# Style Swap - GANs Notes

Gaurav Kumar Rampuria

May 28, 2024

## 1 Introduction

GANs, also known as Generative Adversarial Networks consist of 2 models - A generator (G) and a discriminator (D), which compete against each other.

The generator performs the role of generating realistic data from random noise, to try and fool the Discriminator and make it seem that it has come from the real dataset. For example, given a large corpus of photographs of faces, we might want to be able to generate a new photo realistic image that looks like it might plausibly have come from the same dataset. This kind of learning is called generative modeling.

Discriminator on the other hand, helps to differentiate between fake synthetic data and real data. Most ML tasks like Classification, Regression etc. are examples of Discriminative learning.

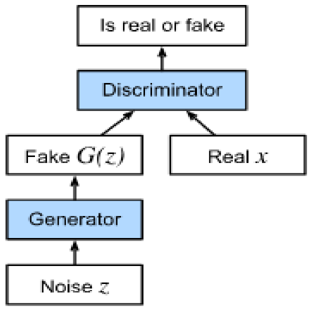

Figure 1: GANs Model

## 2 Adversarial nets

The adversarial modeling framework is most straightforward to apply when the models are both multilayer perceptrons. We use the following notation:-

- $P_z$ - Probability distribution of random noise

- $P_g$ - Probability distribution of Generator's output
- $P_{data}$ - Probability distribution of original data
- G(z, $\theta_g$) - Generator
- D(x; $\theta_d$) - Discriminator
- $\theta_g, \theta_d$ - Weights of the generator and discriminator model respectively

## 3 Value Function

$$V_{minG,maxD} = \mathbf{E_{x \approx p_d ata}} log[D(x)] + \mathbf{E_{z \approx p_z}} log[1 - D(G(z))]$$

## 4 Algorithm for Training

**Mini batch stochastic gradient descent training of generative adversarial nets.**
The number of steps to apply to the discriminator, k, is a hyper parameter. The paper uses k = 1, the least expensive option, in the experiments.

`for` number of training iterations do:

- `for` k steps do:
  - Sample mini batch of m noise samples z(1), . . . , z(m) from noise prior $p_g(z)$.
  - Sample mini batch of m examples samples x(1), . . . , x(m) from data generating distribution $p_d ata(x)$.
  - Update the discriminator using gradient ascent (as the discriminator is trying to maximise the value function).

$$\frac{\partial}{\partial \theta_d} \frac{1}{m} \sum_{i=1}^{m} [log(D(x^{(i)})) + log(1 - D(G(z^{(i)})))]$$

- end `for`
- Sample mini batch of m noise samples z(1), . . . , z(m) from noise prior $p_g(z)$.
- Update the generator using gradient descent (as the generator is trying to minimise the value function).

$$\frac{\partial}{\partial \theta_g} \frac{1}{m} \sum_{i=1}^{m} [log(1 - D(G(z^{(i)})))]$$

```
end for
```

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.