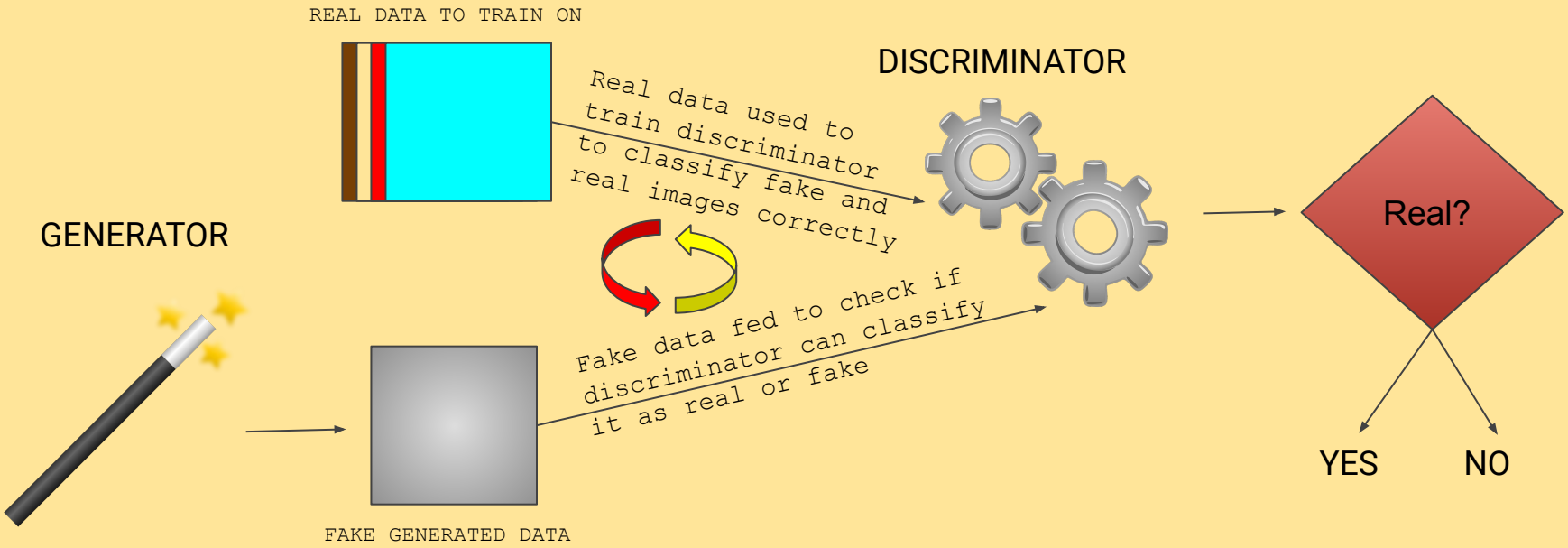
Generative Adversarial Nets

Basics Explained

By-
Arnab Sinha
IIIT Intern Report

What is GANs?

- Generative Adversarial Networks is a deep learning technique designed to generate data by training on real data.
- It is like a cat and mouse game, with each one trying to outsmart the other by learning better.



MATHEMATICAL MODEL

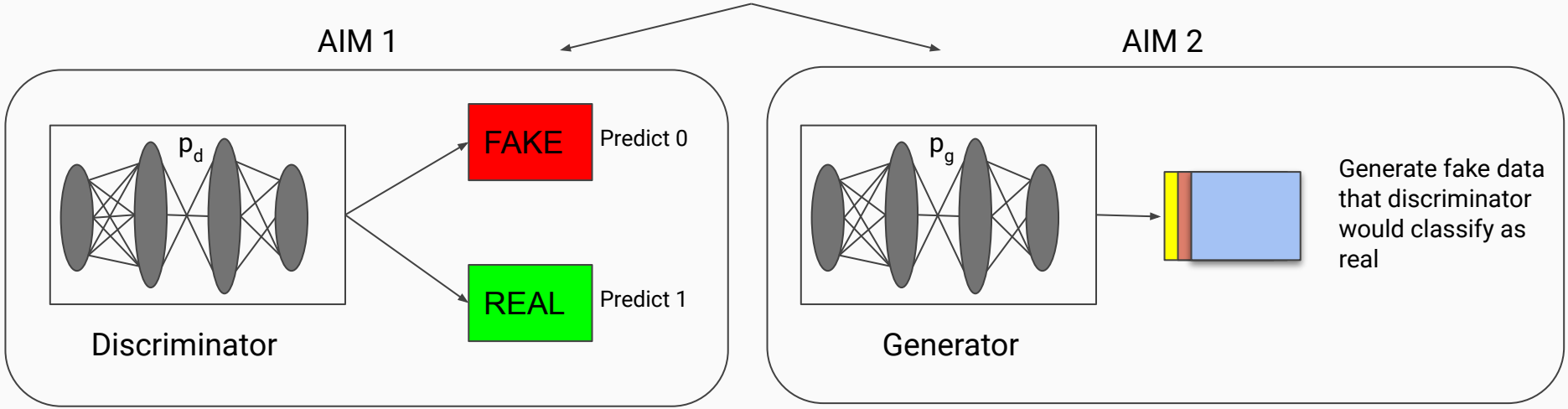
Terms to define

Let us first define some terms needed to understand GANs:

1. G = generator network
2. D = discriminator network
3. x = real data that we wish to train the adversarial net on
4. $x^{(i)}$ = i^{th} example from training set
5. $z^{(j)}$ = j^{th} example from noise data set
6. z = the noise data we input into the generator (latent/hidden variable)
7. $p_z(z)/p_g(z)/p_g$ = the generative distribution/ noise prior (prior distribution)
8. $p_{data}(x)/p_{data}/p(x)$ = the data generating distribution
9. D(x) = probability that x came from p_{data} rather than p_g
10. G(z) = the fake image generator creates
11. θ_g = the weights of the generative network
12. θ_d = the weights of the discriminative network
13. $E_{x\sim P(x)}(f(x))$ = S(f(x)P(X=x)) where S() means summation for all outcomes

MINIMAX FORMULA

- Aim : To train p_g over data x
- A prior on noise input is defined, normal distribution for instance
- The generator $G(z;\theta_g)$
  - is a multilayer perceptron that represents a differentiable function
  - Represents a mapping to the data space using z and θ_g
- The discriminator $D(x;\theta_d)$
  - is also a multilayer perceptron that outputs a single scalar value
  - Represents the probability that x comes from data rather than p_g
- The aim of the composite network is divided into 2 parts:



MINIMAX FORMULA

ERROR TERM

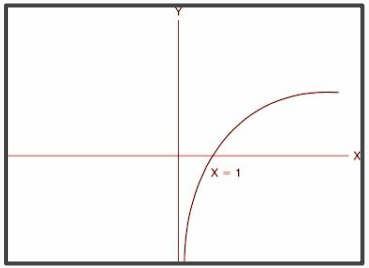
$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \quad (1)$$

1

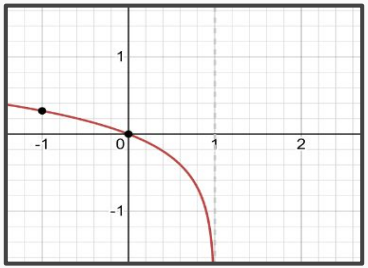
- Expected error in classifying the real data correctly.
- Discriminator tries to maximize this.

2

- Expected error in classifying the fake data.
- Discriminator tries to maximize this.
- Generator tries to minimize this.



Graph for log(x)



Graph for log(1-x)

NOTE:-

1. **D(x) is a probability scalar value ranging from [0,1]**
2. **If discriminator is 100% confident that image is fake, then D(x) = 0. If discriminator is 100% confident that image is real, else D(x) = 1.**

MINIMAX FORMULA

ERROR TERM

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

1 2

- The **first term deals with real data** and **second term with fake data**
- Note that if **D(x) = 1** and **D(G(z)) = 0** then **V(D,G) = 0**, which means the discriminator is totally dominating the generator.
- Also, if **D(x) = 0** and **D(G(z)) = 1** then **V(D,G) = -INF**, which means the generator totally dominates the discriminator.
- The **generator tries to minimize** the **second term** since it generates only the fake data
- The **discriminator tries to maximize both the terms** since it classifies both fake and real data

However, in practise equation (1) may not provide sufficient gradients to train G well since log(1-D(G(z))) will be close to 0 initially and log(1-D(G(z))) saturates.
So, instead of minimizing G, the authors prefer to use log(D(G(z))) to train the generator.

ALGORITHM

- **Mini batch SGD** is used to train the model two networks.
- As explained before, the whole error term is used to update the discriminator whereas only the second term is used to update the generator.
- **The discriminator and generator are trained alternatively.**
- The algorithm converges when

$$p_g = p_{data}$$

- When this happens,

$D_G^*(x) = p_{data}(x)/(p_{data}(x) + p_g(x))$

and

$V^*(D,G) = -\log(4)$ and $D_G^*(x) = 1/2$

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **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 by ascending its stochastic gradient:

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

  end for

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

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

end for

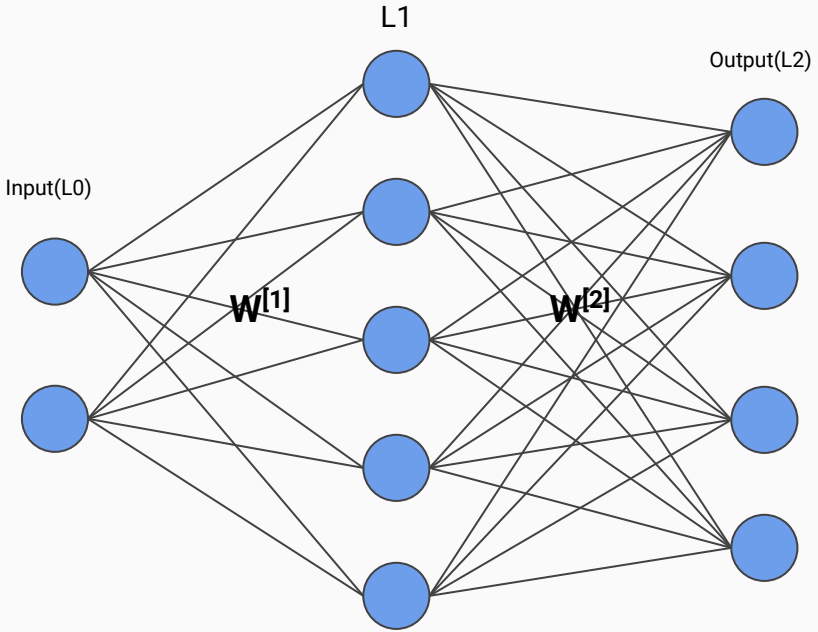
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

NUMERICAL EXAMPLE

THE GENERATOR MODEL

Let us assume an already **trained generator** model

- Having a 2*1 noise input
- Output layer with Tanh activation and 4*1 output / 2*2 image
- 1 hidden dense layer with Leaky Relu activation with alpha = 0.2
- Batch Norm excluded for simplified example



Assuming the weights of the layers are as follows:-

$W^{[1]}$

L1 \ L0	
0.1	0.2
0.2	0.25
0.3	0.1
0.6	0.5
0.2	0.9

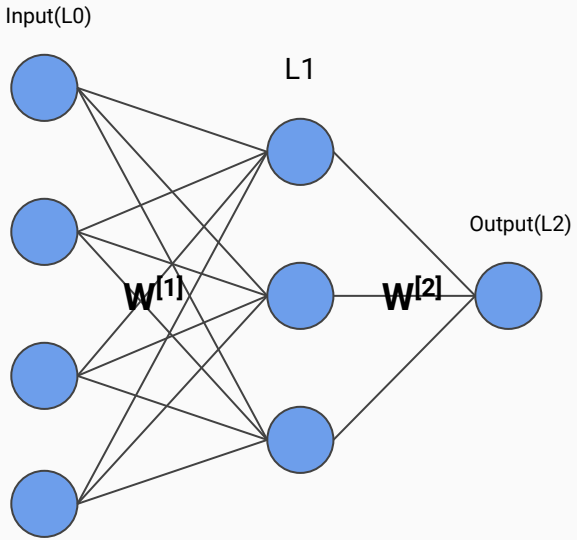
$W^{[2]}$

L2 \ L1				
0.1	0.2	0.15	0.45	0.1
0.2	0.2	0.2	0.3	0.2
0.1	0.5	0.6	0.1	0.2
0.8	0.2	0.9	0.75	0.2

THE DISCRIMINATOR MODEL

Let us assume an already **trained discriminator** model

- Having 4*1 input
- Having to generate a 1*1 scalar sigmoid output
- 1 hidden dense layer with Leaky Relu activation
- No dropout layer (for simplified numerical)



$W^{[1]}$

L1 \ L0				
	0.1	0.2	0.35	0.1
	0.5	0.2	0.2	0.1
	0.3	0.3	0.2	0.25

$W^{[2]}$

L2 \ L1			
	0.3	0.5	0.6

GENERATOR PART: SOLVING WITH A SINGLE INPUT

Let the input be a 2*1 vector with values from a gaussian distribution N(0,1)

This is the output of the second layer

0.1	0.2
0.2	0.25
0.3	0.1
0.6	0.5
0.2	0.9

5*2

x

0.25
-0.25

2*1

=

0.1*0.25 + 0.2*-0.25
0.2*0.25 + 0.25*-0.25
0.3*0.25 + 0.1*-0.25
0.6*0.25 + 0.5*-0.25
0.2*0.25 + 0.9*-0.25

5*1

=

-0.025
-0.0125
0.05
0.025
-0.175

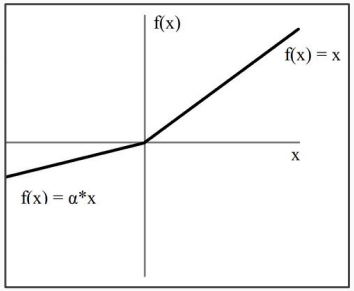
5*1

Leaky Relu (alpha = 0.2)

-0.005
-0.0025
0.05
0.025
0.035

5*1

We calculate the vector product of the two matrices to calculate the value of $Y^{[1]} = W^{[1]} * x^{[0]}$, which is the vector dot product of the two matrices



LeakyRelu

GENERATOR PART: SOLVING WITH A SINGLE INPUT

Now we have to calculate the output of the output layer.
Following operations similar to the previous slide:

0.1	0.2	0.15	0.45	0.1
0.2	0.2	0.2	0.3	0.2
0.1	0.5	0.6	0.1	0.2
0.8	0.2	0.9	0.75	0.2

4*5

x

-0.005
-0.0025
0.05
0.025
0.035

5*1

=

0.02125
0.0230
0.03775
0.06625

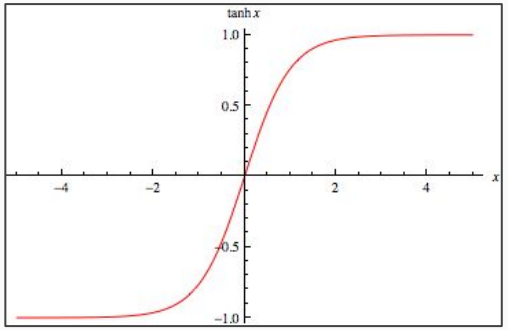
4*1

Tanh activation →

0.0213
0.0229
0.0377
0.0662

4*1

We again calculate the vector product of the two matrices to calculate the value of $Y^{[2]} = W^{[2]} * x^{[1]}$



This is the output of the generator which is fed into the discriminator

DISCRIMINATOR PART: SOLVING WITH A SINGLE INPUT

We now calculate the output of the hidden layer with the output of the generator and the $W^{[1]}$ matrix

0.1	0.2	0.35	0.1
0.5	0.2	0.2	0.1
0.3	0.3	0.2	0.25

3*4

X

0.0213
0.0229
0.0377
0.0662

4*1

=

0.026525
0.02939
0.03735

3*1

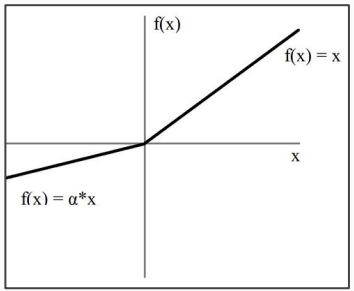
LeakyRelu Activation (alpha = 0.2)

0.0053
0.0058
0.0075

3*1

This is the output of the hidden layer in the discriminator.

We calculate the vector product of the two matrices to calculate the value of $Y^{[1]} = W^{[1]} * x^{[0]}$, which is the vector dot product of the two matrices

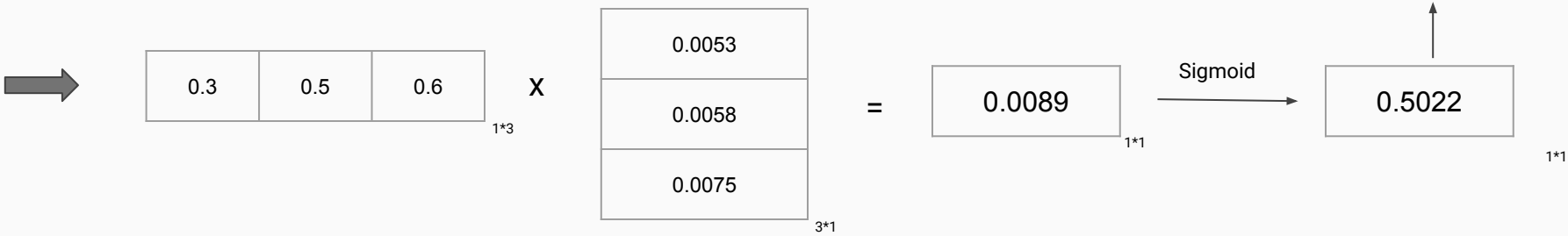


LeakyRelu

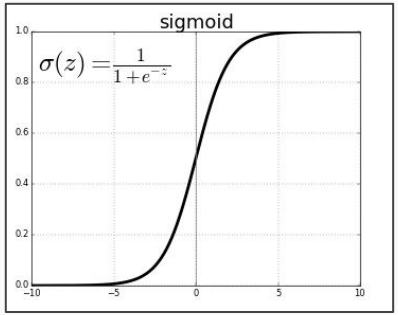
DISCRIMINATOR PART: SOLVING WITH A SINGLE INPUT

We now calculate the output of the final output layer which gives us the probability that the generator input image is real or not

This is the probability that the image is real. Since it is >0.5, discriminator classifies it as real image.



We again calculate the vector product of the second weight matrix and the output of the previous layer to calculate the value of $Y^{[2]} = W^{[2]} * x^{[1]}$



ASSUMING BATCH SIZE = 1

For loss term calculation

LOSS TERM

- Now, the ground truth for the generated image is 0, which is fake.
- The error term would therefore be calculated by binary cross entropy as mentioned before.

Loss = (1-y)log(1-y') + ylog(y') , where y = actual truth and y' = predicted probability

Here, y' = 0.5022 and y = 0

Hence, loss for fake image = log(1-0.5022) = log(0.4978) = -.30295 = L2 (say) **..(1)**

This loss corresponds to the second term in the error term.

LOSS TERM

- Now, suppose a real image is fed into the discriminator.
- The error term would therefore be calculated by binary cross entropy as mentioned before.

Loss = (1-y)log(1-y') + ylog(y') , where y = actual truth and y' = predicted probability

Here, suppose y' is found to be = 0.7032 and y = 1, since image is real in truth

Hence, loss for real image = log(0.7032) = -.15292 = L1 (say) **.. (2)**

This loss corresponds to the first term in the error term.

The discriminator loss = L1 + L2 = -0.45587
The generator loss = L2 = -0.30295

CONCLUSION

- WIth a bigger batch size, the slide 17 and 18 are repeated for batches of real images and fake images alternately.
- We therefore, now know how the images are evaluated and how the discriminator loss and generator loss is calculated.
- The losses are then passed into backward propagation to update the weights and improve both the generator and discriminator.
- **Backpropagation is not discussed here.**

WHY GANS?

Advantages:

- No inference needed during learning. Backprop is sufficient
- Represent very sharp and even degenerate distributions.
- No difficulty in sampling
- Any differentiable function is permitted to use in the model.

One careful point to be noted is that the discriminator must be synchronized well with the generator during training, otherwise too many values of z may be mapped to the same value of x.

THANK YOU