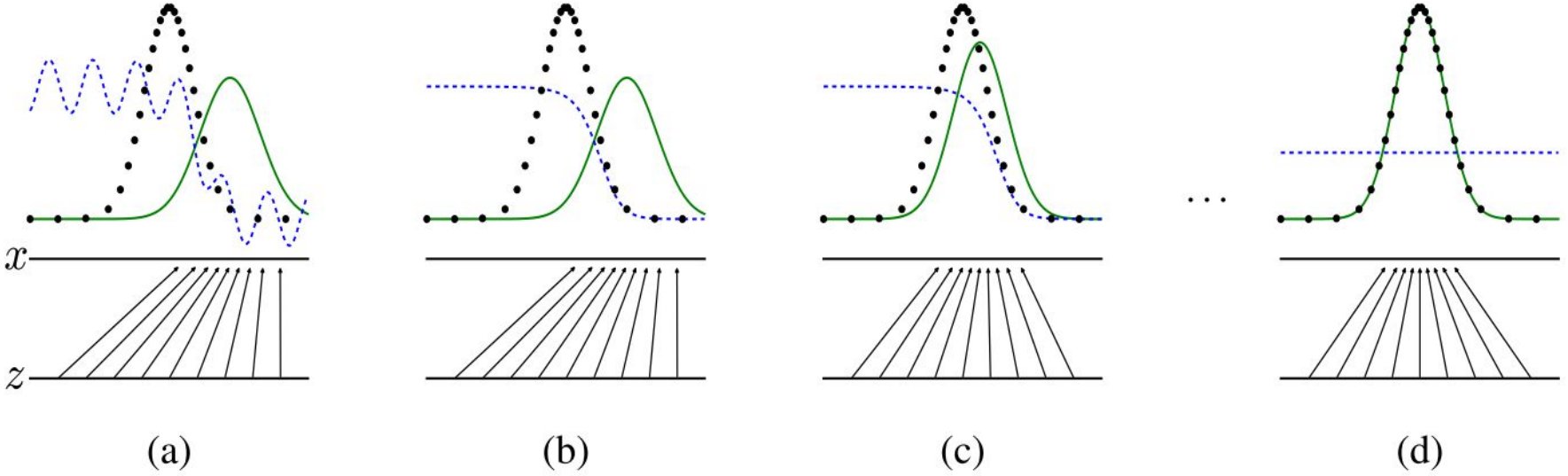


Generator

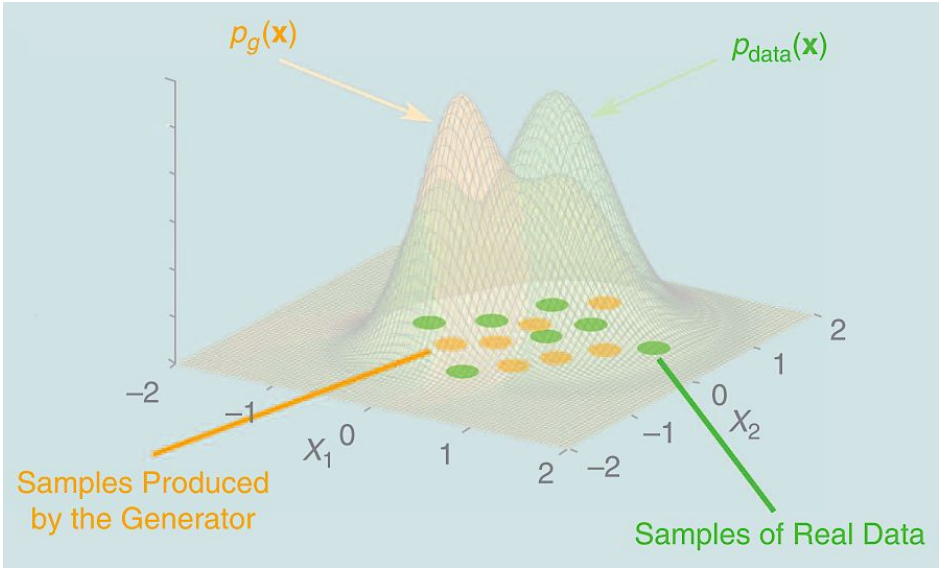
Fake data

Intuition behind GAN

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



AIM: Formally, given samples from real data distribution $P_{data}(x)$, generate realistic samples by sampling from a distribution $P_g(x)$, where $P_g(x)$ is the an approximation of $P_{data}(x)$.

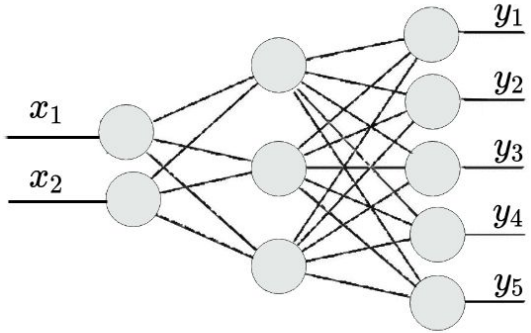


Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., & Bharath, A. A. (2018). Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, *35*(1), 53-65.

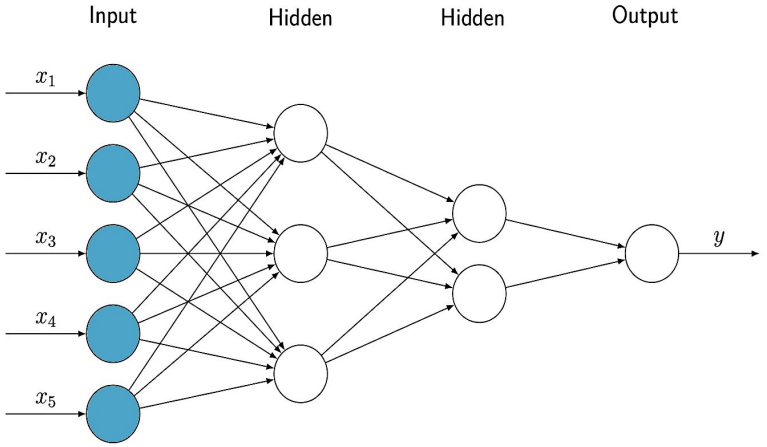
Vanilla GAN

Vanilla GAN (Illustration)

Generator

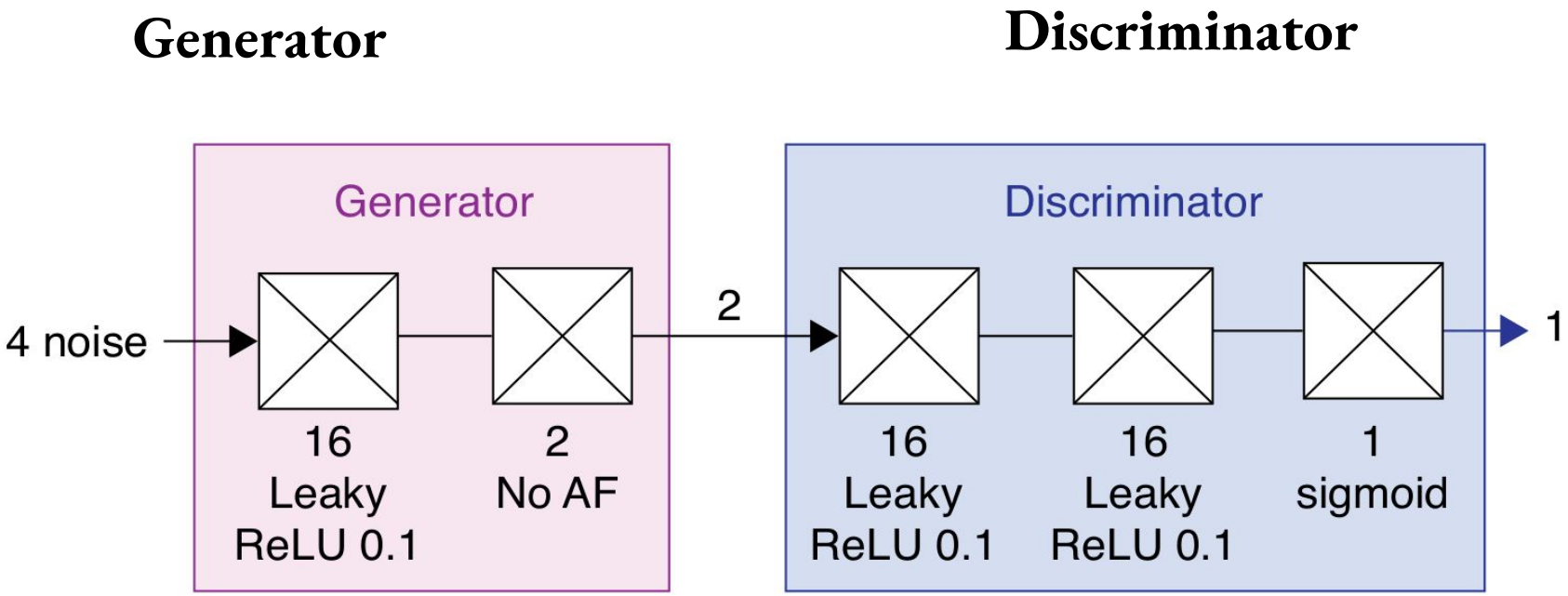


Discriminator



Generator and Discriminator are Deep Neural Networks (DNNs) with dense layers only.

Vanilla GAN Example for Gaussian



Generator and Discriminator are Deep Neural Networks (DNNs) with dense layers only.

Real Data

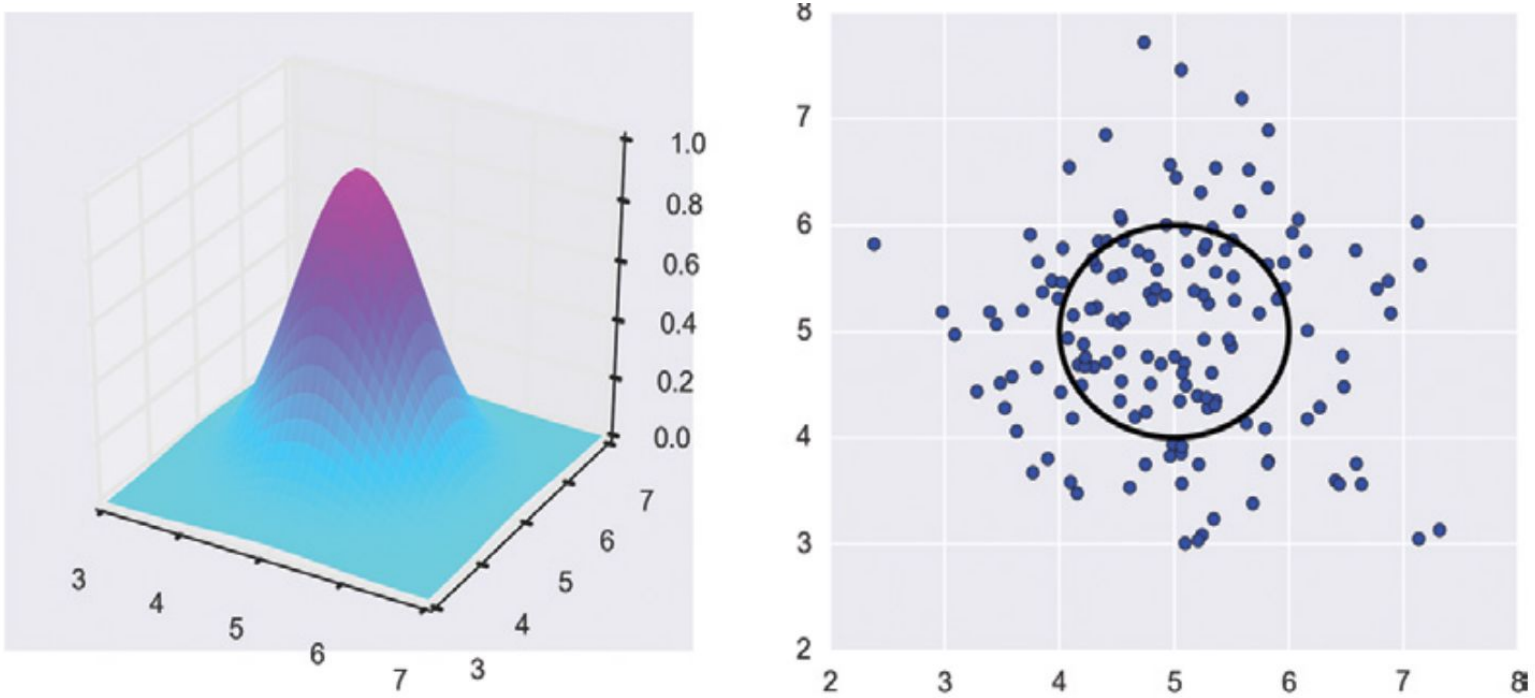


Figure 22-13: Our starting distribution is a Gaussian bump centered at (5,5) with a standard deviation of 1. Left: The blob in 3D. Right: A circle showing the location of one standard deviation of the blob in 2D, and some representative points randomly drawn from this distribution.

Identification of a point

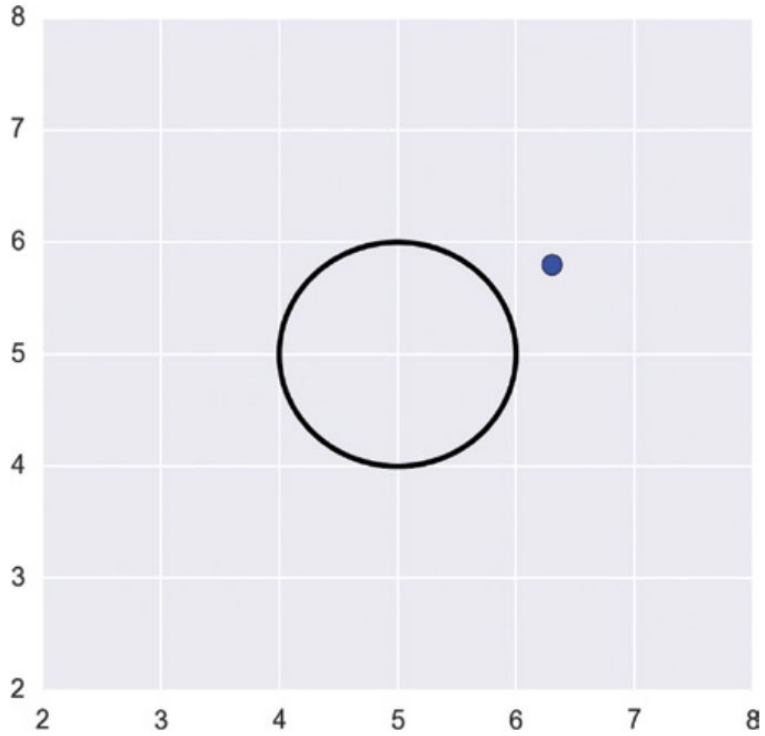


Figure 22-14: We have a single sample and we want to determine if it was drawn from the Gaussian distribution.

Fake Data

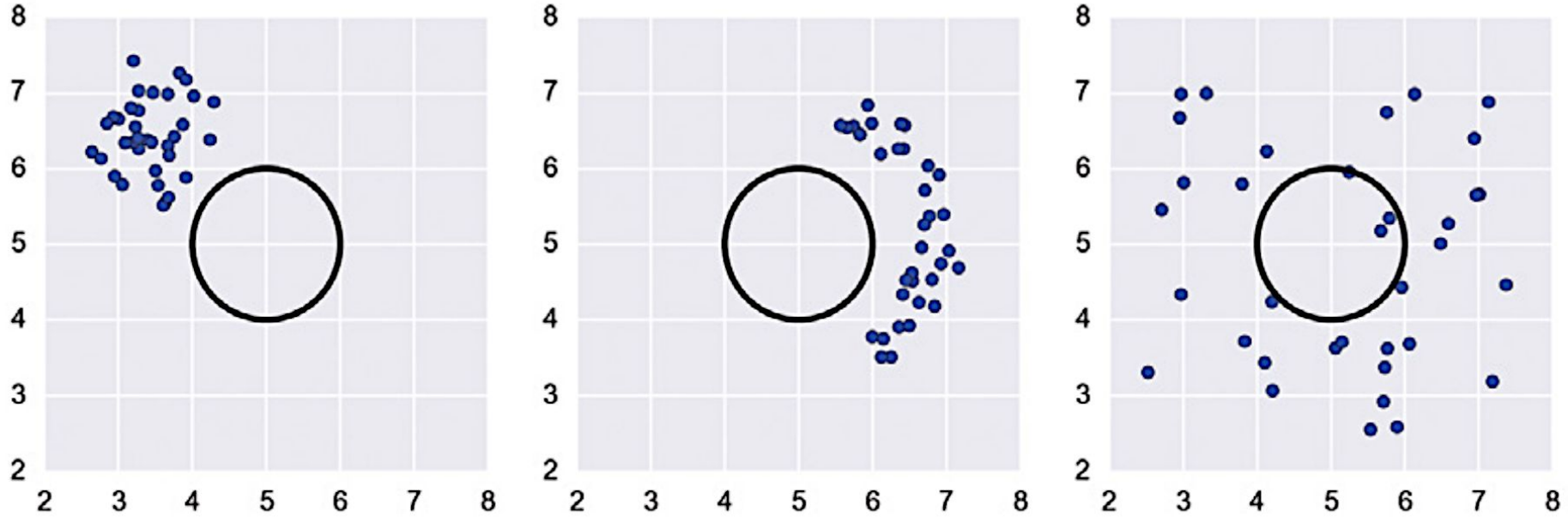
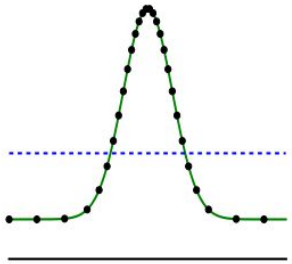


Figure 22-15: Some sets of points that are unlikely to have been the result of picking random values from our starting Gaussian

Training Procedure



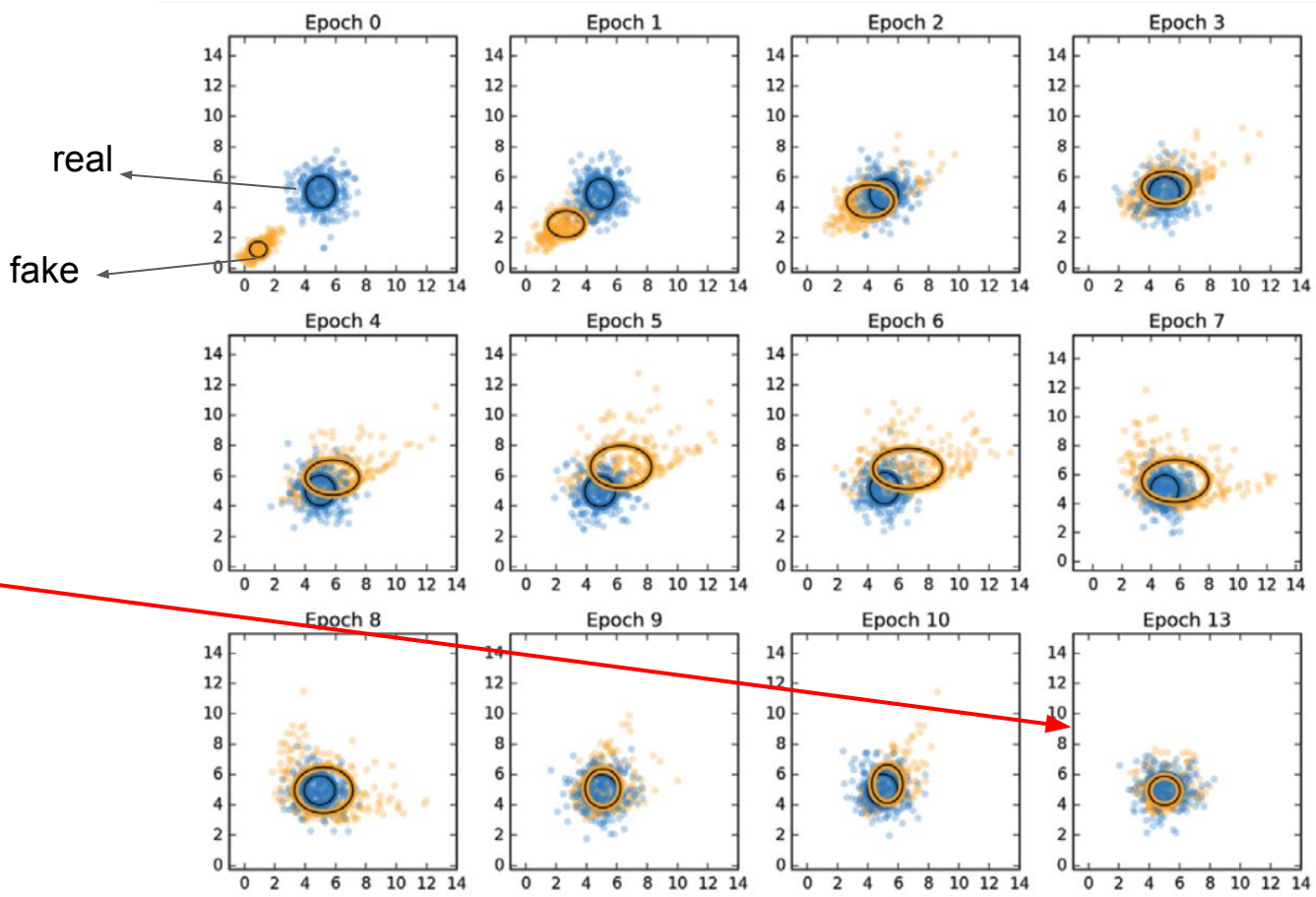
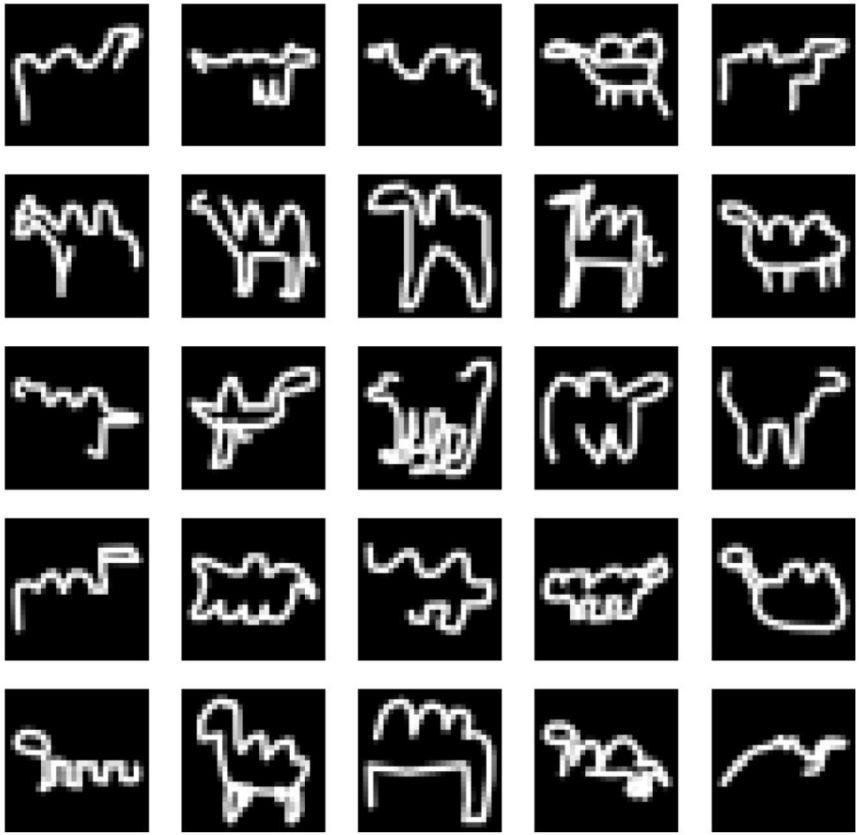


Figure 22-19: Our simple GAN in action. The blue points are the original dataset. The orange points were produced by the generator. Read the plots left to right, top to bottom. Epoch 0 refers to results after the first epoch of training.

Ganimal
(example application)

Ganimal (example application)

- Suppose Ganimal is a mythical creature that is said to roam around the jungle in the night.
- A a collection of nighttime photos of the Ganimal beast is only available.
- The figure shows some of the photos of Ganimal.



Ganimal

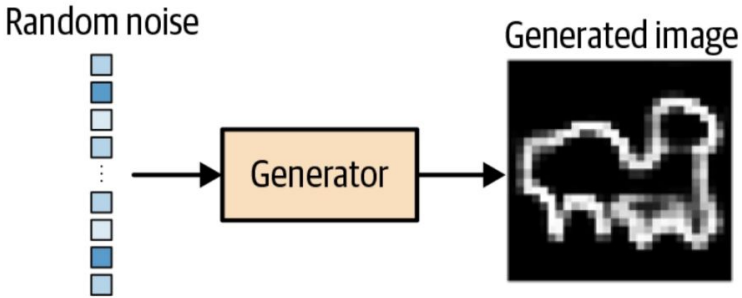
How to **synthesize new images of Ganimal ?**

By using GAN framework consisting of **Generator** and **Discriminator**

Figure 4-2. Samples of Gene's ganimal photography

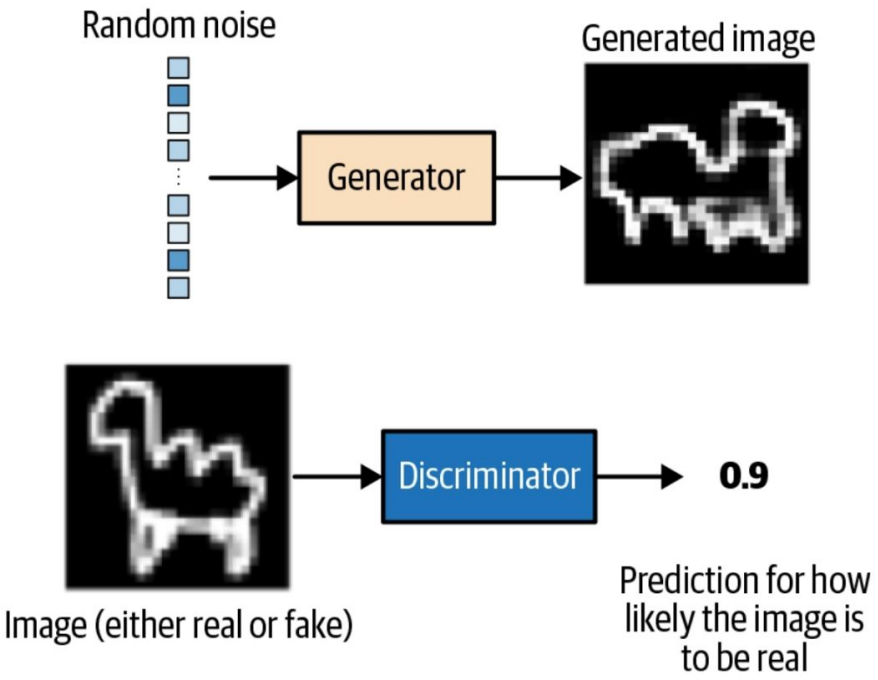
Generator and Discriminator

- Generator **G** tries to convert random noise into observations that look as if they have been sampled from the original dataset.

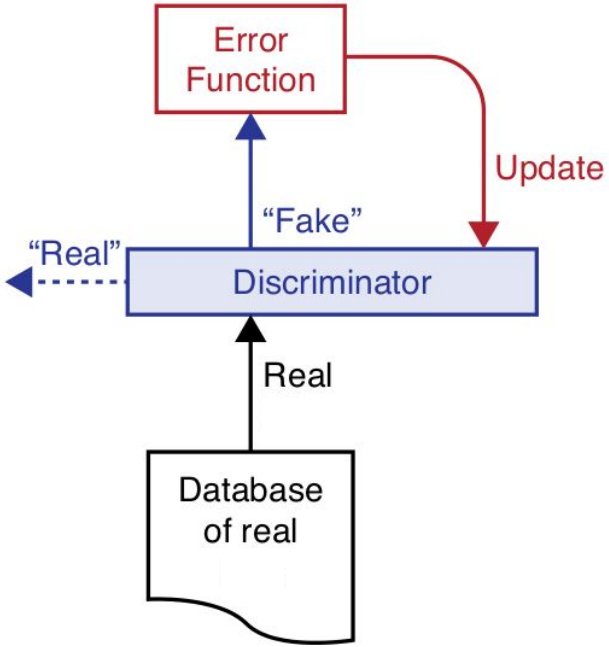


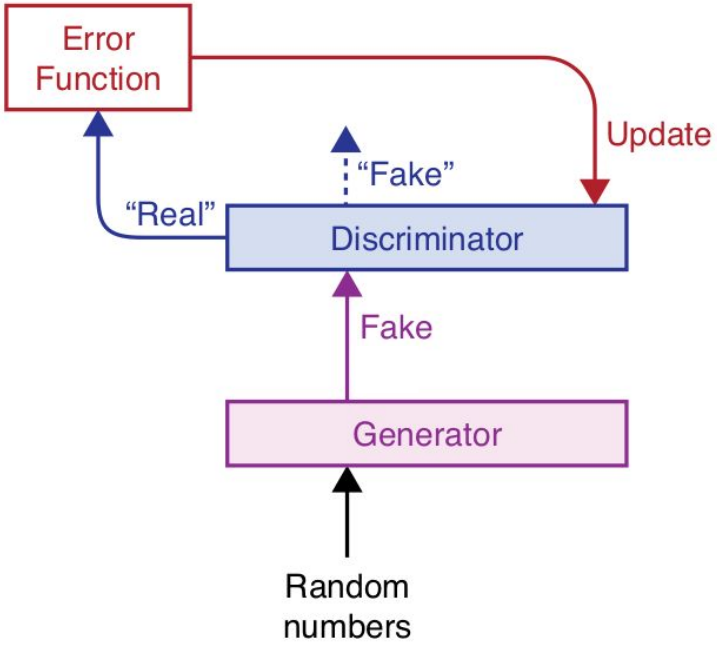
Generator and Discriminator

- Discriminator **D** tries to predict whether an observation comes from the original dataset or is one of the generator's forgeries.



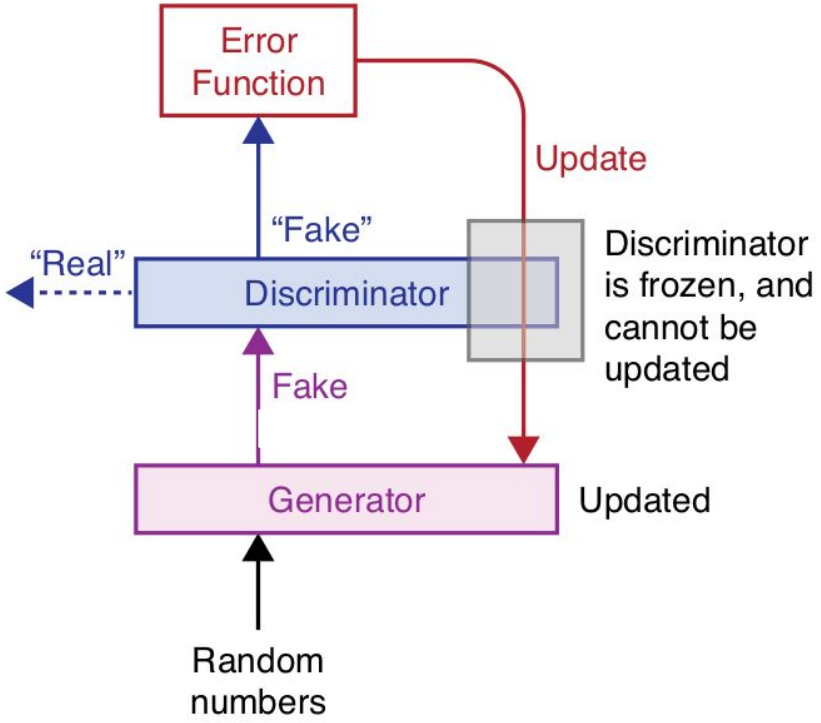
Training Discriminator





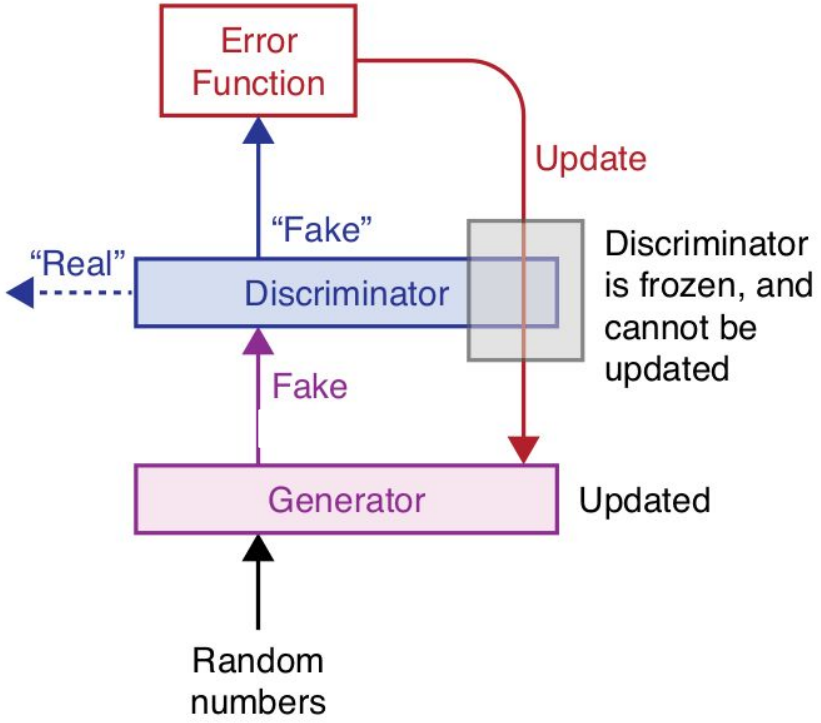
Training Generator

1. AIM of generator is to output realistic samples.
2. Note that **generator do not have access to real data samples.**



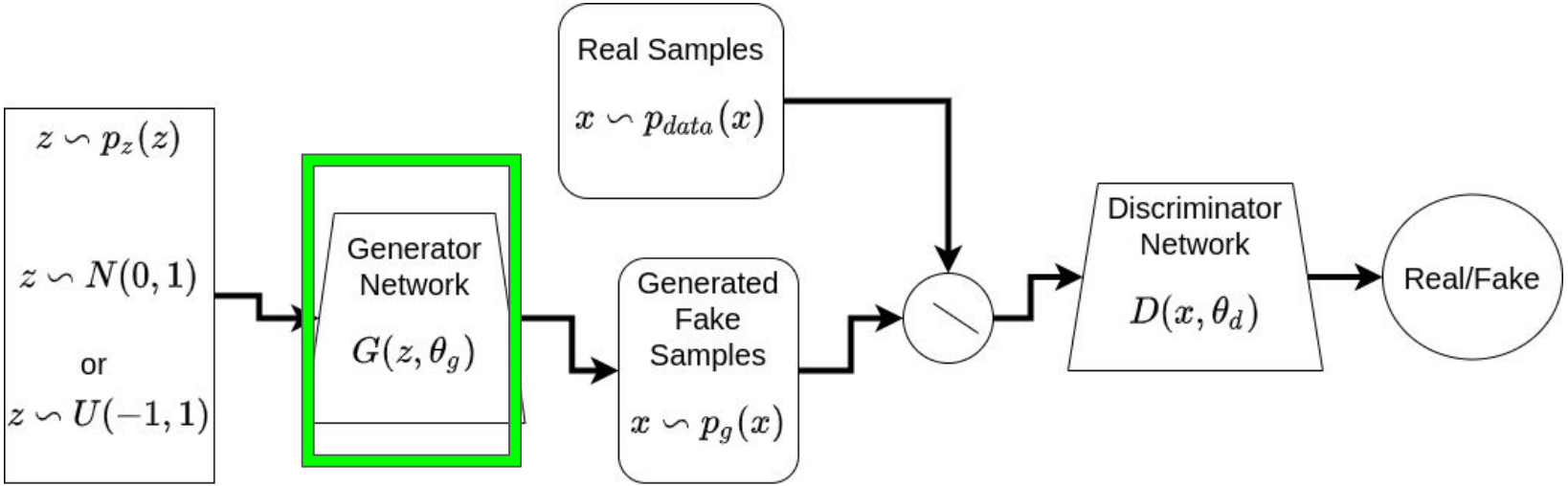
Training Generator

1. AIM of generator is to output realistic samples.
2. Note that generator do not have access to real data samples.
3. **Key idea**: GAN is based on the **"indirect" training** through the discriminator. Generator learns the distribution of real data samples through discriminator output.



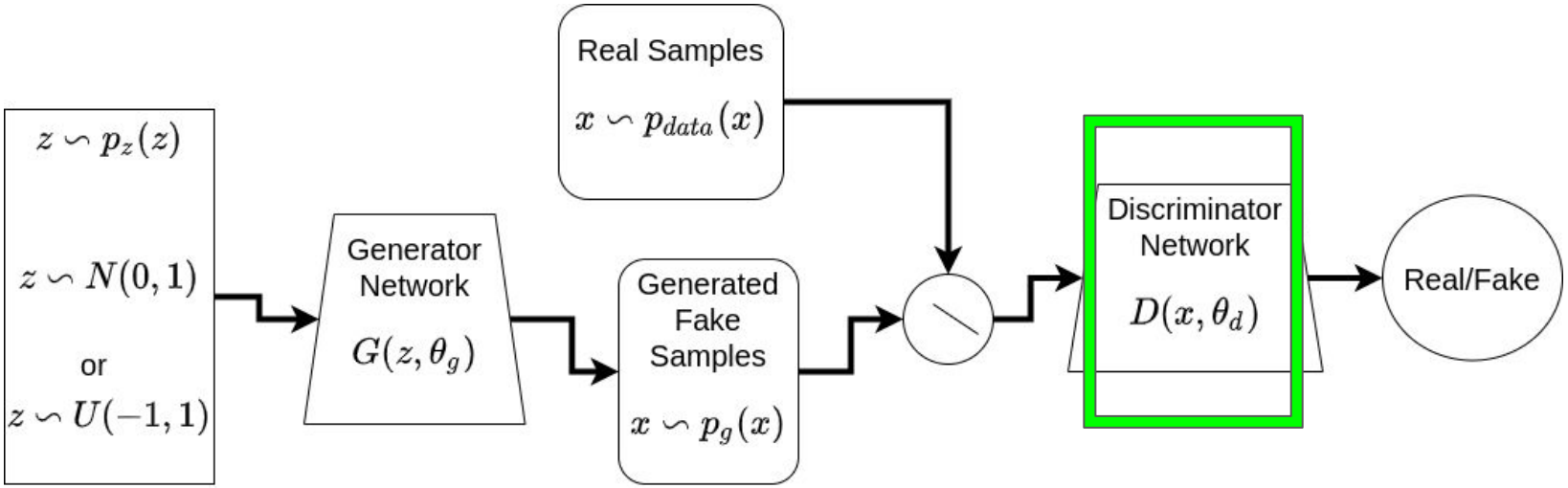
GANs
(Formal Description)

Formal Description



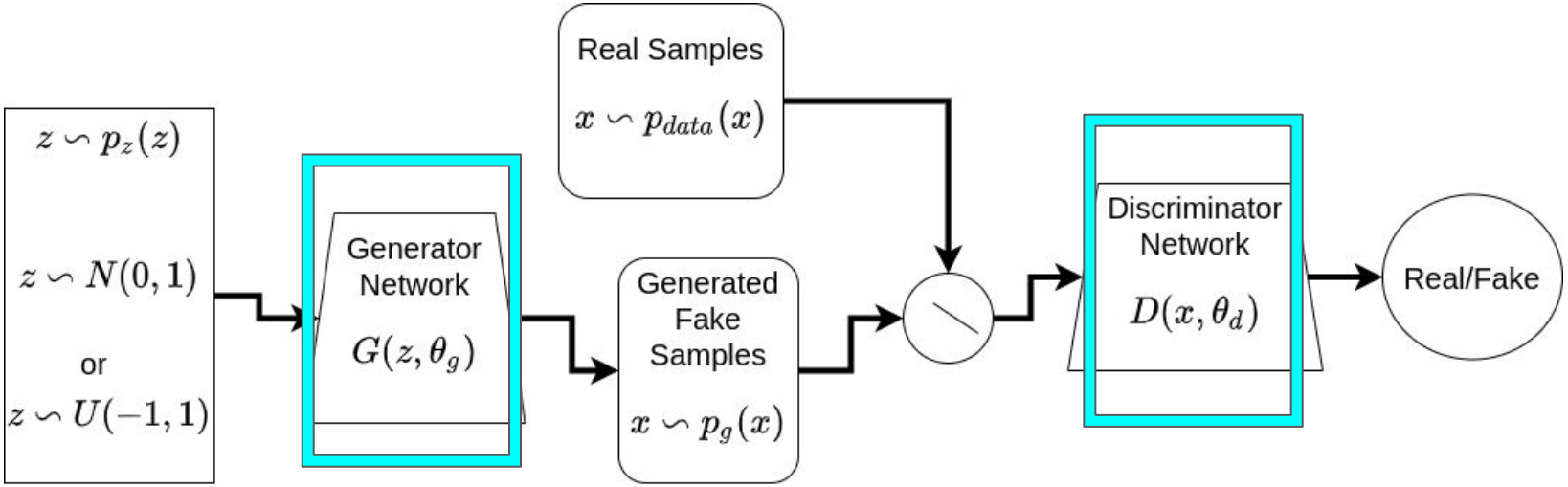
- Define a mapping to data space as G(z; $\boldsymbol{\theta}_g$), where G is the generator network.

Formal Description



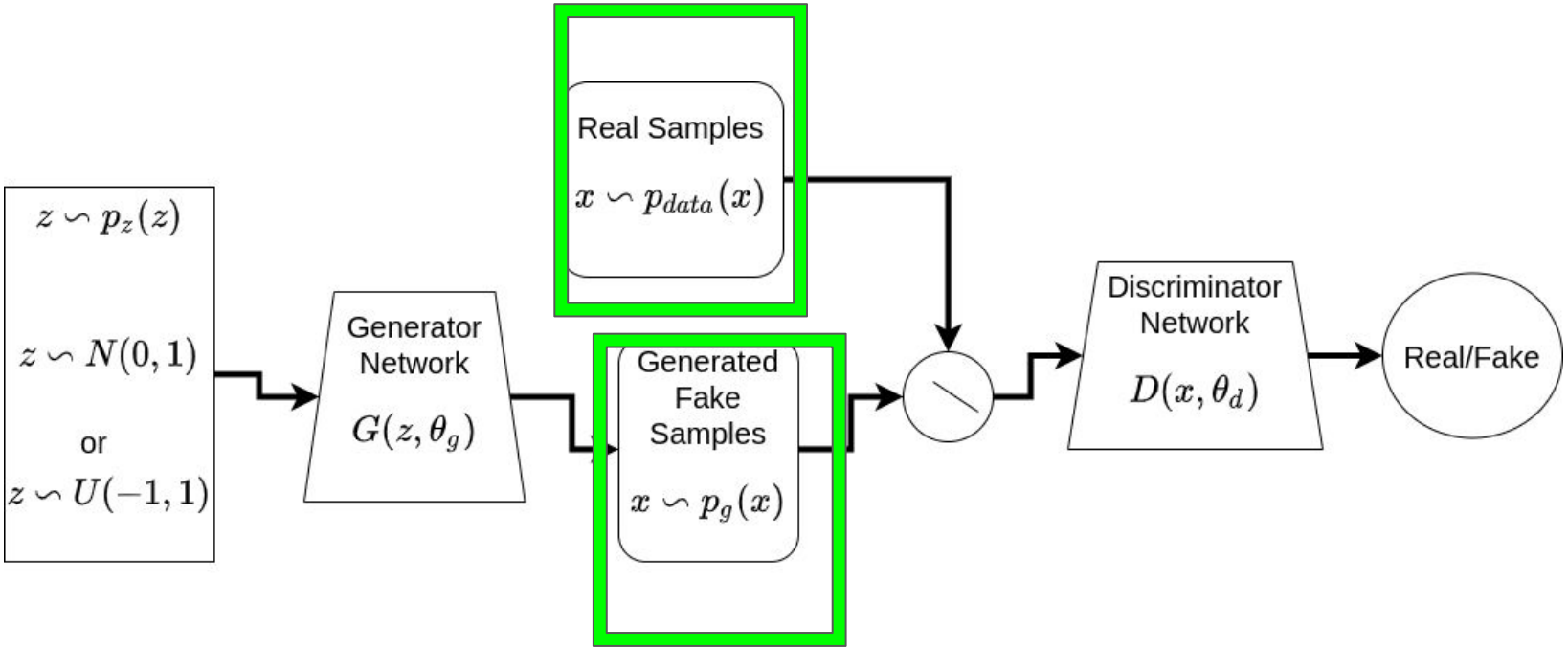
- Define a mapping D(x; $\boldsymbol{\theta}_{\mathbf{d}}$), that output a scalar, denoting the probability of x has come from real data distribution P_{data}( ).

Formal Description



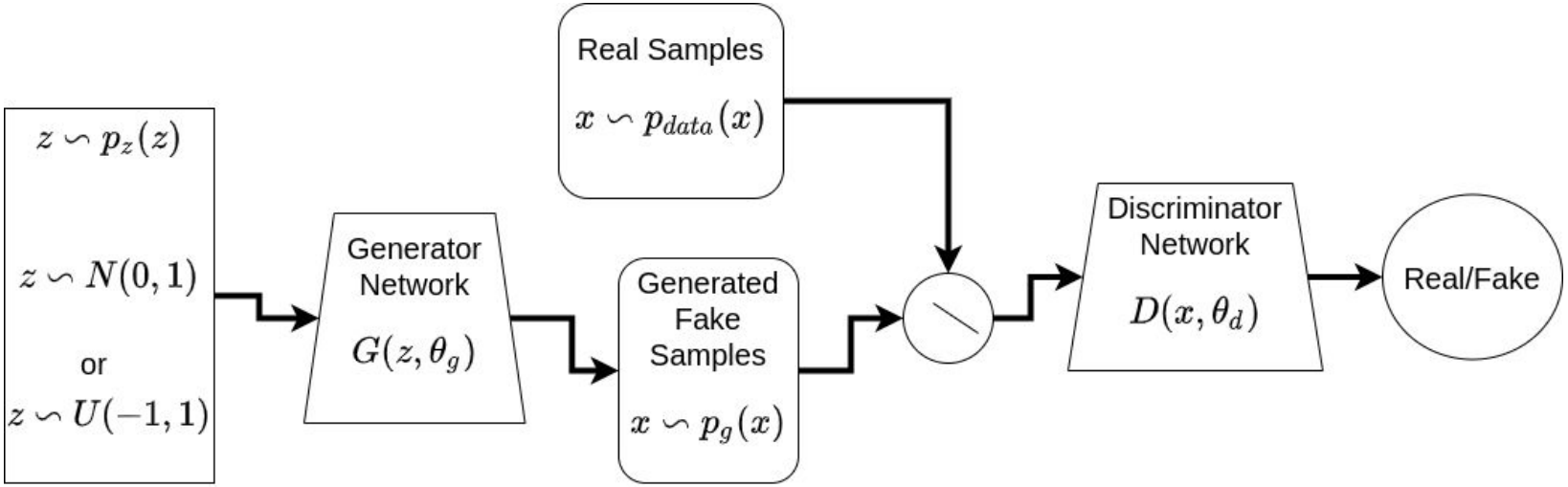
- **AIM** is to ***train*** G and D networks such that generator output realistic samples and discriminator is not able to distinguish between real samples and generated samples

Formal Description



Generated sample distribution $P_g(x)$ is a good approximation of real data distribution $P_{data}(x)$.

Formal Description

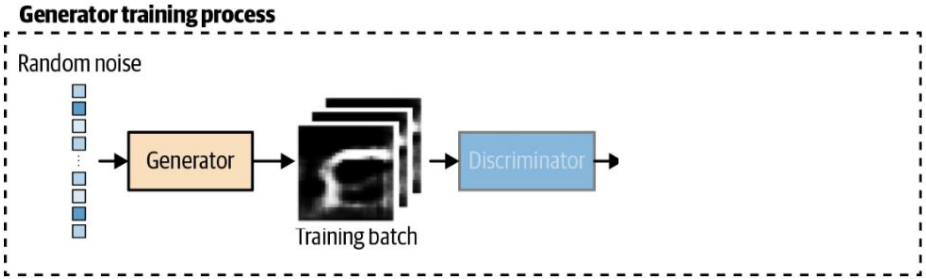


- Optimise the objective function **V** for $\boldsymbol{\theta}_g$ and $\boldsymbol{\theta}_d$.

$$\max_{\mathcal{D}} \min_{\mathcal{G}} V(\mathcal{G}, \mathcal{D})$$

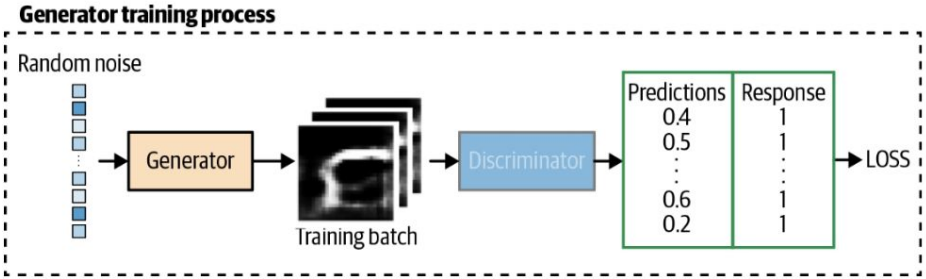
Generator Training Process

1. We start with a random latent vector, z, and feed it through the generator to get $G(z; \theta_g)$.
2. Then we feed $x = G(z; \theta_g)$ into discriminator to produce the output $D(G(z; \theta_g))$.



Generator Training Process

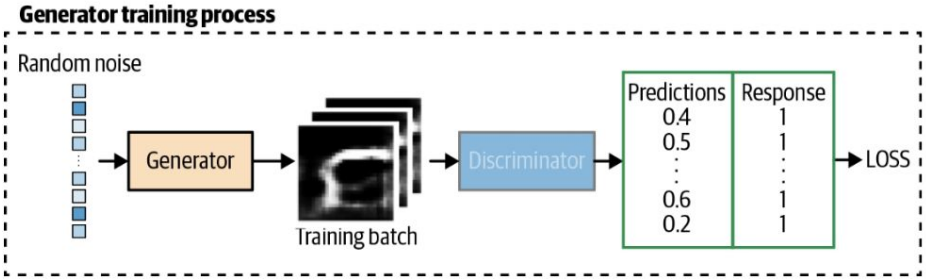
1. We start with a random latent vector, z, and feed it through the generator to get $G(z; \theta_g)$.
2. Then we feed $x = G(z; \theta_g)$ into discriminator to produce the output $D(G(z; \theta_g))$.
3. Next, minimize the **Generator loss** and update weights θ_g.
4. Note that the discriminator weights θ_d are locked and we only update θ_g.



$$\arg\min_G \log\left(1 - \qquad\qquad\right))$$

Generator Training Process

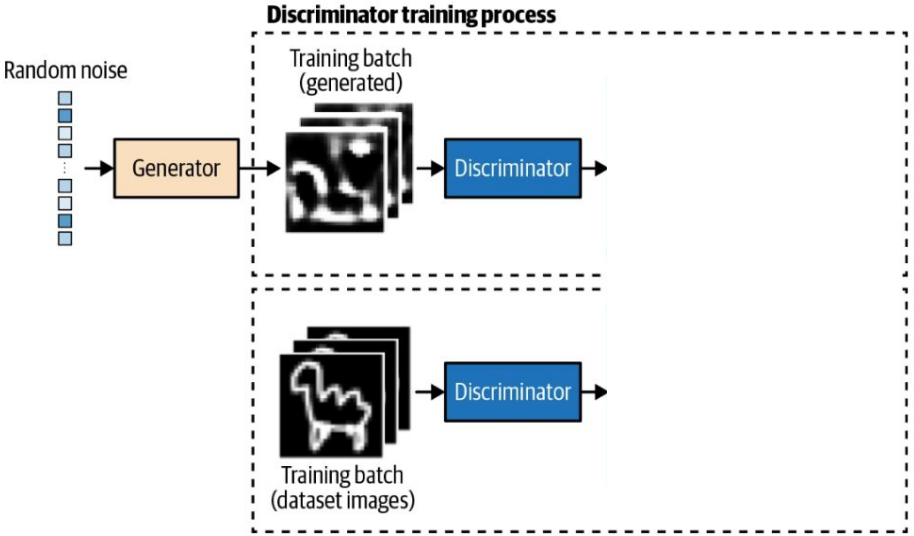
1. We start with a random latent vector, z, and feed it through the generator to get $G(z; \theta_g)$.
2. Then we feed $x = G(z; \theta_g)$ into discriminator to produce the output $D(G(z; \theta_g))$.
3. Next, minimize the **Generator loss** and update weights θ_g.
4. Note that the discriminator weights θ_d are locked and we only update θ_g.



$$\arg\min_{G} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

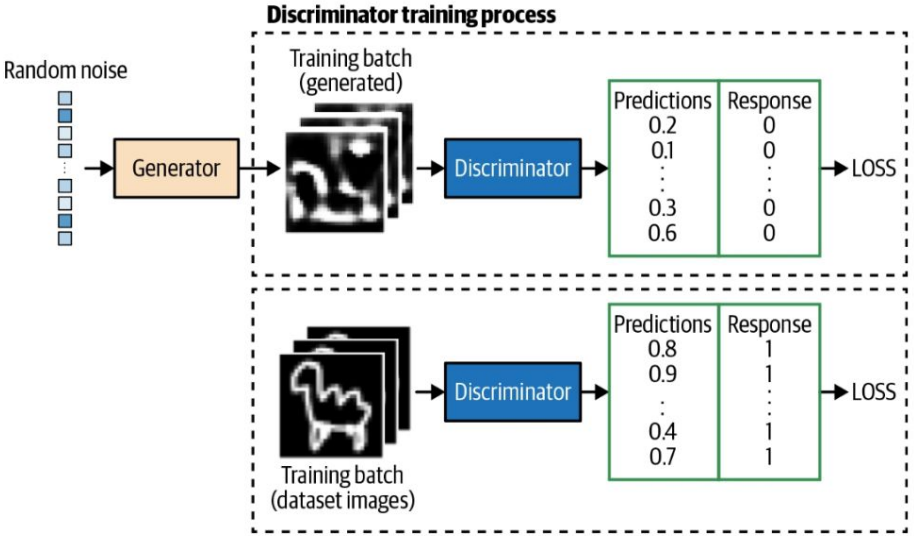
Discriminator Training Process

1. Based on input, training proceed as below:
   - **Real**: Select sample from the real data, $x \sim p_{data}(x)$, and use it to produce $D(x; \theta_d)$.
   - **Fake**: Generate a fake sample $x \sim p_g(x)$, start with a random vector z, to generate sample $G(z; \theta_g)$. Then, compute final output $D(G(z))$.



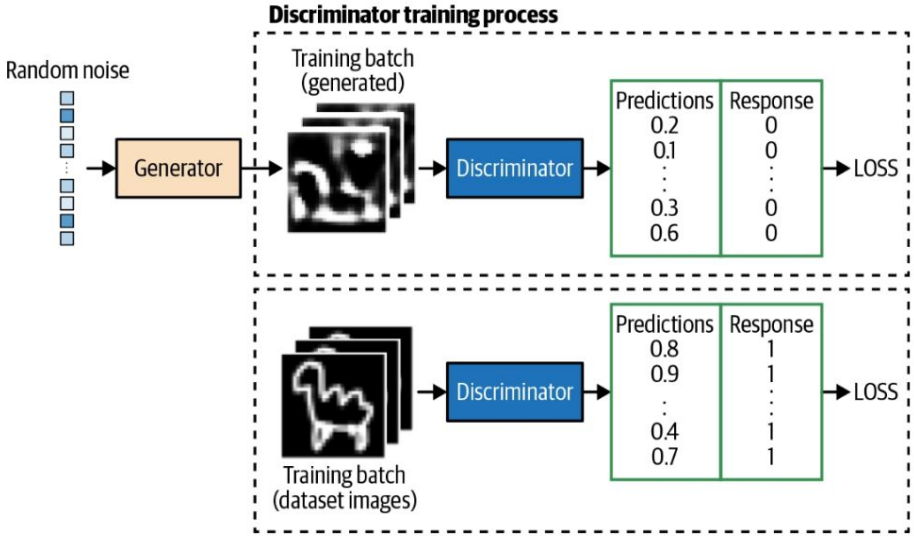
Discriminator Training Process

1. Based on input, training proceed as below:
    - **Real**: Select sample from the real data, $x \sim p_{data}(x)$, and use it to produce $D(x; \theta_d)$.
    - **Fake**: Generate a fake sample $x \sim p_g(x)$, start with a random vector z, to generate sample $G(z; \theta_g)$. Then, compute final output $D(G(z))$.
2. Next, compute **<u>Discriminator loss</u>** to update the weights θ_d.
3. Note that the generator weights θ_g, will be locked.



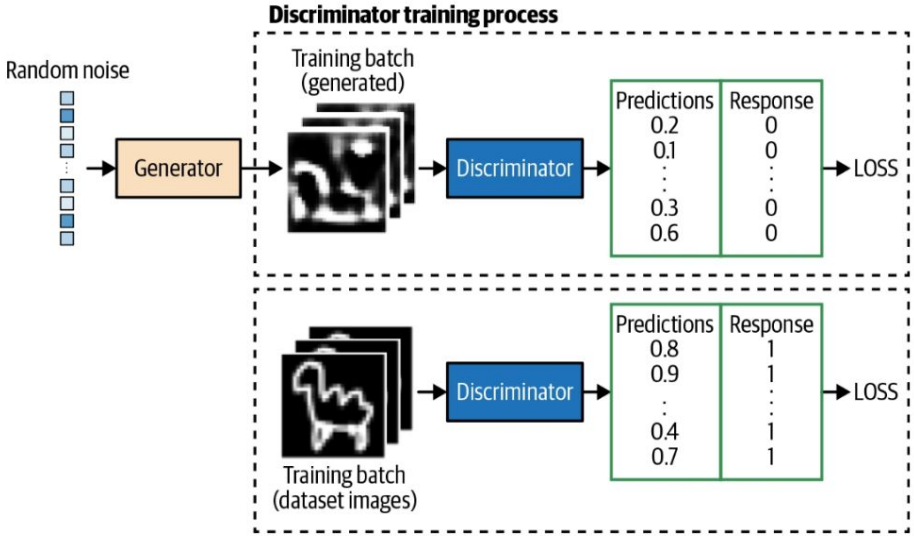
Discriminator Training Process

1. Based on input, training proceed as below:
   - **Real**: Select sample from the real data, $x \sim p_{data}(x)$, and use it to produce $D(x; \theta_d)$.
   - **Fake**: Generate a fake sample $x \sim p_g(x)$, start with a random vector z, to generate sample $G(z; \theta_g)$. Then, compute final output $D(G(z))$.
2. Next, compute **Discriminator loss** to update the weights θ_d.
3. Note that the generator weights θ_g, will be locked.



$$\arg\max_D \Big[\qquad + \qquad \Big]$$

Discriminator Training Process

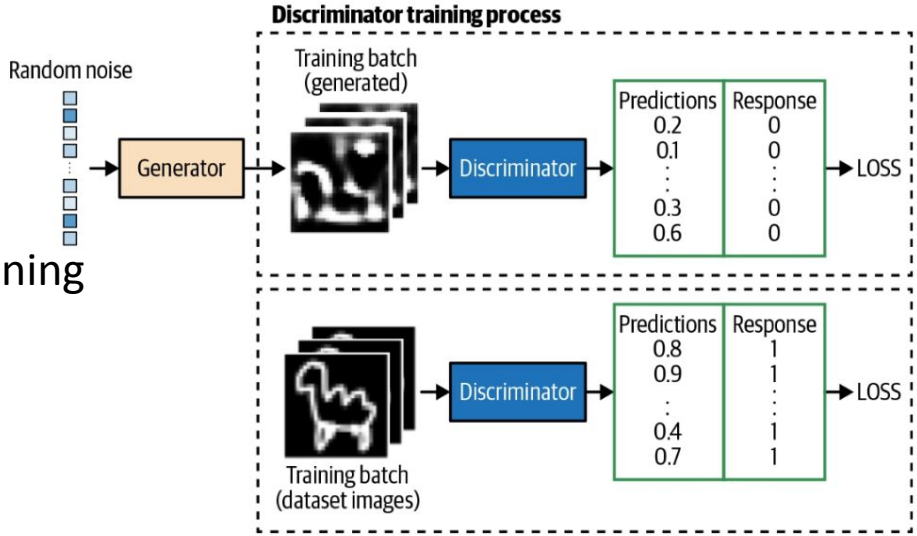


$$\arg\max_{D} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$$

Discriminator Training Process

Discriminator Loss:

- D aims to maximize the probability of assigning the correct label to both real training examples and samples from G.

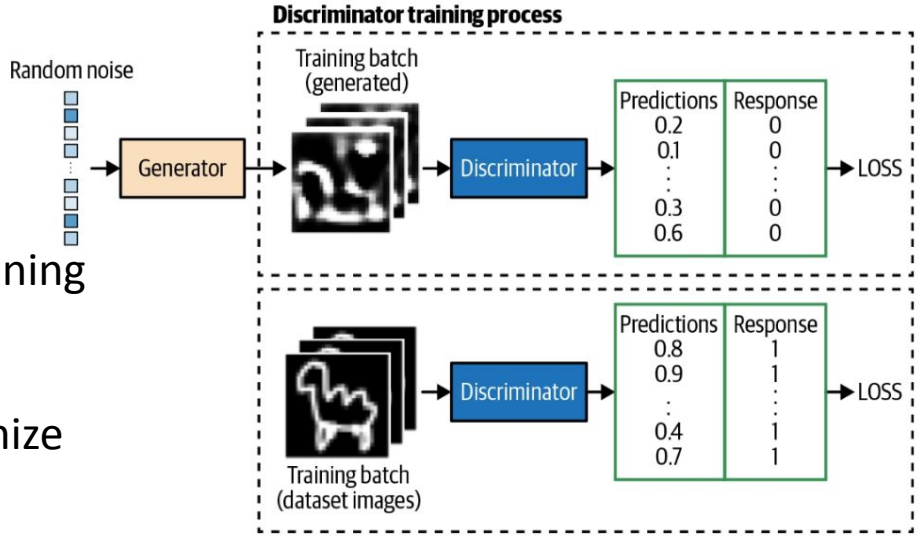


$$\arg\max_{D} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$$

Discriminator Training Process

Discriminator Loss:

- D aims to maximize the probability of assigning the correct label to both real training examples and samples from G.
- Formally, the discriminator seeks to maximize the average of the log probability for real images and the log of the inverted probabilities of fake images.



$$\arg\max_{D} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$$

Putting Everything Together

for number of training iterations **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

Goodfellow, Ian, et al. "Generative adversarial nets." *Advances in neural information processing systems* 27 (2014).

Putting Everything Together (Minimax GAN Loss)

for number of training iterations **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

$$\max_{\mathcal{D}} \min_{\mathcal{G}} V(\mathcal{G}, \mathcal{D})$$

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

Goodfellow, Ian, et al. "Generative adversarial nets." *Advances in neural information processing systems* 27 (2014).

Putting Everything Together (Minimax GAN Loss)

for number of training iterations **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator

$$\mathbb{E}_{x \sim P_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim P_z} [\log(1 - D(G(z)))]$$

$$\max_{\mathcal{D}} \min_{\mathcal{G}} V(\mathcal{G}, \mathcal{D})$$

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator

$$\mathbb{E}_{z \sim P_z} [\log(1 - D(G(z)))]$$

end for

Goodfellow, Ian, et al. "Generative adversarial nets." *Advances in neural information processing systems* 27 (2014).

Minimax objective

- The following minimax objective applied for training G and D models jointly via solving:

$$\min_{\theta_G} \max_{\theta_D} V(G, D) = \min_{G} \max_{D} \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log(1 - D(G(z)))].$$

- $V(G, D)$ is a binary cross entropy function, commonly used in binary classification problems.

- In practice, above equation is solved by alternating the following two gradient updates:

$$\theta_D^{t+1} = \theta_D^t + \lambda^t \nabla_{\theta_D} V\left(D^t, G^t\right) \text{ and } \theta_G^{t+1} = \theta_G^t + \lambda^t \nabla_{\theta_G} V\left(D^{t+1}, G^t\right),$$

- where θ_G is the parameter of G, θ_D is the parameter D, λ is the learning rate, and t is the iteration number.

Minimax objective

- The second term in Minimax objective, $\log(1-D(G(z)))$, saturates and makes insufficient gradient flow through G, i.e., gradients value gets smaller and stop learning.

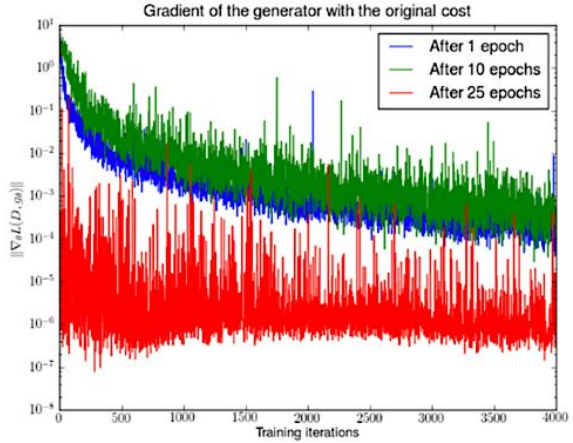


Fig. 6. DCGANs for 1, 10, and 25 epochs. G is fixed while D is trained from scratch and using original cost function to compute the gradients

~~Minimax objective~~

To overcome the vanishing gradient problem, the Minimax objective is reframed into two separate objectives:

$$\max_{\theta_D} \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log(1 - D(G(z)))] \text{ and } \max_{\theta_G} \mathbb{E}_{z \sim p_z} [\log(D(G(z)))].$$

- Moreover, G's gradient for these two separate objectives have the same fixed points and are always trained in the same direction.
- After cost computation in above Equation, Backpropagation can be used for updating the model parameters.

~~Minimax objective~~

To overcome the vanishing gradient problem, the Minimax objective is reframed into two separate objectives:

$$\max_{\theta_D} \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log(1 - D(G(z)))] \text{ and } \max_{\theta_G} \mathbb{E}_{z \sim p_z} [\log(D(G(z)))].$$

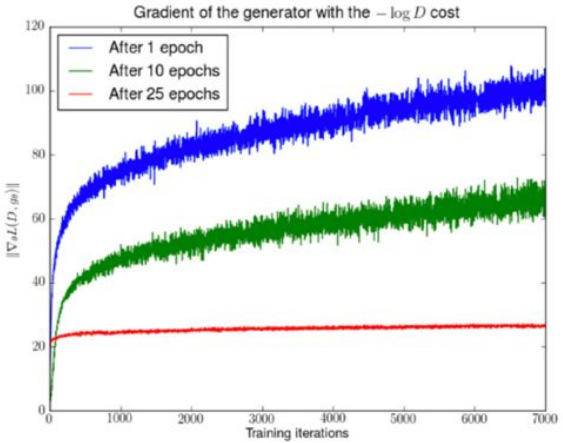


Fig. 7. DCGANs for 1, 10, and 25 epochs. G is fixed, D is trained from scratch and using − log D cost function to compute the gra-dients

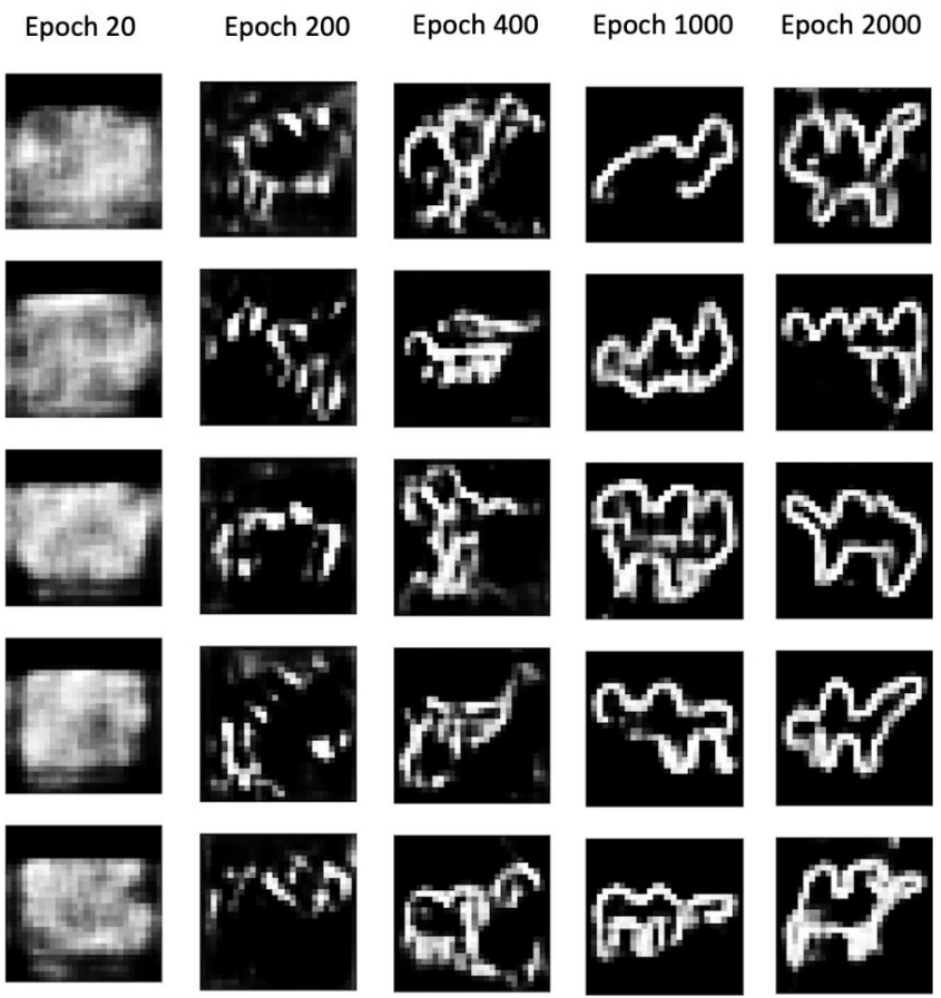
~~Minimax objective~~

Because of these two different objectives, the update rule is given as:

$$\left\{\theta_D^{t+1}, \theta_G^{t+1}\right\} \leftarrow \begin{cases} \text{Update} & \text{if } \mathrm{D}(\mathrm{x}) \text{ predicts wrong} \\ \text{Update} & \text{if } \mathrm{D}(\mathrm{G}(\mathrm{z})) \text{ predicts wrong} \\ \text{Update} & \text{if } \mathrm{D}(\mathrm{G}(\mathrm{z})) \text{ predicts correct} \end{cases} .$$

- If D and G are given enough capability with sufficient training iterations, then G can convert a simple latent distribution p_g to more complex distributions, i.e., p_g converges to p_{data}, such as $p_g = p_{\text{data}}$.

Output from the Generator at specific epochs during training



Generated Images and Training Set

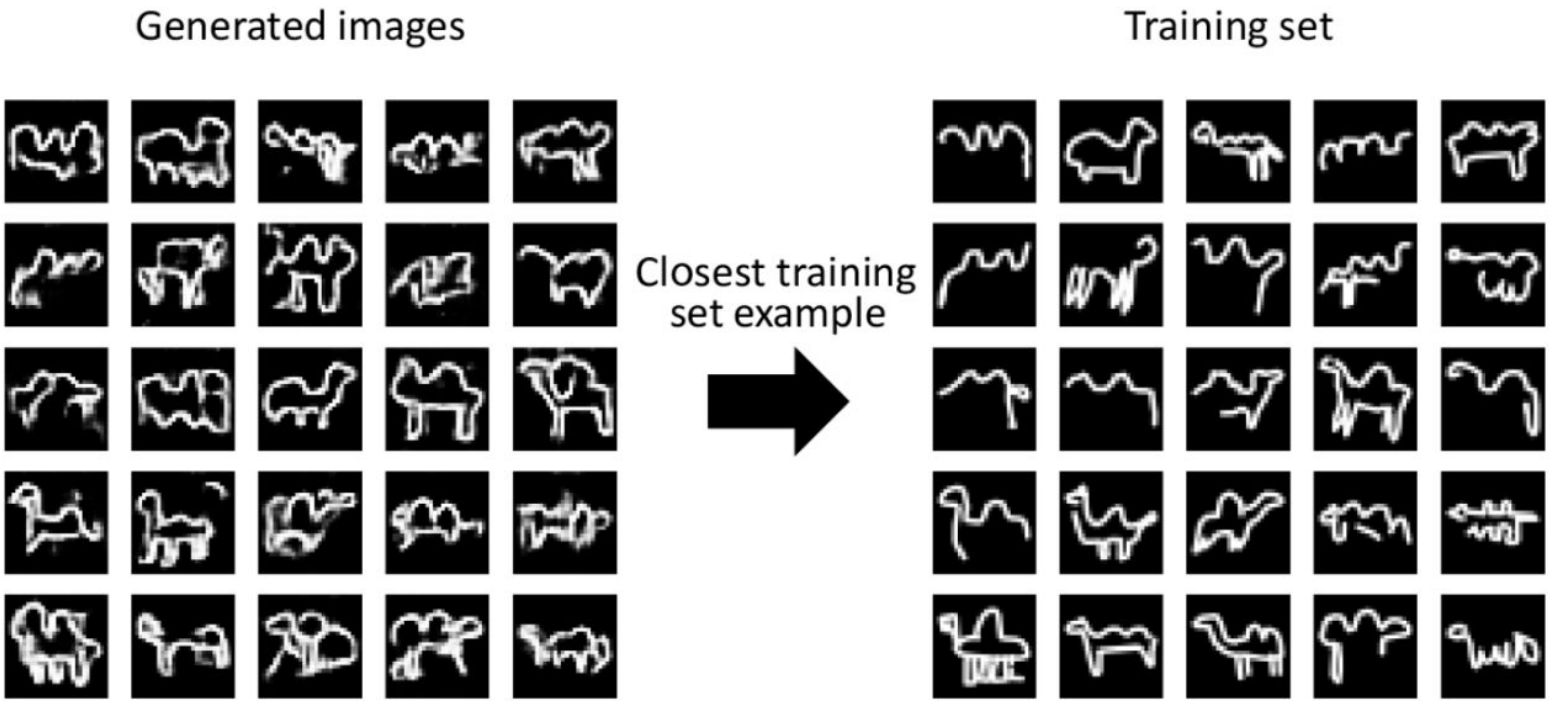


Figure 4-10. Closest matches of generated images from the training set

References

- Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play. By David Foster. O'Reilly Media.
- Creswell, Antonia, et al. "Generative adversarial networks: An overview." IEEE Signal Processing Magazine (2018).
- Advanced Deep Learning with Python. By Ivan Vasilev. Packt Publishing Ltd.

DCGAN

Deep Convolutional GAN (DCGAN)

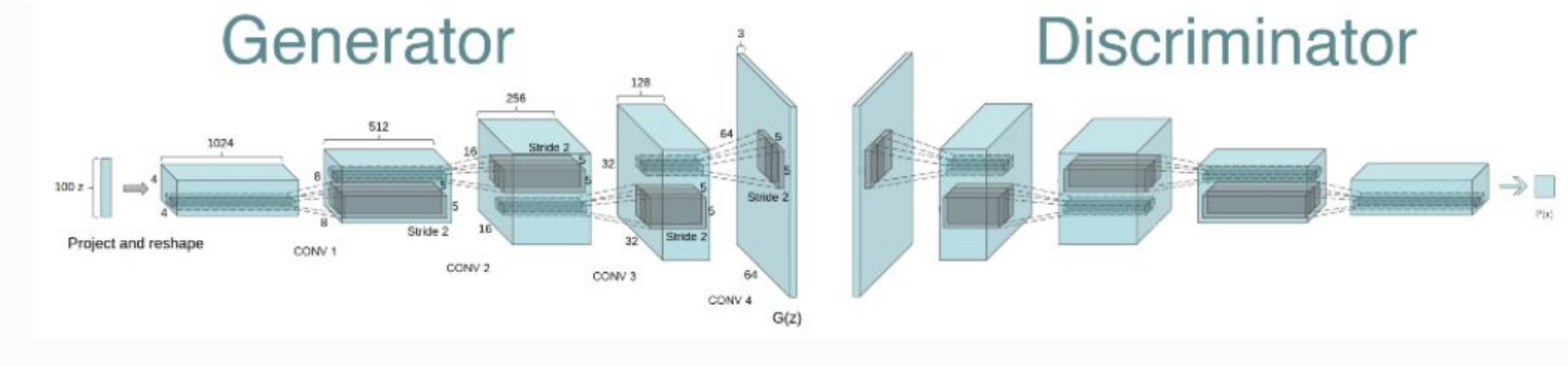
- DCGANs extend the idea of Vanilla GAN using convolutional layers.
- The generator is a DNN with convolutional layers, and the discriminator is a DNN with convolutional layers.

LSUN Bedroom Dataset

Figure 3: Generated bedrooms after five epochs of training. There appears to be evidence of visual under-fitting via repeated noise textures across multiple samples such as the base boards of some of the beds.

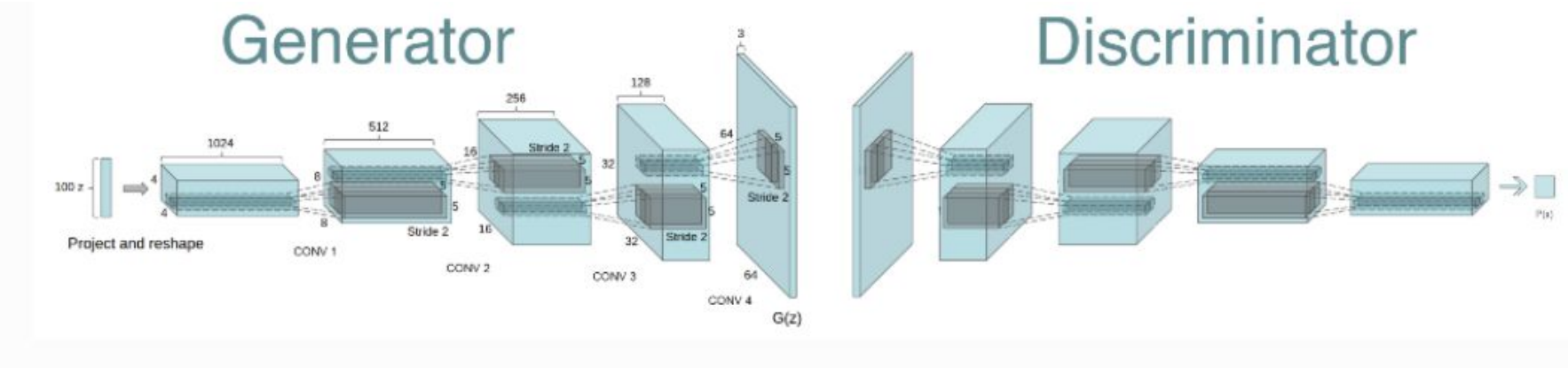
Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

DCGAN



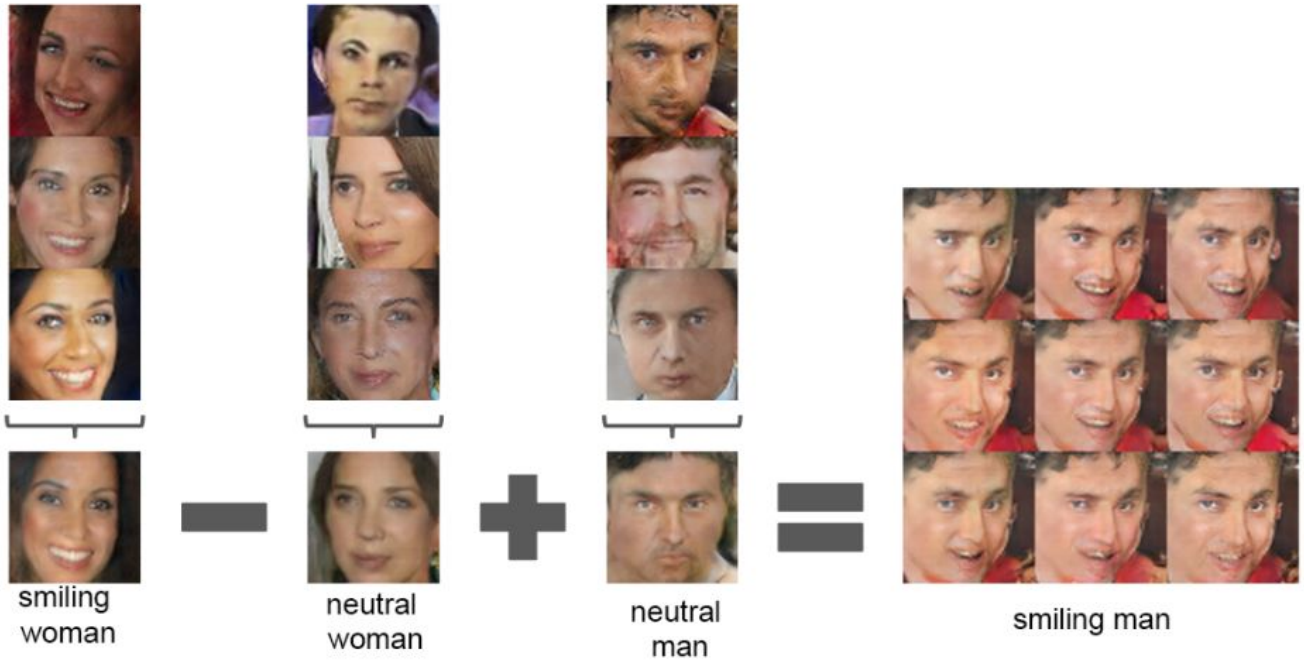
https://gluon.mxnet.io/chapter14_generative-adversarial-networks/dcgan.html

DCGAN



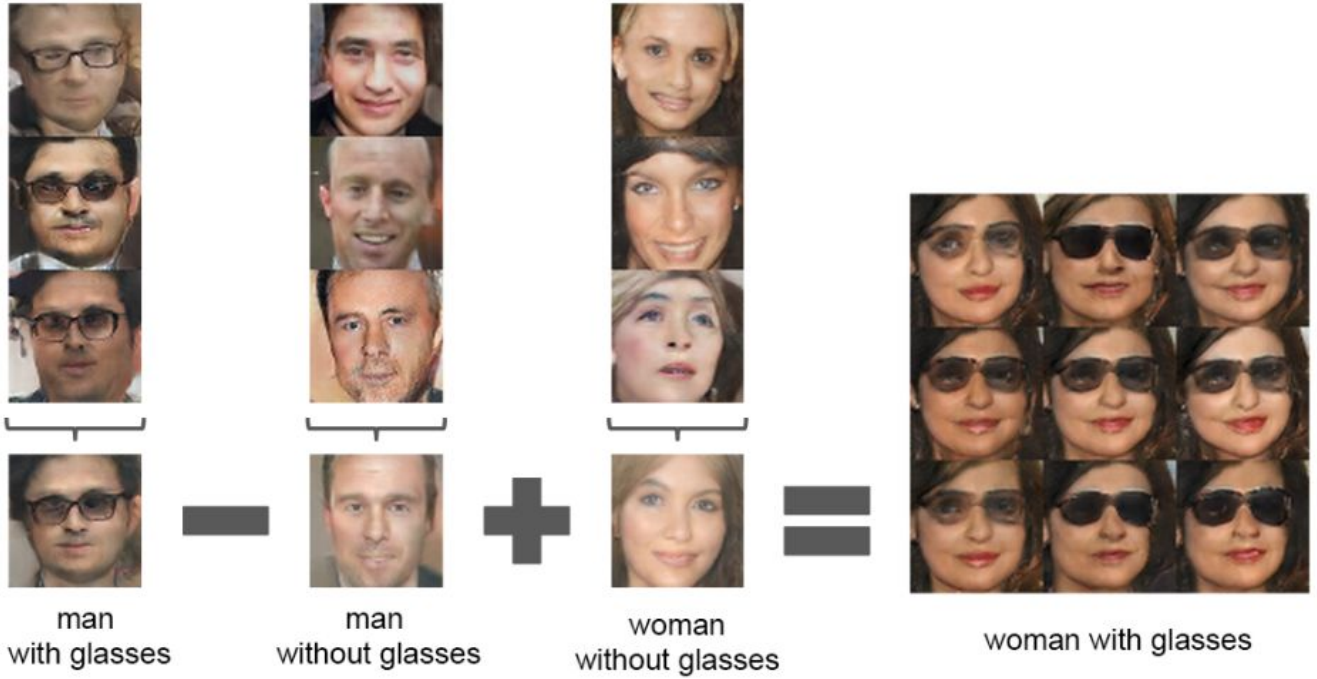
- Generator uses **transpose convolutions** to upsample latent vector z to the size of the generated image.
- Discriminator uses **strided convolutions** instead of pooling layers.
- Both uses batch normalisation.
- **LeakyReLu** activations for all the layers of the generator and discriminator, except their output.
- **Tanh** for output for the generator.

DCGAN Results



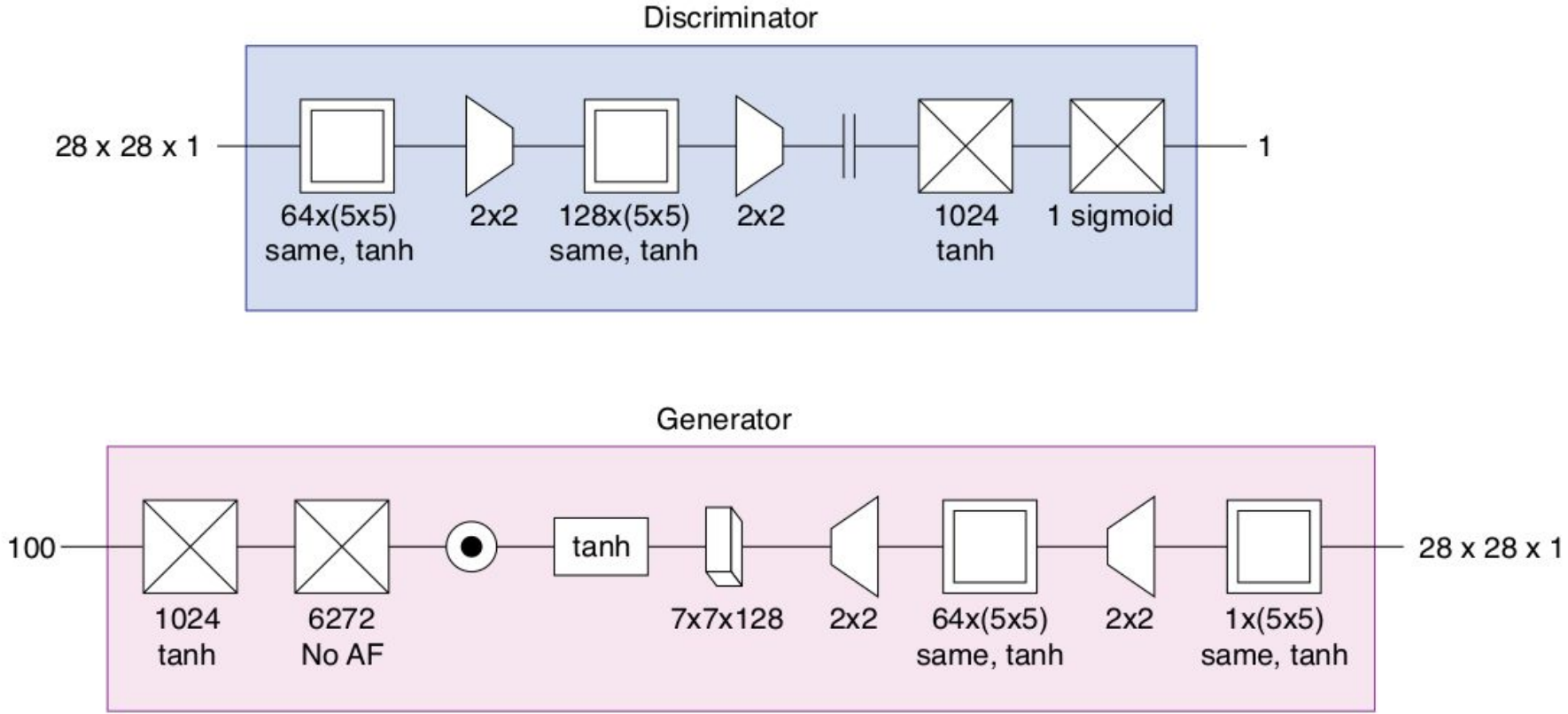
Authors generated some images and recorded the z vectors. Then created new z vectors (vector arithmetic) to controllable generation.

DCGAN Results



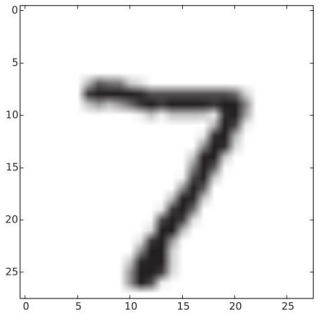
Authors generated some images and recorded the z vectors. Then created new z vectors (vector arithmetic) to controllable generation.

Deep Convolutional GAN (for MNIST)



Deep learning a visual approach Andrew Glassner Github

MNIST



	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	185	159	151	60	36	0	0	0	0	0	0	0	0	0
8	254	254	254	254	241	198	198	198	198	198	198	198	198	170
9	114	72	114	163	227	254	225	254	254	254	250	229	254	254
10	0	0	0	0	17	66	14	67	67	67	59	21	236	254
11	0	0	0	0	0	0	0	0	0	0	0	83	253	209
12	0	0	0	0	0	0	0	0	0	0	22	233	255	83
13	0	0	0	0	0	0	0	0	0	0	129	254	238	44
14	0	0	0	0	0	0	0	0	0	59	249	254	62	0
15	0	0	0	0	0	0	0	0	0	133	254	187	5	0
16	0	0	0	0	0	0	0	0	9	205	248	58	0	0
17	0	0	0	0	0	0	0	0	126	254	182	0	0	0
18	0	0	0	0	0	0	0	75	251	240	57	0	0	0
19	0	0	0	0	0	0	19	221	254	166	0	0	0	0
20	0	0	0	0	0	3	203	254	219	35	0	0	0	0
21	0	0	0	0	0	38	254	254	77	0	0	0	0	0
22	0	0	0	0	31	224	254	115	1	0	0	0	0	0
23	0	0	0	0	133	254	254	52	0	0	0	0	0	0
24	0	0	0	61	242	254	254	52	0	0	0	0	0	0
25	0	0	0	121	254	254	219	40	0	0	0	0	0	0
26	0	0	0	121	254	207	18	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 1.1: An Mnist discretized version of an image

Why Image Generation is challenging?