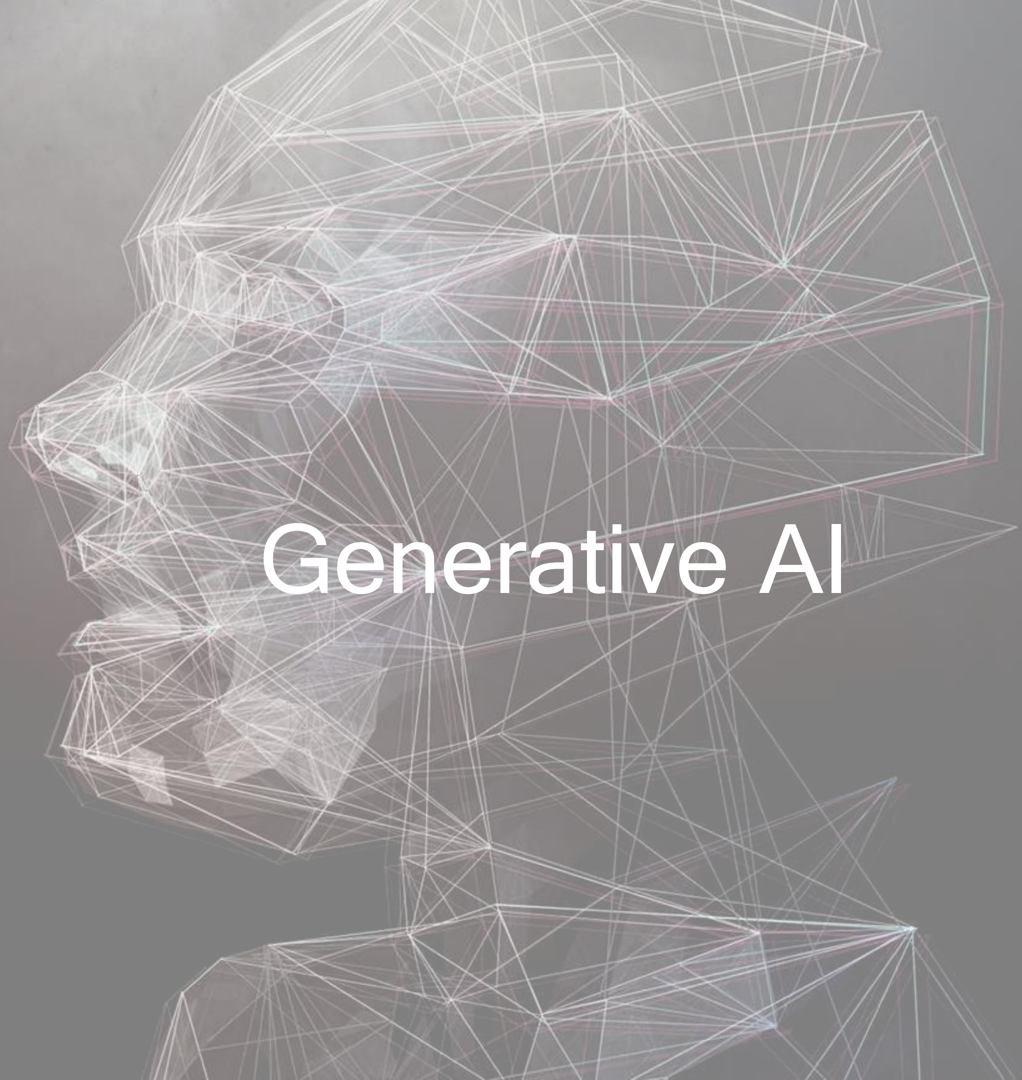
Generative AI

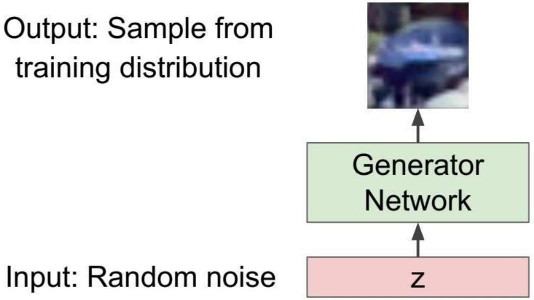


Generative Adversarial Networks (GANs)

https://github.com/eriklindernoren/PyTorch-GAN/blob/master/implementations/began/began.py

Generative Adversarial Networks

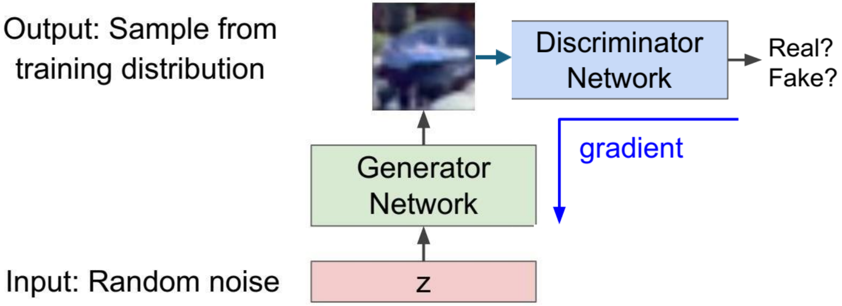
- Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!
- Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.
- Q: What can we use to represent this complex transformation?
- A: A neural network!



But we don't know which sample z maps to which training image -> can't learn by reconstructing training images

Generative Adversarial Networks

- Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!
- Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.
- Q: What can we use to represent this complex transformation?
- A: A neural network!

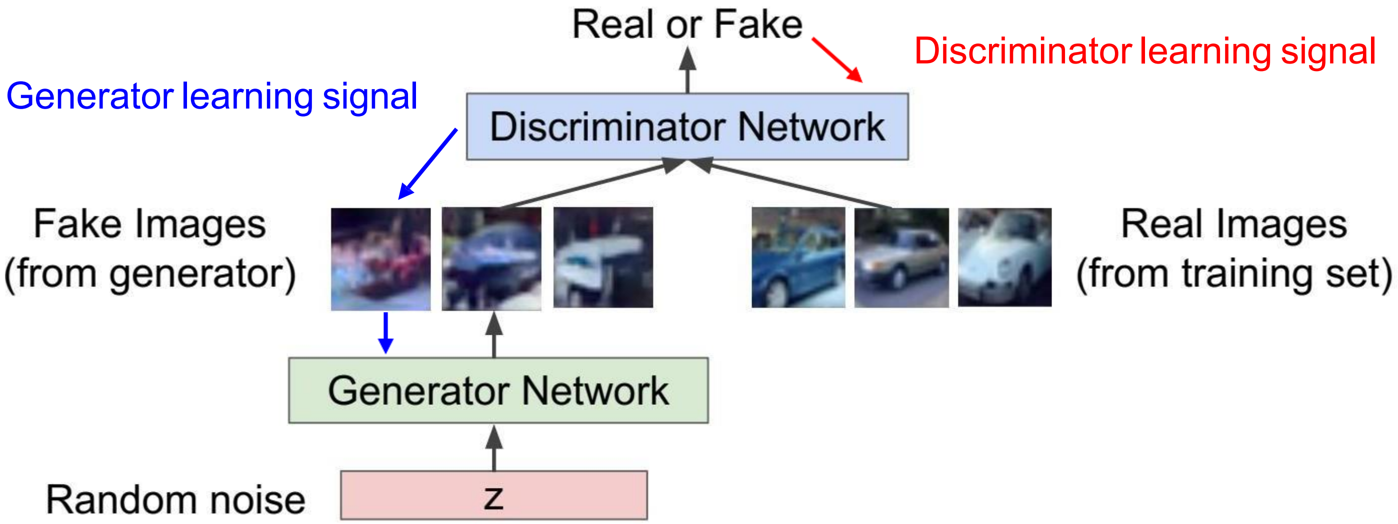


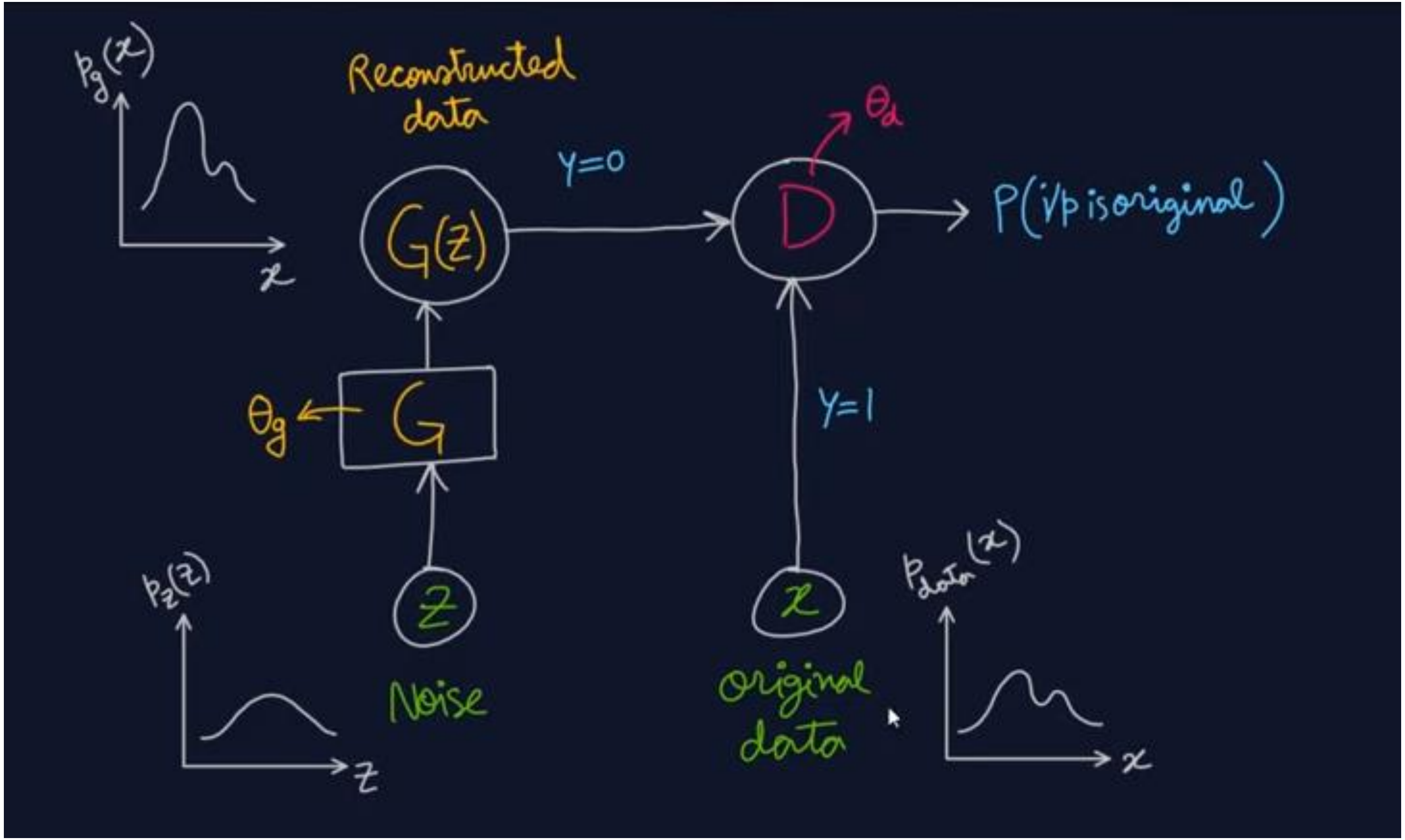
Solution: Use a discriminator network to tell whether the generate image is within data distribution or not

But we don't know which sample z maps to which training image -> can't learn by reconstructing training images

Training GANs: Two Player Game

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images





$p_g(x)$
x
Reconstructed data
$G(z)$
$Y=0$
θ_d
D
P(i/p is original)
θ_g
G
$Y=1$
$p_z(z)$
z
z
Noise
x
original data
$p_{data}(x)$
x

Adversarial Training Analogy: is it fake money?

Imagine we have a counterfeiter (**G**) trying to make fake money, and the police (**D**) has to detect whether money is real or fake.

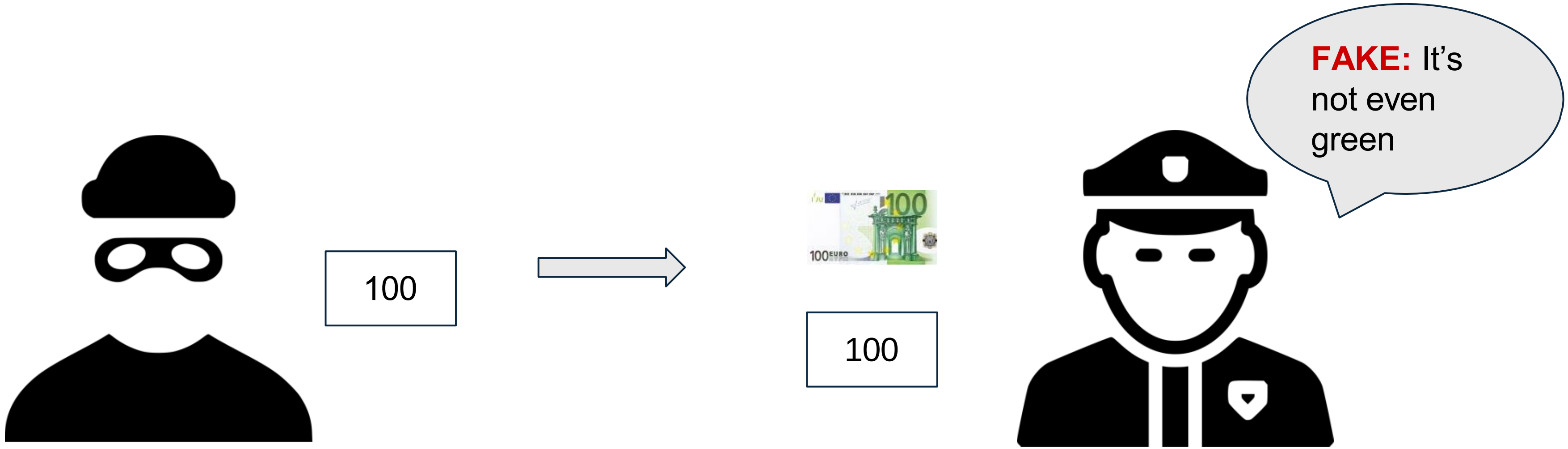


Figure: Santiago Pascual (UPC)

Adversarial Training Analogy: is it fake money?

Imagine we have a counterfeiter (**G**) trying to make fake money, and the police (**D**) has to detect whether money is real or fake.

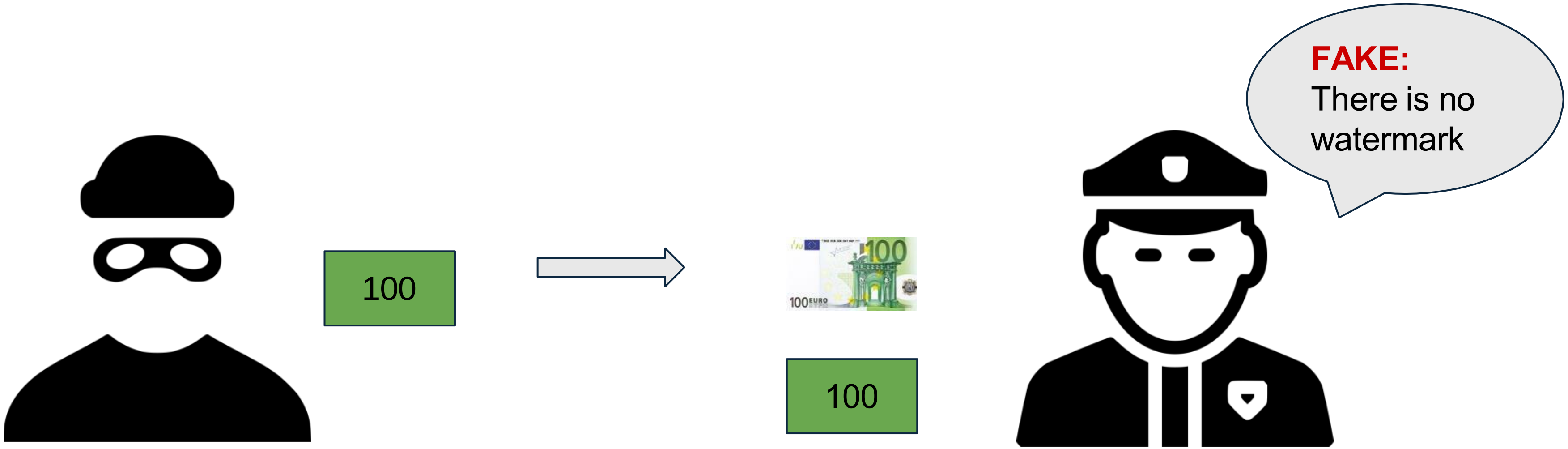


Figure: Santiago Pascual (UPC)

Adversarial Training Analogy: is it fake money?

Imagine we have a counterfeiter (**G**) trying to make fake money, and the police (**D**) has to detect whether money is real or fake.

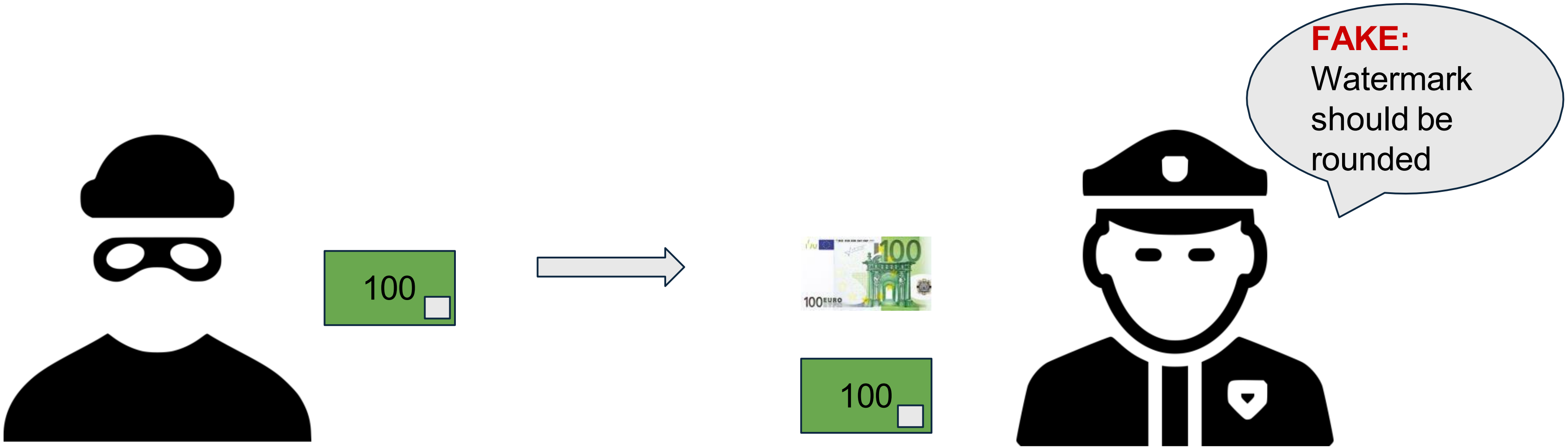


Figure: Santiago Pascual (UPC)

Adversarial Training Analogy: is it fake money?

Imagine we have a counterfeiter (**G**) trying to make fake money, and the police (**D**) has to detect whether money is real or fake.

After enough iterations, and if the counterfeiter is good enough (in terms of **G** network it means "has enough parameters"), the police should be confused.

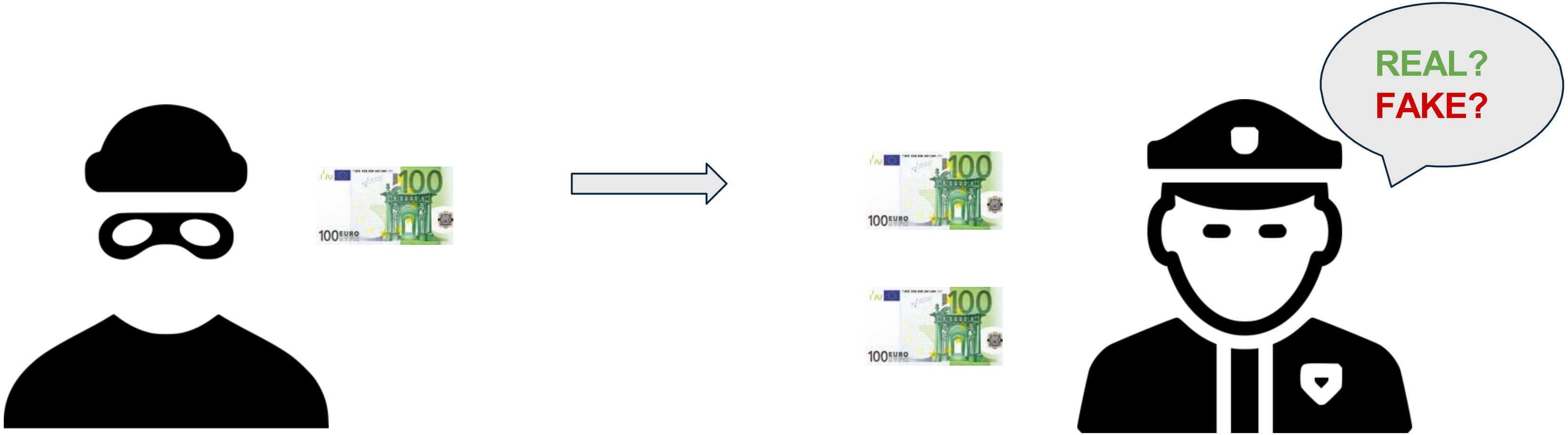


Figure: Santiago Pascual (UPC)

Adversarial Training

Alternate between training the discriminator and generator

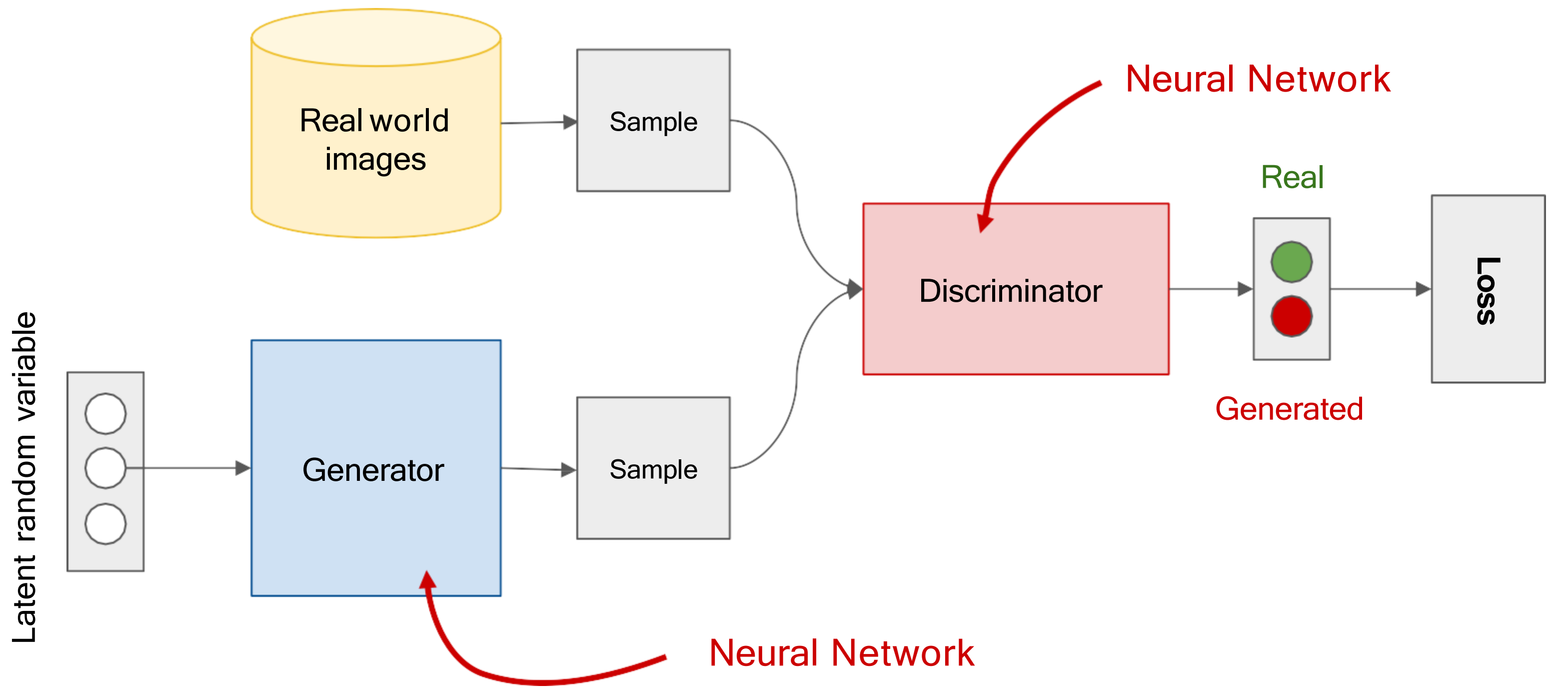


Figure: Kevin McGuinness (DCU)

Rough Work

BCE loss: $Loss(\hat{y}, y) = y \log \hat{y} + (1-y) \log(1-\hat{y})$

discriminator: real data (A)

$\hat{y} = D(x) \approx 1$

$y = 1$

⇓ substituting $y=1$ & $\hat{y} = D(x)$

$1 \cdot \log D(x) + (1-1)$

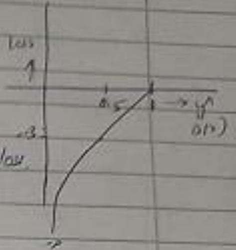
$\log D(x) \quad 0$

if $D(x) = 1$

$L(\hat{y}, y) = L(1,1) = \log(1) = 0$

$L(0.5,1) = \log(0.5) = -0.3$

$L(0,1) = \log(0) = -\infty$



You get maximum value 1 at 0, hence you need to maximize the loss

(B) discriminator: fake data

$\hat{y} = D(G(z))$, $y = 0$

substituting ↑ in BCE eq. $0 \cdot \log \ldots + (1-0) \log(1-D(G(z)))$

$\Rightarrow \log(1-D(G(z)))$

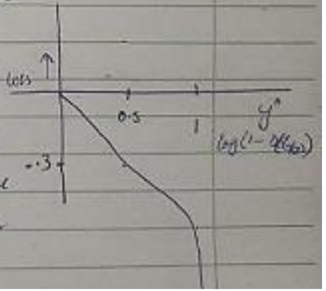
$L(\hat{y}, y) = L(0,0) = \log(1-0)$

$= \log(1) = 0$

taking $\hat{y} = 0, 0.5, 1$

$L(0.5,0) = \log(1-0.5) = -0.3$

$L(1,0) = \log(1-1) = -\infty$



For this case you want the D to classify this as 0, and you're getting this at 0, and increasing $\hat{y}$ values is giving you lower values, therefore u need to maximize this loss for discriminator.

Generator Loss: $1 - D(G(z))$: Case B is applied

$y = 0$, $\hat{y} = \log(1 - D(G(z))) \approx 1$ bc G wants to fool D.

Graph will remain the same, and observe the values, if u want to reach 1 for $\hat{y}$, you need to minimize this function. Hence we get a compact representation of our loss

$$\min_G \max_D \log D(x) + \log(1 - D(G(z)))$$

This loss equation is valid for only one data point, but it must be done for the entire training dataset

To represent this, we use the expectation term

Training GANs: Two Player Game

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images

Train jointly in minmax game

Discriminator outputs likelihood in (0,1) of real image

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

- Discriminator (θ_d) wants to **maximize objective** such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- Generator (θ_g) wants to **minimize objective** such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

Training GANs: Two Player Game

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

GANs

Traning loop :

* fix the learning of G *

Inner loop for D :

- take m data samples & m fake data samples
- update θ_d by grad. ascent

$$\frac{\partial}{\partial \theta_d} \frac{1}{m} \left[\ln[D(x)] + \ln[1 - D(G(z))] \right]$$

* fix the learning of D *

take m fake data samples

update θ_g by grad. descent

$$\frac{\partial}{\partial \theta_g} \frac{1}{m} \left[\ln[1 - D(G(z))] \right]$$

GANs

Binary Crossentropy Function

$$\mathcal{L} = -\sum y \ln \hat{y} + (1-y) \ln(1-\hat{y})$$

when, $y = 1$, $\hat{y} = D(x) \Rightarrow \mathcal{L} = \ln[D(x)]$

when $y = 0$, $\hat{y} = D(G(z)) \Rightarrow \mathcal{L} = \ln[1 - D(G(z))]$

Adding, $\mathcal{L} = \ln[D(x)] + \ln[1 - D(G(z))]$

This expression is valid for only one data point, but it must be done for the entire training dataset

To represent this, we use the expectation term

GANs

$$E(\mathcal{L}) = E\left(\ln[D(x)]\right) + E\left(\ln[1 - D(G(z))]\right)$$

$$\sum p_{data}(x) \ln[D(x)] + \sum p_z(z) \ln[1 - D(G(z))]$$

$$V(G,D) = E_{x \sim p_{data}}[\ln(D(x))] + E_{z \sim p_z}[\ln(1 - D(G(z)))]$$

Adversarial Training: Discriminator

1. Fix generator weights, draw samples from both real world and generated images
2. Train discriminator to distinguish between real world and generated images

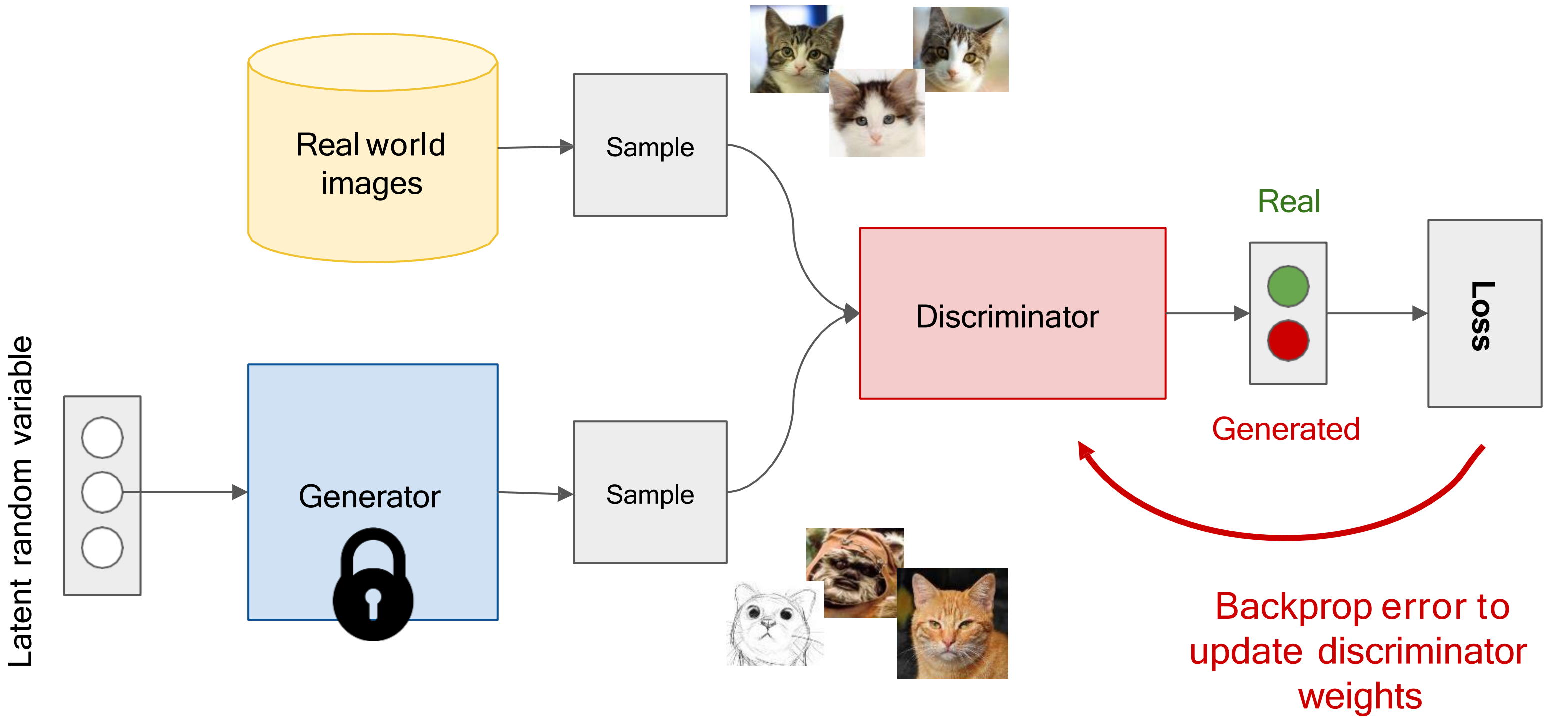


Figure: Kevin McGuinness (DCU)

Adversarial Training: Discriminator

Consider a binary encoding of “1” (Real) and “0” (Fake). Which of these two values should the discriminator predict when we <u>train the discriminator</u> with a generated image ?

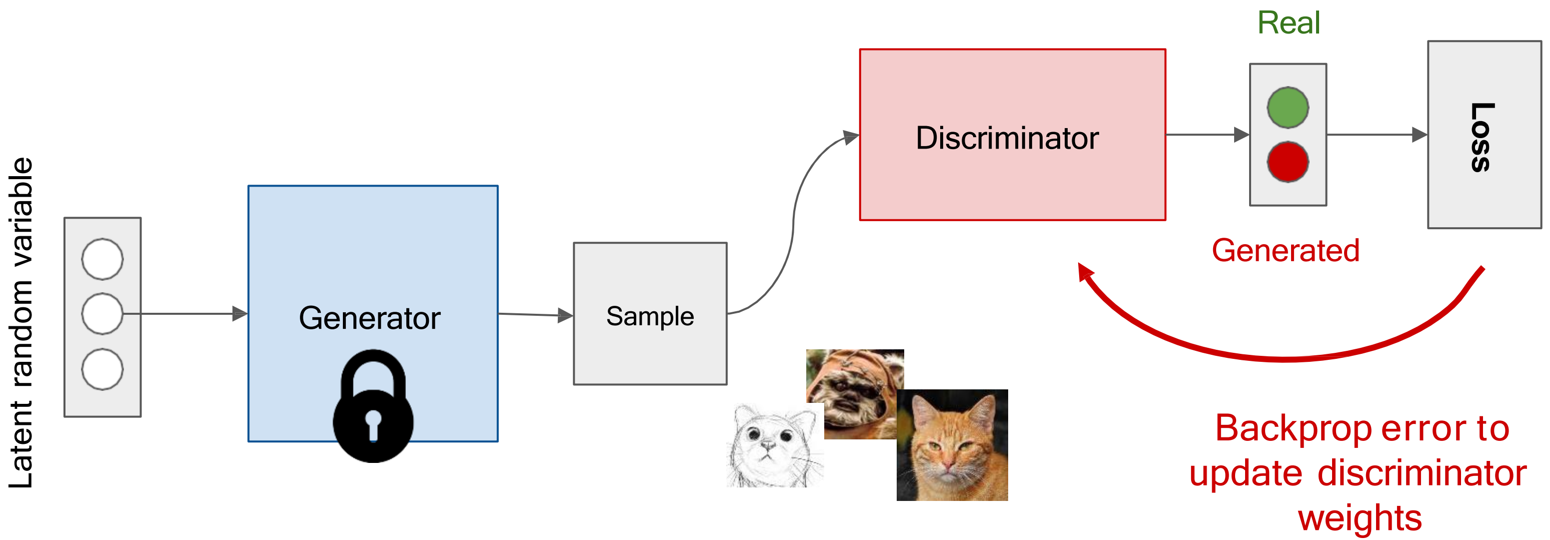


Figure: Kevin McGuinness (DCU)

Adversarial Training: Generator

1. Fix discriminator weights
2. Sample from generator by injecting noise.
3. Backprop error through discriminator to update generator weights (D(G(z)))

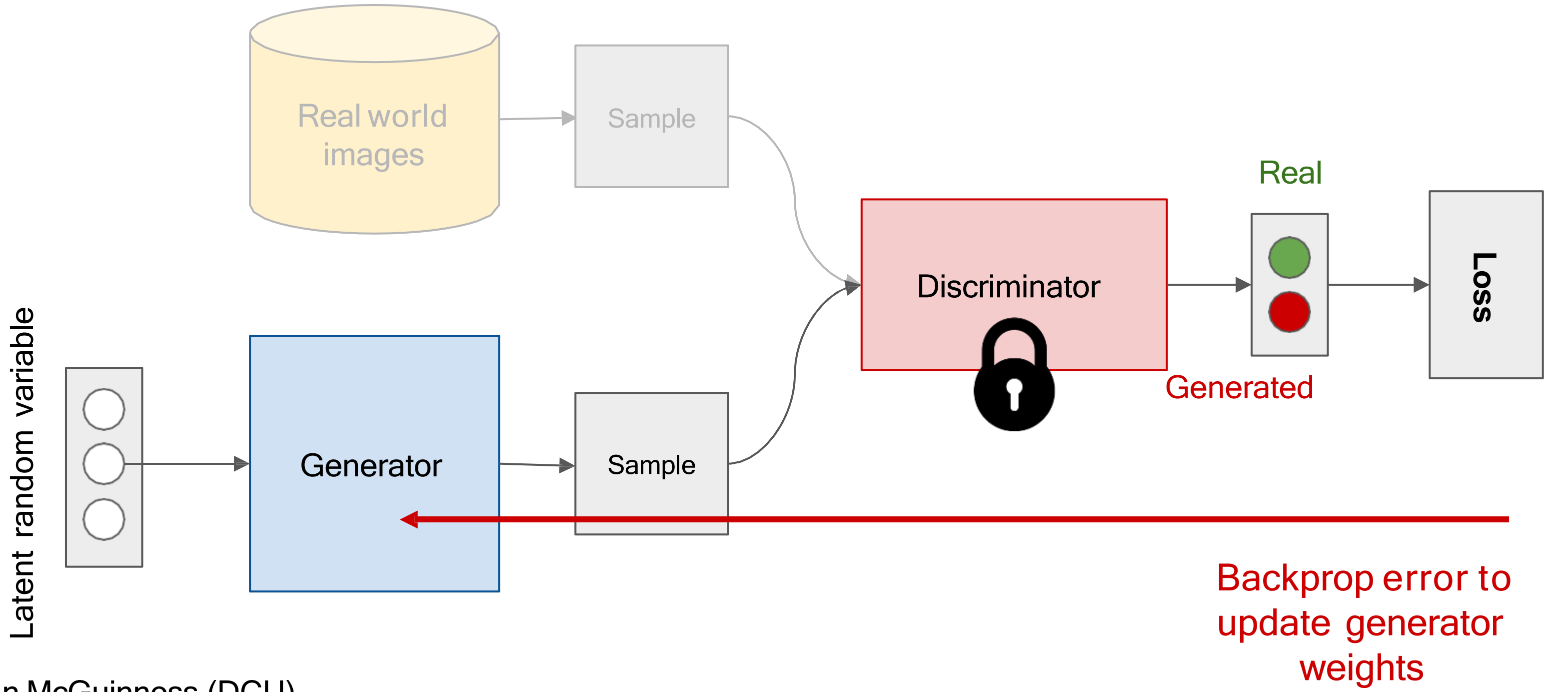


Figure: Kevin McGuinness (DCU)

Adversarial Training: Generator

Consider a binary encoding of "1" (Real) and "0" (Fake). Which of these two values should the discriminator predict when we train the generator with a generated image ?

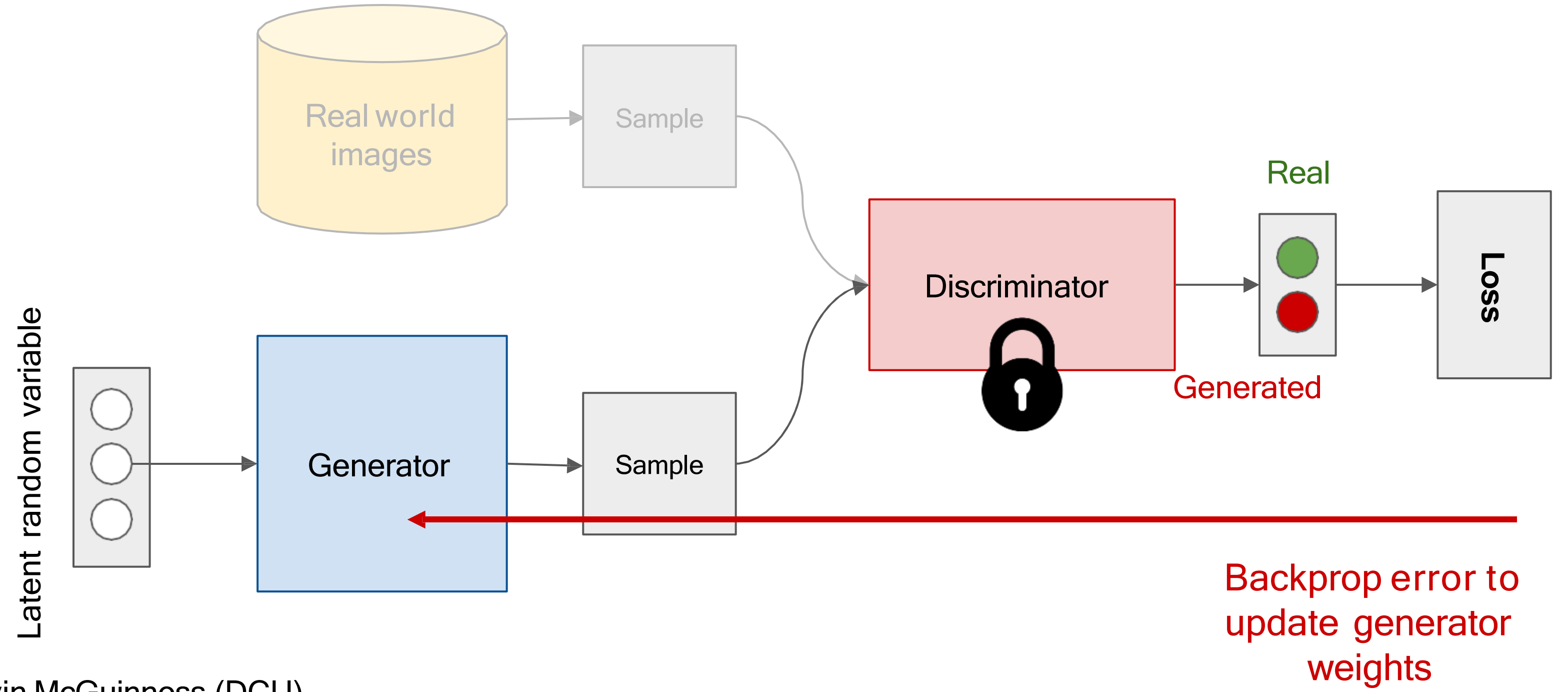


Figure: Kevin McGuinness (DCU)

Noise Dimensions

1. For Images (e.g., MNIST):

- It is common to use a lower-dimensional z (e.g., $z = 64$) to generate higher-dimensional outputs (e.g., 784 for MNIST or even 1024×1024 in high-res image GANs).
- This is **normal practice** because the generator learns how to expand the latent space into complex high-dimensional image representations.

2. For Tabular Data:

1. **Small Dataset or Simple Structure**: Use z equal to or slightly smaller than the output dimension.
2. **Complex Dataset or High Variability**: Slightly increase z (e.g., $z = 2\times$ output dimension) to give the generator more flexibility.
3. **Empirical Approach**: The best choice of z often depends on experimentation—try different sizes of z and evaluate performance metrics.

Generation Phase (After Training)

- Only the **generator** G is used after training. The discriminator is discarded.
- To generate new samples:
  1. **Sample noise** $z \sim p(z)$ (e.g., from a Gaussian $\mathcal{N}(0, 1)$ or uniform distribution).
  2. **Feed** z **into the trained generator**:

$$x_{\text{generated}} = G(z)$$

  3. **Output samples** are obtained based on the architecture (images, tabular data, etc.).

Controlling Sample Size:

- The number of samples generated is determined by how many z vectors are sampled.
- Example: If we want **100** generated samples, we sample **100 noise vectors** and pass them to G.