2019

# DCGAN and SAGAN

DEEP LEARNING - 1

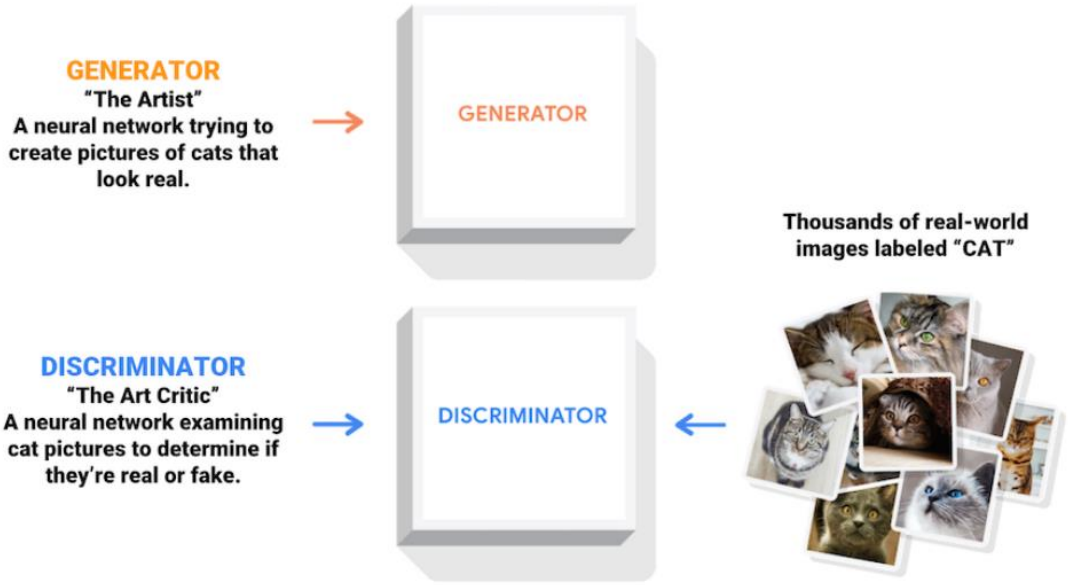
RAVI TEJA SUNKARA

UBID: 50292191

UBNAME: RSUNKARA

# What is GAN?

GAN stands for Generative Adversarial Network which is a class of neural networks that belong to unsupervised learning. Ian Goodfellow is known as the father of GANS when he made this discovery in 2014. It is all about creating from nothing(given some training data), be it images, music, video clips etc. This is the biggest step to achieve intelligence as the machine will start creating stuff of it's own, isn't that cool?..We would be able to see the unseen.. Nowdays GANS have started painting pictures on their own, rendering images, creating movies from scratch, creating game environments and many more crazy stuff.




## So how does this GAN work?

Generative adversarial nets consists of two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G.

The generator distribution $p_g$ over data data $x$, the generator builds a mapping function from a prior noise distribution $p_z(z)$ to data space as $G(z; \theta_g)$.

The discriminator, $D(x; \theta_d)$, outputs a single scalar representing the probability that $x$ came form training data rather than $p_g$.

The **cost function** $V(G, D)$:

$$\min_G \max_D V_{GAN}(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[log D(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))]$$

# What is Deep Convolution Generative Adversarial Network (DCGAN)?

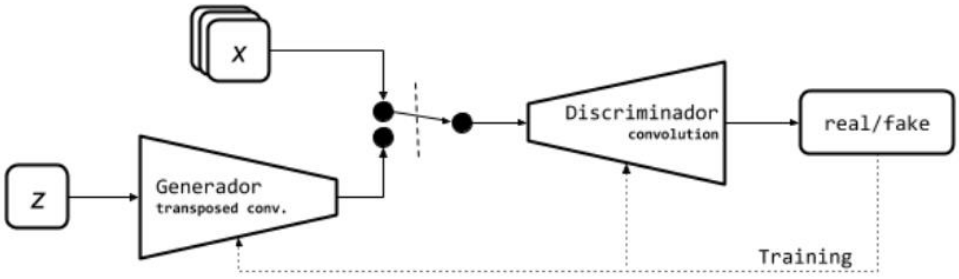
The difference between the simple GAN and the DCGAN, is the generator of the simple GAN is a simple fully connected network. The generator of the DCGAN uses the transposed convolution (Fractionally-strided convolution or Deconvolution) technique to perform up-sampling of 2D image size.

DCGAN are mainly composes of convolution layers without max pooling or fully connected layers. It uses convolutional stride and transposed convolution for the downsampling and the upsampling.

Here is the summary of DCGAN:

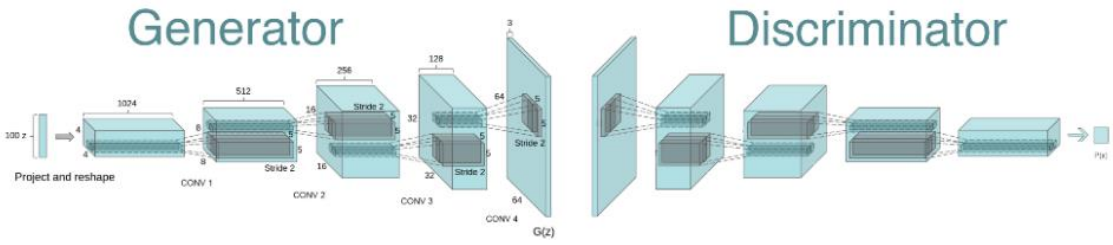
- Replace all max pooling with convolutional stride
- Use transposed convolution for upsampling.
- Eliminate fully connected layers.
- Use Batch normalization except the output layer for the generator and the input layer of the discriminator.
- Use ReLU in the generator except for the output which uses tanh.
- Use LeakyReLU in the discriminator.

**Network Design**




## How is it Implemented?

From the DCGAN paper, the authors specify that all model weights shall be randomly initialized from a Normal distribution with mean=0, stdev=0.02.



## 4. Training DCGANs with CIFAR-10 dataset, Keras and TensorFlow

A DCGANs implementation using the transposed convolution technique and the Keras library.

- **Data**
  - Rescale the CIFAR-10 images to be between -1 and 1.
- **Generator**
  - Use the **inverse of convolution**, called transposed convolution.
  - **ReLU activation** and **BatchNormalization**.
  - The input to the generator is the **normal distribution** $z$ or latent sample (100 values).
  - The last activation is **tanh**.
- **Discriminator**
  - Use the **Convolutional neural network**.
  - **LeakyReLU activation** and **BatchNormalization**.
  - The last activation is **sigmoid**.
- **Loss**
  - binary_crossentropy

Optimizer:

Adam(lr=0.0002, beta_1=0.5)

Epochs = 250

### What are Loss Functions and Optimizers for DCGAN?

We are using Binary Cross Entropy loss (BCEloss) function. This function provides the log components in the objective function (i.e. log(D(z)) and log(1-D(G(z)))).

real label = 1
fake label = 0.

We set up two separate optimizers, one for D and one for G. As specified in the DCGAN paper, both are Adam optimizers with learning rate 0.0002 and Beta1=0.5. For keeping track of the generator's learning progression, we will generate a fixed batch of latent vectors that are drawn from a Gaussian distribution (i.e. fixed_noise). In the training loop, we will periodically input this fixed_noise into G, and over the iterations we will see images form out of the noise.

## 3. How to Train?

We "constructed different mini-batches for real and fake" images, and also adjust G's objective function to maximize logD(G(z)). Training is split up into two main parts. Part 1 updates the Discriminator and Part 2 updates the Generator.

### 3.1 How to Train the Discriminator?

The goal of training the discriminator is to maximize the probability of correctly classifying a given input as real or fake. In terms of Goodfellow, we wish to "update the discriminator by ascending its stochastic gradient". Practically, we want to maximize log(D(x))+log(1−D(G(z))). Due to the separate mini-batch suggestion from ganhacks, we will calculate this in two steps. First, we will construct a batch of real samples from the training set, forward pass through D, calculate the loss (log(D(x))), then calculate the gradients in a backward pass. Secondly, we will construct a batch of fake samples with the current generator, forward pass this batch through D, calculate the loss (log(1−D(G(z)))), and accumulate the gradients with a backward pass. Now, with the gradients accumulated from both the all-real and all-fake batches, we call a step of the Discriminator's optimizer.

### 3.2 How to Train the Generator?

As stated in the original paper, we want to train the Generator by minimizing log(1−D(G(z))) in an effort to generate better fakes. This was shown by Goodfellow to not provide sufficient gradients, especially early in the learning process. As a fix, we instead wish to maximize log(D(G(z))). In the code we accomplish this by: classifying the Generator output from Part 1 with the Discriminator, computing G's loss using real labels as GT, computing G's gradients in a backward pass, and finally updating G's parameters with an optimizer step. It may seem counter-intuitive to use the real labels as real labels for the loss function, but this allows us to use the log(x) part of the BCELoss (rather than the log(1−x) part) which is exactly what we want.

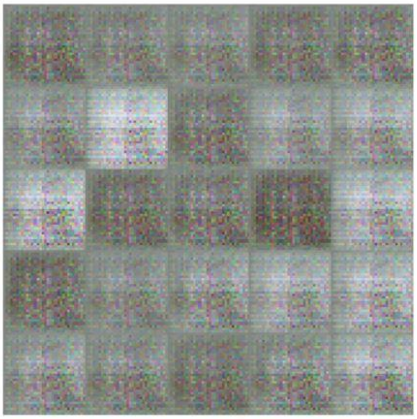
Finally, we did some statistic reporting and at the end of each epoch we will push our fixed_noise batch through the generator to visually track the progress of G's training.

## 4. How to Evaluate?

### 4.1 Show me the Loss Plots and Images!!

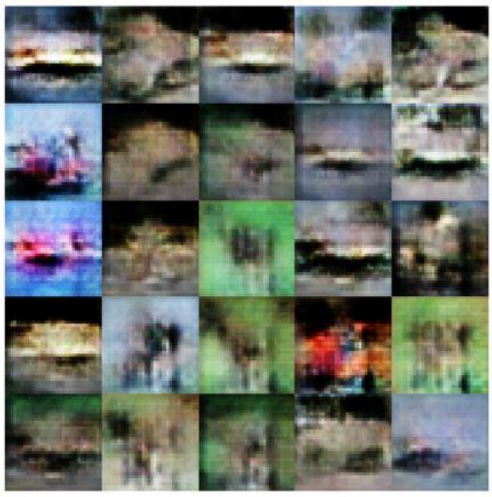
Initially when the generator was untrained, we were getting random noise images but later on for more number of epochs the DCGAN is learning very well and we are getting good images.

Epochs: 0/300 , D_Losses: 0.061 , G_Losses: 0.533

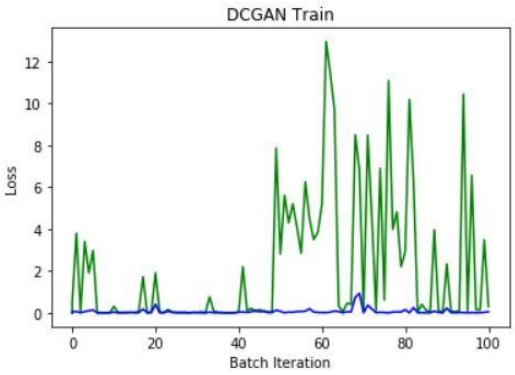


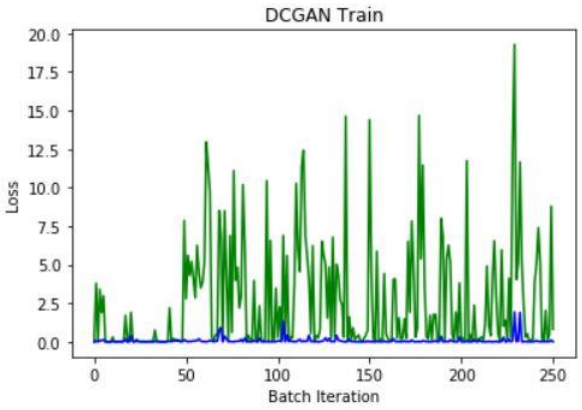
Epochs: 100/300 , D_Losses: 0.057 , G_Losses: 0.297

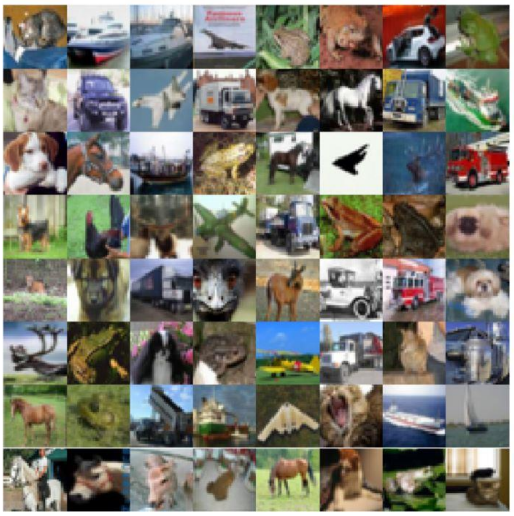
Epochs: 200/300 , D_Losses: 0.001 , G_Losses: 0.103

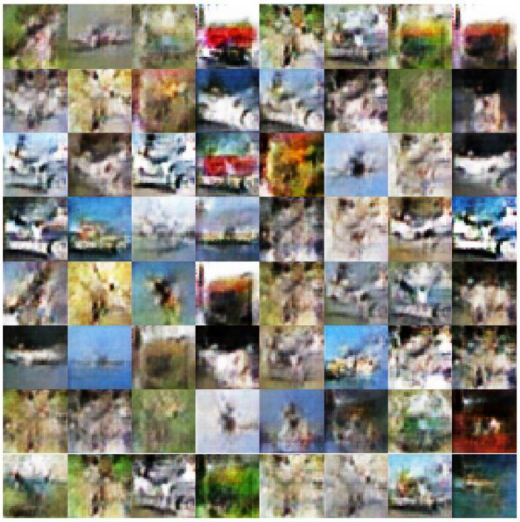


Epochs: 250/300 , D_Losses: 0.005 , G_Losses: 0.820





Test images Generated images

## 4.2 Show me the FID Values!!

The Frechet Inception Distance score, or FID for short, is a metric that calculates the distance between feature vectors calculated for real and generated images.

The FID score is then calculated using the following equation taken from the paper:

$$d^2((\mathbf{m}_r, \mathbf{C}_r), (\mathbf{m}_g, \mathbf{C}_g)) = \|\mathbf{m}_r - \mathbf{m}_g\|^2 + \mathrm{Tr}(\mathbf{C}_r + \mathbf{C}_g - 2(\mathbf{C}_r\mathbf{C}_g)^{\frac{1}{2}}),$$

| Epoch | FID |
|---|---|
| 100 | 257.465 |
| 150 | 209.549 |
| 200 | 169.427 |
| 250 | 123.777 |
| 300 | 96.375 |

# Conditional Spectrum Normalized Self Attention Generative Adversarial Network (CSNSAGAN)

## 1. Why do we need Self Attention in GAN?

For GAN models trained with ImageNet, they are good at classes with a lot of texture (landscape, sky) but perform much worse for structure. For example, GAN may render the fur of a dog nicely but fail badly for the dog's legs. While convolutional filters are good at exploring spatial locality information, the receptive fields may not be large enough to cover larger structures.

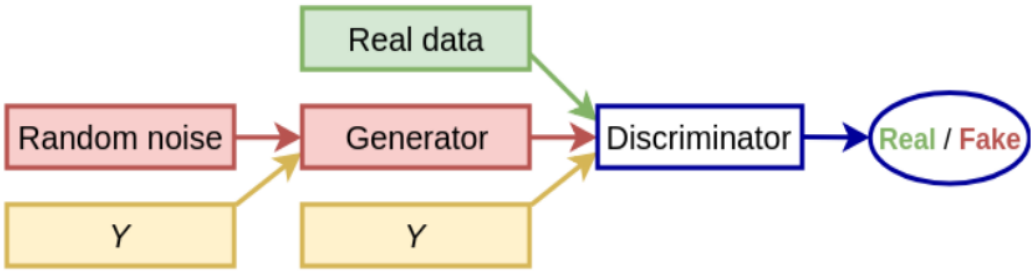
Given the limited representation capacity of the convolution operator (i.e., the receptive field is local), conventional GANs can only capture long-range relationships after several convolutional layers. One approach to alleviating this problem is to Increase the size of the kernel, but this is statistically and computationally inefficient. Various attention and self-attention models have been formulated to capture and use structural patterns and non-local relationships. However, these models are generally not effective in striking a balance between computational efficiency and modeling long-range relationships.

Alternatively, we can apply the attention concept. For example, to refine the image quality of the eye region (the red dot on the left figure), SAGAN only uses the feature map region on the highlight area in the middle figure. As shown below, this region has a larger receptive field and the context is more focus and more relevant. The right figure shows another example on the mouth area (the green dot).

## 2. What is Conditional GAN?

- Adding a vector of features controls the output and guide Generator figure out what to do.
- Such a vector of features should derive from a image which encode the class(like an image of a woman or a man if we are trying to create faces of imaginary actors) or a set of specific characteristics we expect from the image (in case of imaginary actors, it could be the type of hair, eyes or complexion).
- We can incorporate the information into the images that will be learned and also into the Z input, which is not completely random anymore.

- Discriminator's evaluation is done not only on the similarity between fake data and original data but also on the correspondence of the fake data image to its input label (or features)
- We can use the same DCGANs and imposed a condition on both Generator's and Discriminator's inputs. The condition should be in the form of a one-hot vector version of the digit. This is associated with the image to Generator or Discriminator as real or fake.




Conditional GAN

- The CGAN Discriminator's model is similar to DCGAN Discriminator's model except for the one-hot vector, which is used to condition Discriminator inputs.

## 3. What is Spectrum Normalization?

Though not designed to be a replacement for batch normalization, I thought I would give spectral normalization an honorable mention here since it gives us a very interesting look into normalization in deep learning in general. Spectral normalization was a method proposed to improve the training of GANs by limiting the Lipschitz constant of the discriminator. The Lipschitz constant is a constant L for a function f where for any x and y,

$$||f(x) - f(y)|| \leq L||x - y||$$

The authors restrict the Lipschitz constant by normalizing the weight matrices by their largest eigenvalue (or their spectral norm – hence the name). The largest eigenvalue is computed using the power method which makes the computational cost of this method very cheap. Compared to weight normalization, spectral normalization does not reduce the rank of the weight matrix.

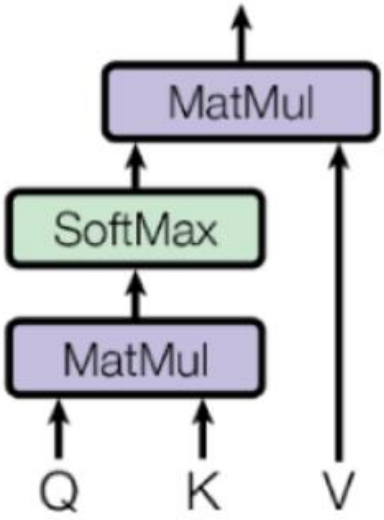
To solve the problem of slow learning and imbalanced update steps, there is a simple yet effective approach. It is important to note that in the GAN framework, G and D train together. In this context, Heusel et al introduced the **two-timescale update rule (TTUR)** in the GAN training. It consists of providing different learning rates for optimizing the generator and discriminator.

We are using the **two-timescale update rule (TTUR)** in the GAN training, which means providing different learning rates for optimizing the generator and discriminator. In our case we are giving G=0.001 and D=0.004 so discriminator trains 4 times quicker than generator. Due to this the larger part of the gradient goes to Discriminator and stabilizes the GAN.

## 4. What is Self-Attention?

### 4.1 How to Understand?

From diagram, we can understand that the Q(query) and K(key) undergo a matrix multiplication and then it goes through a softmax which can convert the resultant vector into the probability distribution which will be multiplied by V. Self-attention has all the value, key and query same.




Source: Attention is all you need paper

## 4.2 Explain me the Model!!

The feature maps moves from left to right as the image shown below. If the feature maps have the dimension of 64x16x16, 64 refers to the number of channels and 16 refers to the spatial dimension. The feature maps are passed through three 1x1 convolutions. The three filters are f, g and h.

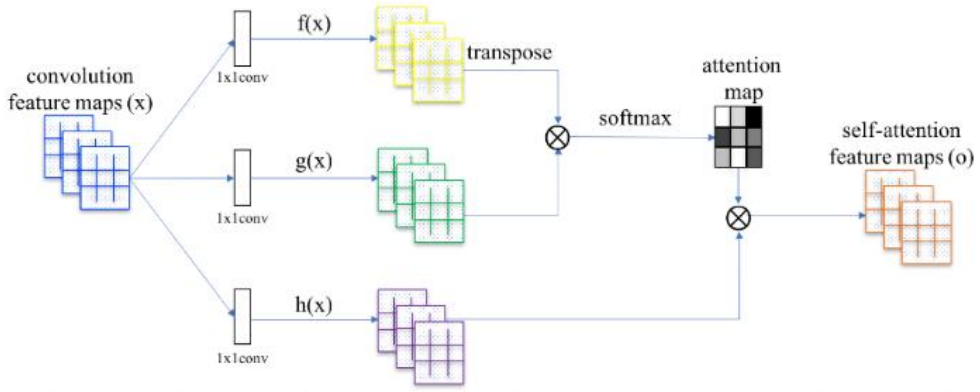



Figure 2: The proposed self-attention mechanism. The ⊗ denotes matrix multiplication. The softmax operation is performed on each row.

Source: SAGAN paper

The 1x1 convolution reduces the number of channels in the image. The filters f and g have 8 filters with 8x64x1x1 dimension whereas h has 64 filters. After the image convolves we get 3 feature maps of dimensions (8 x 16 x 16), (8 x 16 x 16) and (64 x 16 x 16).

f = (8x16x16) ; g = (8x16x16); h = (64x16x16)

- These three things are our query, key and value pairs. In order to perform self attention on the complete image we flatten out last two dimensions and the dimensions become (8 x 256), (8 x 256) and (64 x 256).
- Now we can perform the self-attention over it. We transpose the query and matrix-multiply it by the key and take the softmax on all the rows.
- So we get an output attention map of shape (256 x 256).
- Then we matrix multiply the value vector with the attention map the output is of the shape (64 x 256).
- One last thing that the paper proposes is that to multiply the final output by a learnable scale parameter and add back the input as a residual connection. Let's say the x was the image and o is the output, we multiply o by a parameter y. The final output O then becomes, O=y*o+x.
- Initially, the paper advises initializing the scale parameter as zero so that the output is simple old convolution at the beginning. They initialized y with zero because they wanted their network to rely on the cues in the local neighbourhood — since that was easier and then gradually the network will learn to assign the value to y parameter and use self-attention.

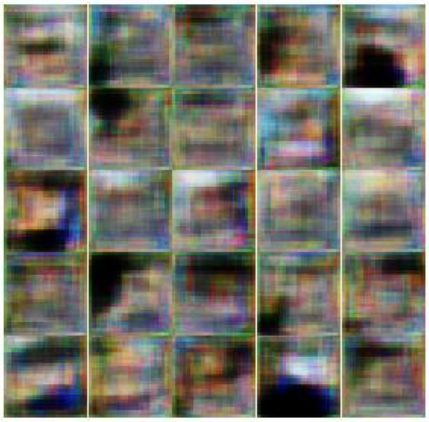
This layer helps the network capture the fine details from even distant parts of the image and remember, it does not replace convolution rather it is complementary to the convolution operation.

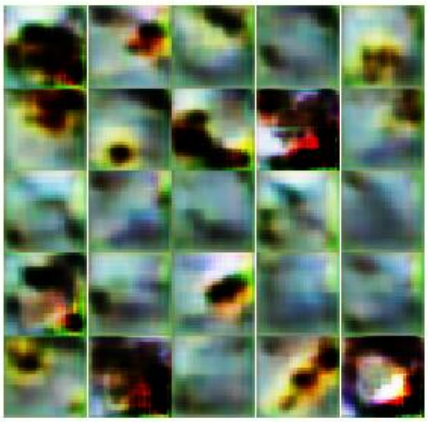
## 5. What are Loss Functions and Optimizers?

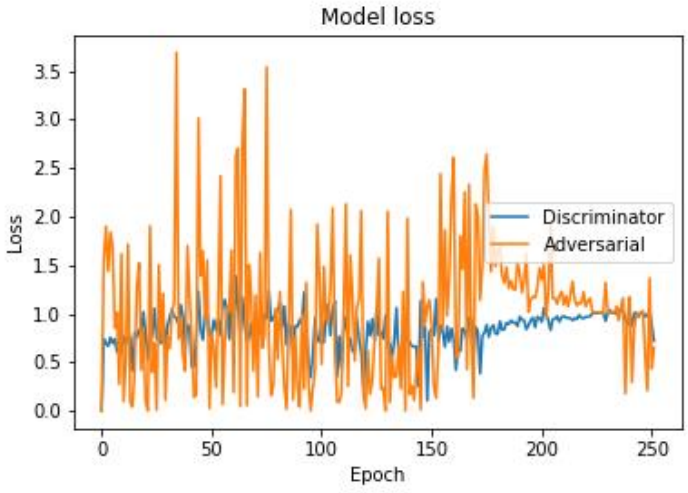
$$L_D = -\mathbb{E}_{(x,y)\sim p_{data}}[\min(0, -1 + D(x,y))] - \mathbb{E}_{z\sim p_z, y\sim p_{data}}[\min(0, -1 - D(G(z),y))],$$
$$L_G = -\mathbb{E}_{z\sim p_z, y\sim p_{data}} D(G(z),y),$$

The loss function used is the hinge version of the adversarial loss. The z is the latent vector from which the noise vector is generated and x and y in the equation refers to the real images from the dataset that is cifar10. The generator loss will try to create realistic image to fool the discriminator whereas the discriminator will try to become better at classifying the images created by the generator that is the fake images and the real images that are fed to it from the dataset.

## 6. What is happening during Training?








## 7. How are we Evaluating?

| Epoch | FID |
|---|---|
| 100 | 242.986 |
| 150 | 198.564 |
| 200 | 164.349 |
| 250 | 135.437 |