What is GANs

“(GANs), and the variations that are now being proposed is the most interesting idea in the last 10 years in ML, in my opinion.” by Yann LeCun

Source Code for: GANs DCGANs

What is GANs

Playing chess:

Compete with an opponent better than you
beat him / her in the next game
repeat this step
defeat the opponent

Source Code for: GANs DCGANs

University of Arkansas at Little Rock

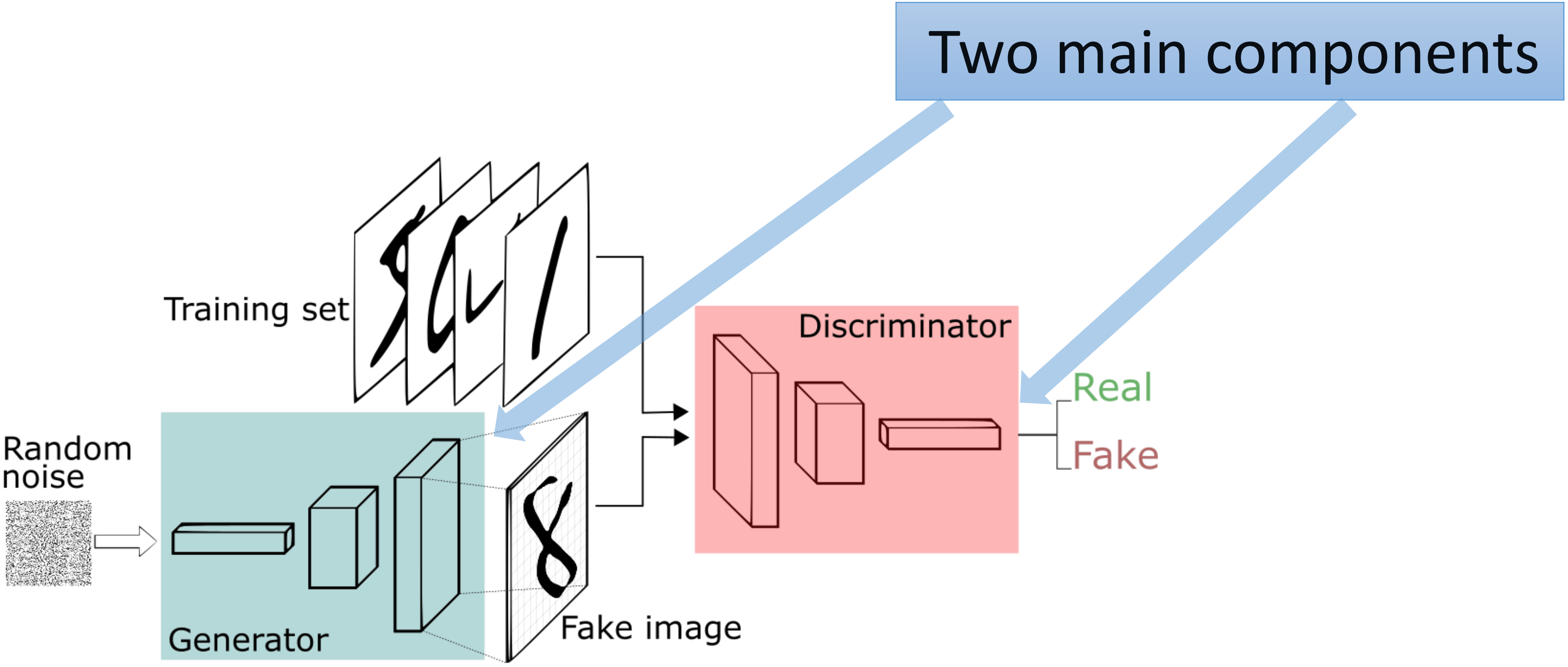
What is GANs

Forger and an investigator:

Forger: create fraudulent imitations
Investigator: catch these forgers who create the fraudulent
contest of forger vs investigator goes on
world class investigators (and unfortunately world class forger)

Source Code for: GANs DCGANs

What is GANs



Source Code for: [GANs](#) [DCGANs](#)

Define GAN

Pdata(x) -> The distribution of real data
X -> Sample from pdata(x)
P(z) -> Distribution of generator
Z -> Sample from p(z)
G(z) -> Generator Network
D(x) -> Discriminator Network

$$\min_G \max_D V(D, G)$$

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

Source Code for: GANs DCGANs

Define GANs

$$\min_G \max_D V(D,G)$$

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

Entropy of (pdata(x)) data from real distribution

Entropy of (p(z)) data from random input

Source Code for: GANs DCGANs

Define GANs

$$\min_G \max_D V(D,G)$$

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

Max

Min

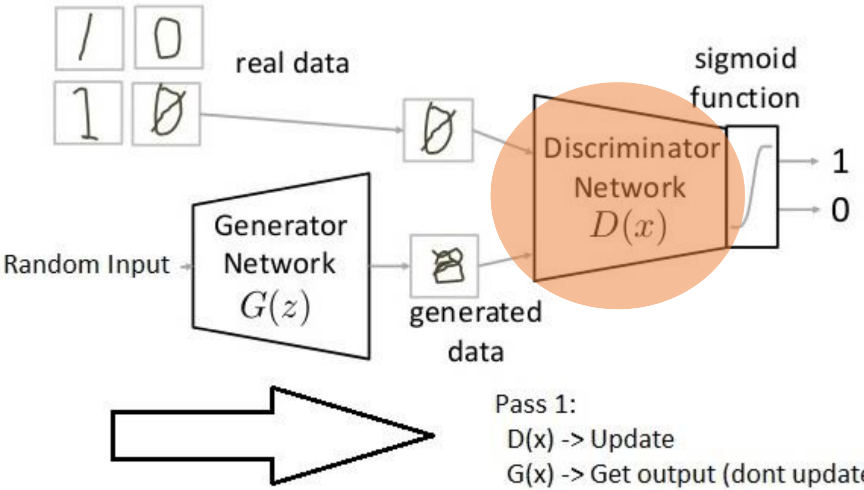
Discriminator is trying to maximize our function V

Generator is trying to minimize the function V

Source Code for: GANs DCGANs

Training GAN

Pass 1: Freeze generator, Training discriminator

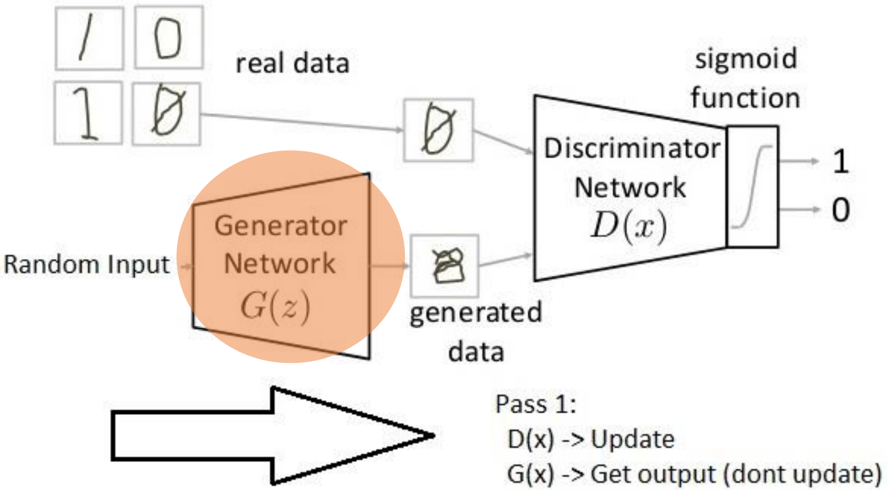


Pass 1:
D(x) -> Update
G(x) -> Get output (dont update)

Source Code for: GANs DCGANs

Training GAN

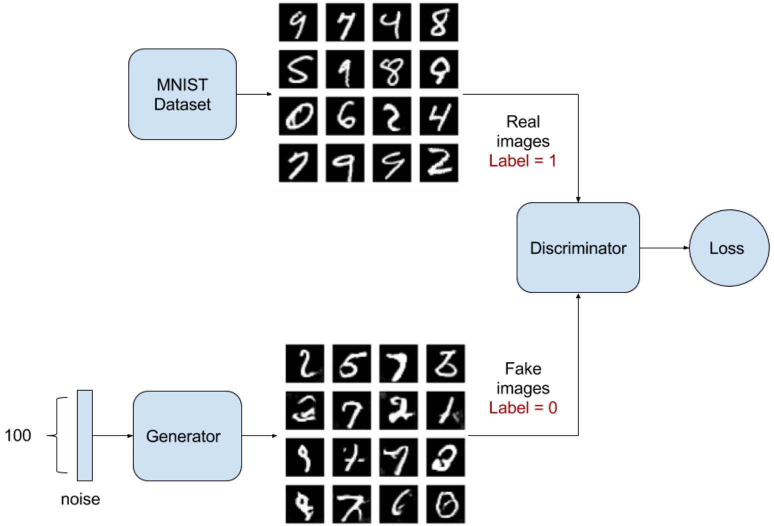
Pass 2: freeze discriminator, Train generator



Source Code for: GANs DCGANs

GANs project: Define problem

Generate fake images



MNIST Dataset Overview

60,000 examples for training

10,000 examples for testing.

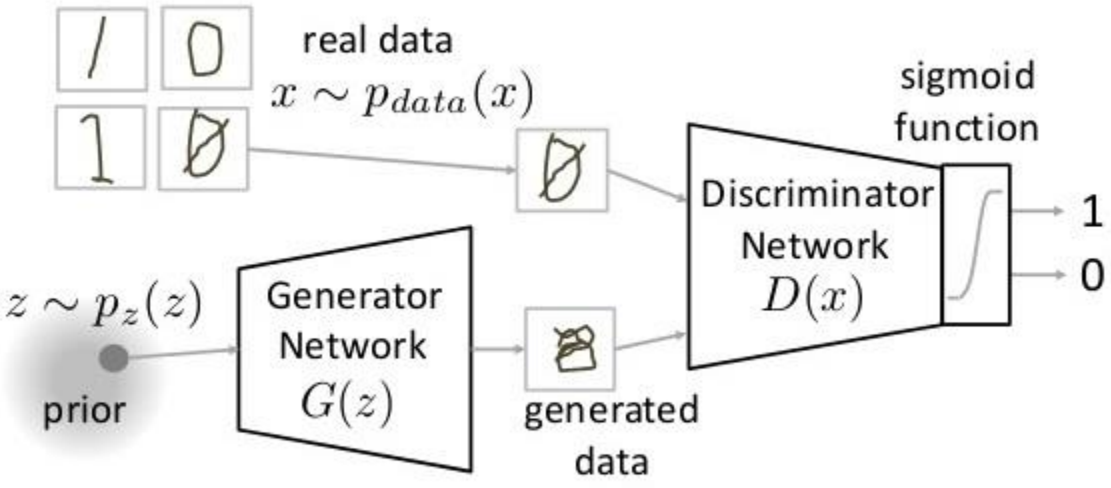
28x28 pixels with values from 0 to 1.

Flattened 1-D array of 784 features (28*28)

Source Code for: [GANs](#) [DCGANs](#)

GANs project: Define architecture

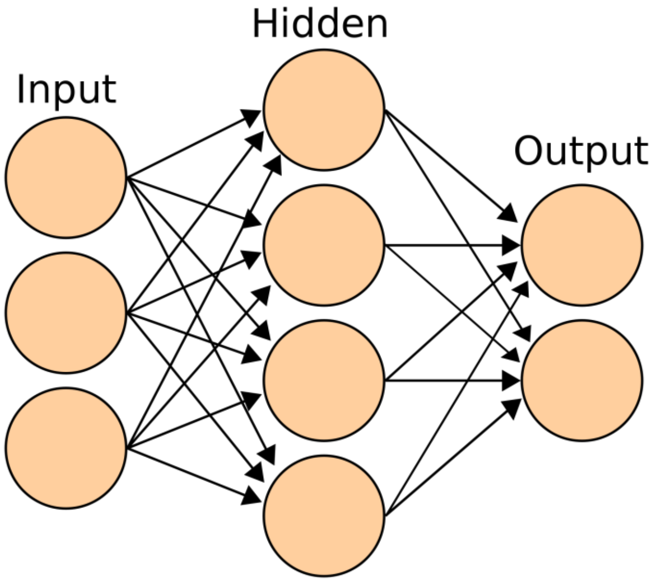
- One hidden layer for discriminator
- One hidden layer for generator



Source Code for: [GANs](#) [DCGANs](#)

GANs project: Define architecture

- One hidden layer for discriminator
- One hidden layer for generator




```
# Generator
def generator(x):
    hidden_layer = tf.matmul(x, weights['gen_hidden1'])
    hidden_layer = tf.add(hidden_layer, biases['gen_hidden1'])
    hidden_layer = tf.nn.relu(hidden_layer)
    out_layer = tf.matmul(hidden_layer, weights['gen_out'])
    out_layer = tf.add(out_layer, biases['gen_out'])
    out_layer = tf.nn.sigmoid(out_layer)
    return out_layer


# Discriminator
def discriminator(x):
    hidden_layer = tf.matmul(x, weights['disc_hidden1'])
    hidden_layer = tf.add(hidden_layer, biases['disc_hidden1'])
    hidden_layer = tf.nn.relu(hidden_layer)
    out_layer = tf.matmul(hidden_layer, weights['disc_out'])
    out_layer = tf.add(out_layer, biases['disc_out'])
    out_layer = tf.nn.sigmoid(out_layer)
    return out_layer
```

Source Code for: [GANs](#) [DCGANs](#)

GANs project: Training

1. Train Discriminator on real data for n epochs
2. Generate fake inputs for generator
3. Train discriminator on fake data
4. Train generator with the output of discriminator
5. Repeat 1-4 steps
6. Check fake data result

Source Code for: GANs DCGANs

GANs project: Results

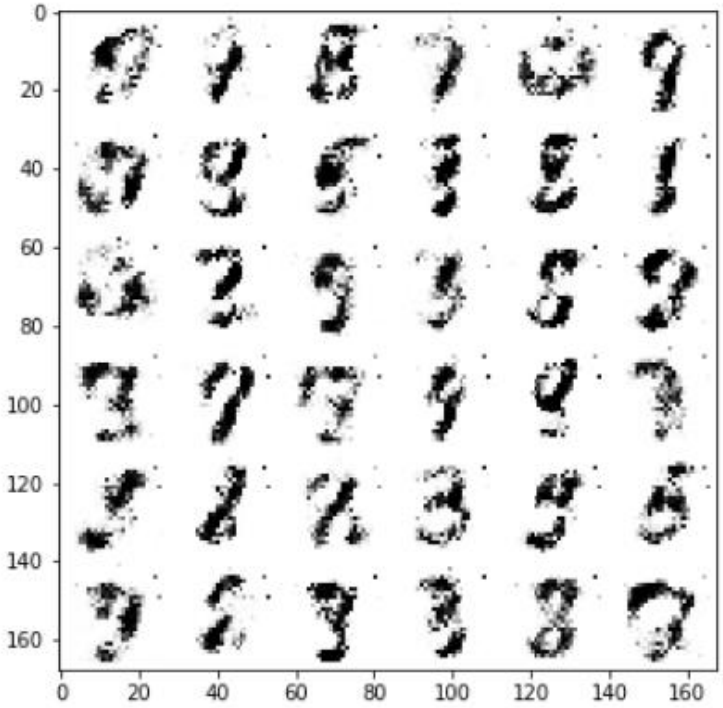
Training 10,000 steps

```
Step 1: Generator Loss: 1.057984, Discriminator Loss: 1.227529
Step 2000: Generator Loss: 4.775870, Discriminator Loss: 0.042116
Step 4000: Generator Loss: 3.751669, Discriminator Loss: 0.136570
Step 6000: Generator Loss: 3.318022, Discriminator Loss: 0.189792
Step 8000: Generator Loss: 4.373503, Discriminator Loss: 0.162921
Step 10000: Generator Loss: 3.622272, Discriminator Loss: 0.264043
```

Source Code for: GANs DCGANs

GANs project: Results

Training 10,000 steps



Source Code for: [GANs](#) [DCGANs](#)

GANs project: Results

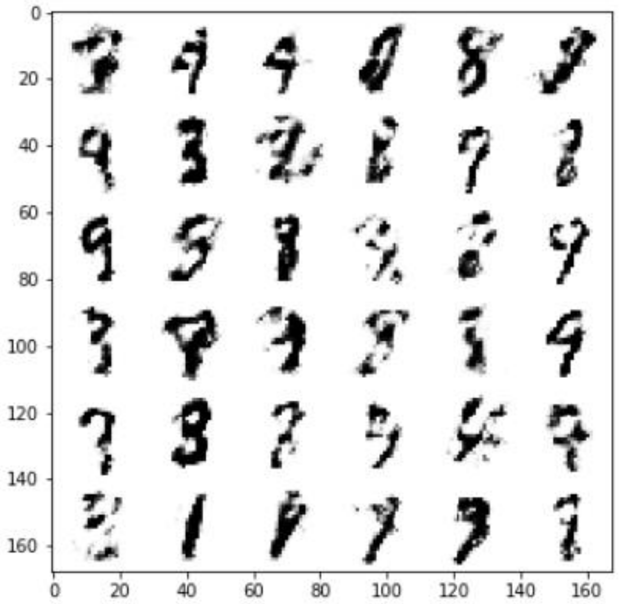
Training 40,000 steps

```
Step 1: Generator Loss: 1.022040, Discriminator Loss: 1.238947
Step 2000: Generator Loss: 4.983431, Discriminator Loss: 0.019256
Step 4000: Generator Loss: 4.562924, Discriminator Loss: 0.040434
Step 6000: Generator Loss: 4.215461, Discriminator Loss: 0.150144
Step 8000: Generator Loss: 4.020543, Discriminator Loss: 0.155626
Step 10000: Generator Loss: 3.525209, Discriminator Loss: 0.205117
Step 12000: Generator Loss: 3.320497, Discriminator Loss: 0.336670
Step 14000: Generator Loss: 2.778084, Discriminator Loss: 0.518560
Step 16000: Generator Loss: 3.285352, Discriminator Loss: 0.277530
Step 18000: Generator Loss: 3.258935, Discriminator Loss: 0.351666
Step 20000: Generator Loss: 3.346839, Discriminator Loss: 0.306597
Step 22000: Generator Loss: 4.782597, Discriminator Loss: 0.111715
Step 24000: Generator Loss: 3.731757, Discriminator Loss: 0.283805
Step 26000: Generator Loss: 3.880025, Discriminator Loss: 0.294006
Step 28000: Generator Loss: 3.450228, Discriminator Loss: 0.309087
Step 30000: Generator Loss: 3.259465, Discriminator Loss: 0.457083
Step 32000: Generator Loss: 3.081173, Discriminator Loss: 0.393552
Step 34000: Generator Loss: 2.973398, Discriminator Loss: 0.378245
Step 36000: Generator Loss: 3.155655, Discriminator Loss: 0.401714
Step 38000: Generator Loss: 3.191599, Discriminator Loss: 0.432039
Step 40000: Generator Loss: 3.334904, Discriminator Loss: 0.439917
```

Source Code for: GANs DCGANs

GANs project: Results

Training 40,000 steps



Source Code for: GANs DCGANs

GANs project: Results

Training 80,000 steps

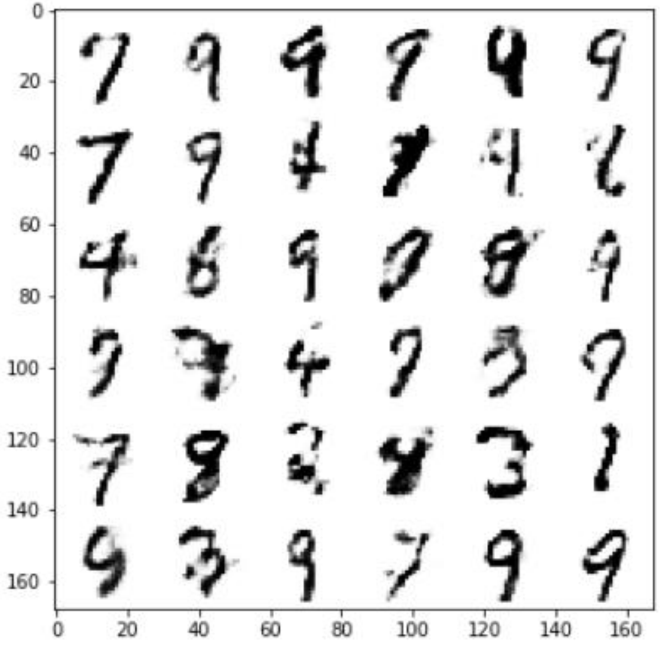
```
Step 1: Generator Loss: 0.482775, Discriminator Loss: 1.683681
Step 2000: Generator Loss: 4.478746, Discriminator Loss: 0.041249
Step 4000: Generator Loss: 4.232111, Discriminator Loss: 0.064319
Step 6000: Generator Loss: 4.142162, Discriminator Loss: 0.117743
Step 8000: Generator Loss: 4.063119, Discriminator Loss: 0.105196
Step 10000: Generator Loss: 4.291493, Discriminator Loss: 0.145389
Step 12000: Generator Loss: 4.307825, Discriminator Loss: 0.237374
Step 14000: Generator Loss: 3.159351, Discriminator Loss: 0.436120
Step 16000: Generator Loss: 4.056623, Discriminator Loss: 0.170912
Step 18000: Generator Loss: 3.793717, Discriminator Loss: 0.299633
Step 20000: Generator Loss: 3.716439, Discriminator Loss: 0.176631
Step 22000: Generator Loss: 4.162383, Discriminator Loss: 0.212528
Step 24000: Generator Loss: 3.983365, Discriminator Loss: 0.253197
Step 26000: Generator Loss: 3.086970, Discriminator Loss: 0.269977
Step 28000: Generator Loss: 3.215426, Discriminator Loss: 0.350482
Step 30000: Generator Loss: 3.515172, Discriminator Loss: 0.298617
Step 32000: Generator Loss: 3.115796, Discriminator Loss: 0.333084
Step 34000: Generator Loss: 3.320462, Discriminator Loss: 0.446589
Step 36000: Generator Loss: 2.987168, Discriminator Loss: 0.417386
Step 38000: Generator Loss: 2.822043, Discriminator Loss: 0.371045
Step 40000: Generator Loss: 2.740946, Discriminator Loss: 0.464534
```

```
Step 42000: Generator Loss: 3.136816, Discriminator Loss: 0.411526
Step 44000: Generator Loss: 2.939960, Discriminator Loss: 0.385261
Step 46000: Generator Loss: 2.284759, Discriminator Loss: 0.399092
Step 48000: Generator Loss: 3.116132, Discriminator Loss: 0.401034
Step 50000: Generator Loss: 2.924815, Discriminator Loss: 0.386848
Step 52000: Generator Loss: 2.809042, Discriminator Loss: 0.365831
Step 54000: Generator Loss: 2.610398, Discriminator Loss: 0.380060
Step 56000: Generator Loss: 2.778062, Discriminator Loss: 0.498709
Step 58000: Generator Loss: 2.868812, Discriminator Loss: 0.523331
Step 60000: Generator Loss: 2.862835, Discriminator Loss: 0.422131
Step 62000: Generator Loss: 3.039068, Discriminator Loss: 0.503733
Step 64000: Generator Loss: 2.840444, Discriminator Loss: 0.496952
Step 66000: Generator Loss: 3.080117, Discriminator Loss: 0.484358
Step 68000: Generator Loss: 2.903224, Discriminator Loss: 0.442403
Step 70000: Generator Loss: 2.672405, Discriminator Loss: 0.555055
Step 72000: Generator Loss: 3.142435, Discriminator Loss: 0.424716
Step 74000: Generator Loss: 2.633177, Discriminator Loss: 0.415394
Step 76000: Generator Loss: 2.820936, Discriminator Loss: 0.546896
Step 78000: Generator Loss: 2.968935, Discriminator Loss: 0.391308
Step 80000: Generator Loss: 2.738734, Discriminator Loss: 0.404072
```

Source Code for: GANs DCGANs

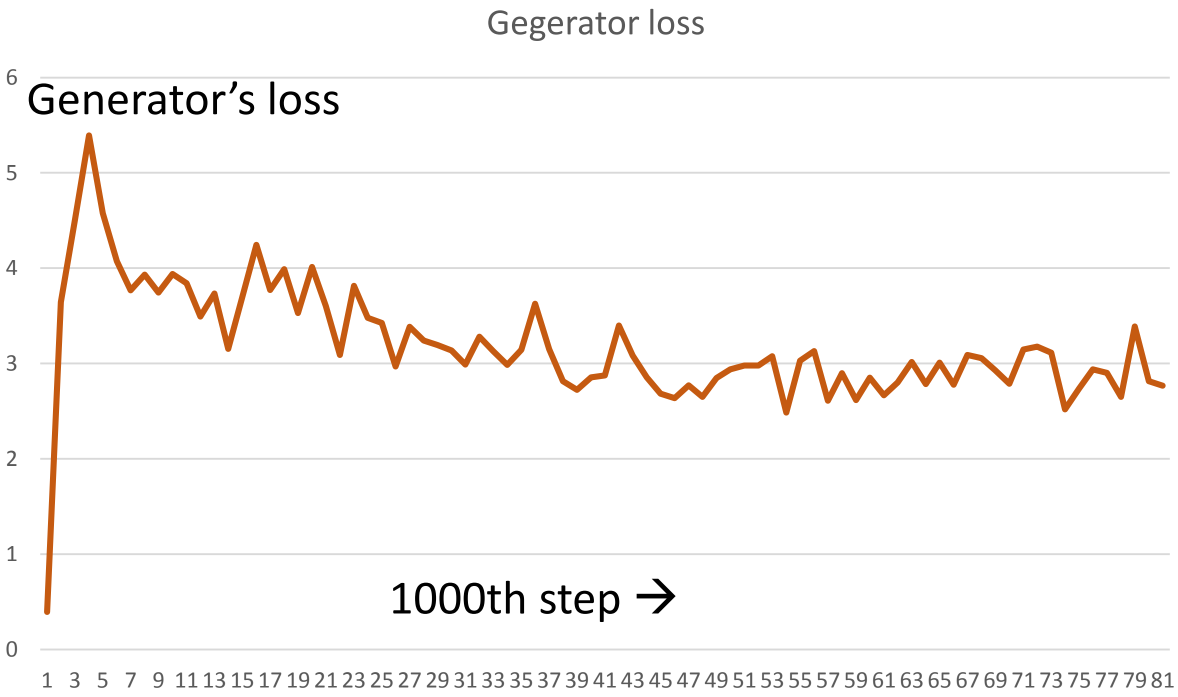
GANs project: Results

Training 80,000 steps:



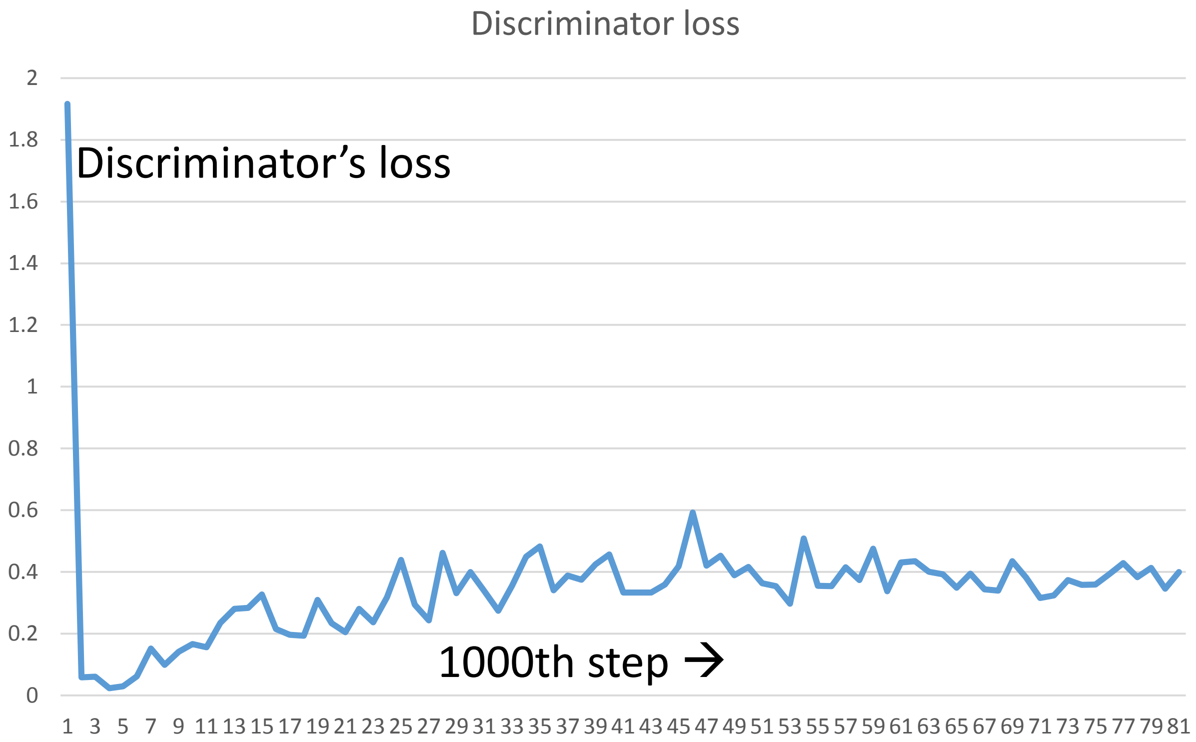
Source Code for: GANs DCGANs

GANs project: Analysis



Source Code for: GANs DCGANs

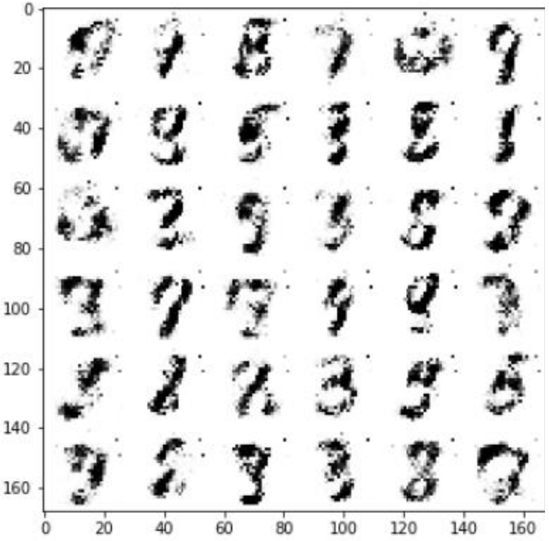
GANs project: Analysis

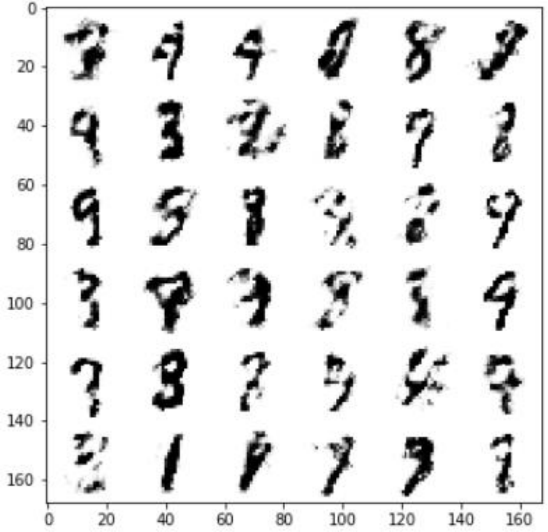


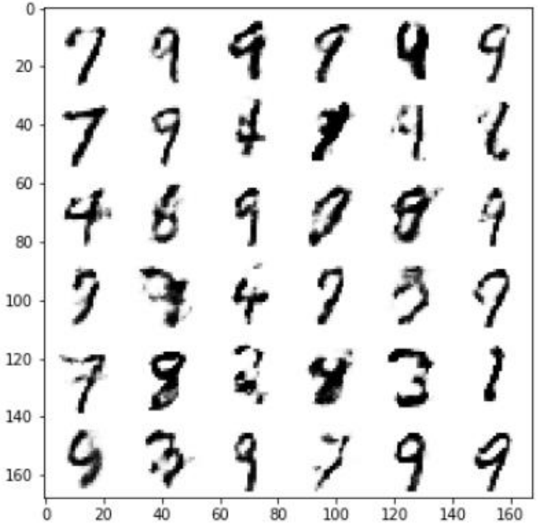
Source Code for: GANs DCGANs

GANs project: Analysis

Step based evaluation:







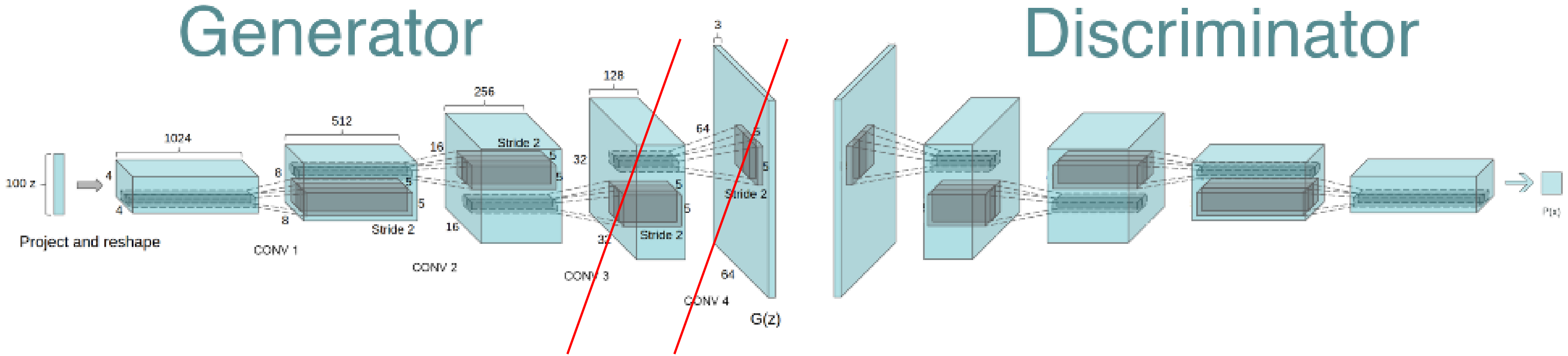
10kth Step

40kth Step

80kth Step

S… CGANs

GANs project: DCGAN
Deep Convolutional Generative Adversarial Network



Source Code for: [GANs](#) [DCGANs](#)

GANs project: DCGAN Deep Convolutional Generative Adversarial Network

References:

- Unsupervised representation learning with deep convolutional generative adversarial networks. A Radford, L Metz, S Chintala, 2016.
- Understanding the difficulty of training deep feedforward neural networks. X Glorot, Y Bengio. Aistats 9, 249-256
- Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Sergey Ioffe, Christian Szegedy. 2015.

Source Code for: GANs DCGANs

UNIVERSITY OF ARKANSAS AT LITTLE ROCK

GANs project: DCGAN
Deep Convolutional Generative Adversarial Network

```
def generator(x, reuse=False):
    with tf.variable_scope('Generator', reuse=reuse):
        # TensorFlow Layers automatically create variables and calculate their
        # shape, based on the input.
        x = tf.layers.dense(x, units=7 * 7 * 128)
        x = tf.layers.batch_normalization(x, training=is_training)
        x = tf.nn.relu(x)
        # Reshape to a 4-D array of images: (batch, height, width, channels)
        # New shape: (batch, 7, 7, 128)
        x = tf.reshape(x, shape=[-1, 7, 7, 128])
        # Deconvolution, image shape: (batch, 14, 14, 64)
        x = tf.layers.conv2d_transpose(x, 64, 5, strides=2, padding='same')
        x = tf.layers.batch_normalization(x, training=is_training)
        x = tf.nn.relu(x)
        # Deconvolution, image shape: (batch, 28, 28, 1)
        x = tf.layers.conv2d_transpose(x, 1, 5, strides=2, padding='same')
        # Apply tanh for better stability - clip values to [-1, 1].
        x = tf.nn.tanh(x)
        return x
```

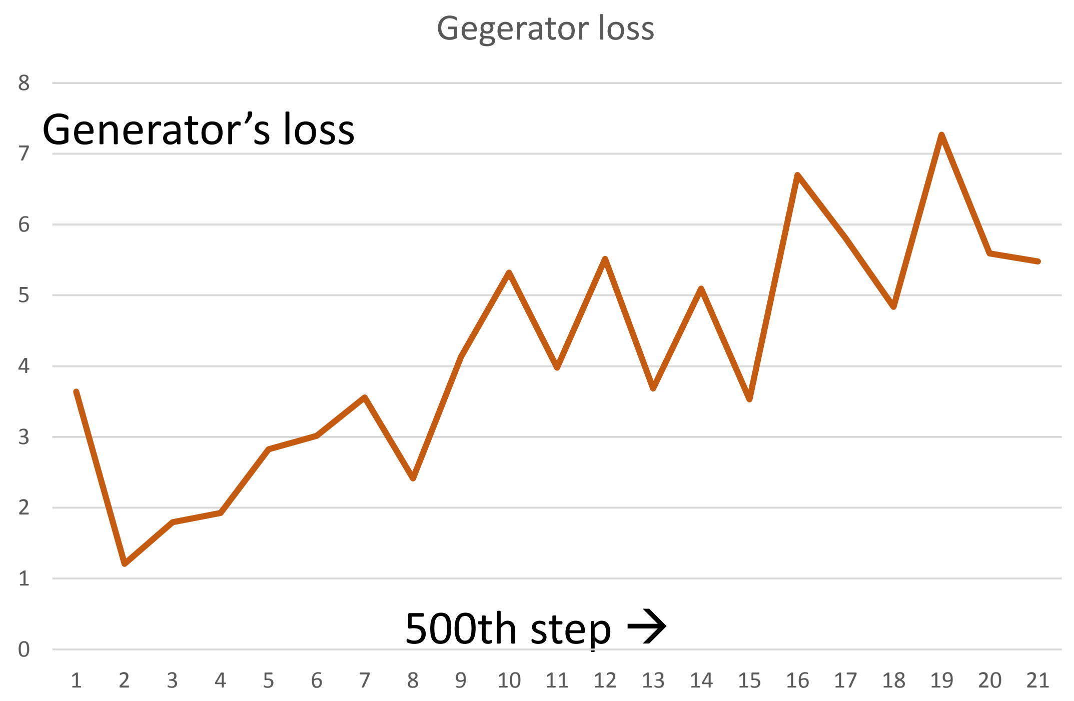
Source Code

GANs project: DCGAN
Deep Convolutional Generative Adversarial Network

```
def discriminator(x, reuse=False):
    with tf.variable_scope('Discriminator', reuse=reuse):
        # Typical convolutional neural network to classify images.
        x = tf.layers.conv2d(x, 64, 5, strides=2, padding='same')
        x = tf.layers.batch_normalization(x, training=is_training)
        x = leakyrelu(x)
        x = tf.layers.conv2d(x, 128, 5, strides=2, padding='same')
        x = tf.layers.batch_normalization(x, training=is_training)
        x = leakyrelu(x)
        # Flatten
        x = tf.reshape(x, shape=[-1, 7*7*128])
        x = tf.layers.dense(x, 1024)
        x = tf.layers.batch_normalization(x, training=is_training)
        x = leakyrelu(x)
        # Output 2 classes: Real and Fake images
        x = tf.layers.dense(x, 2)
    return x
```

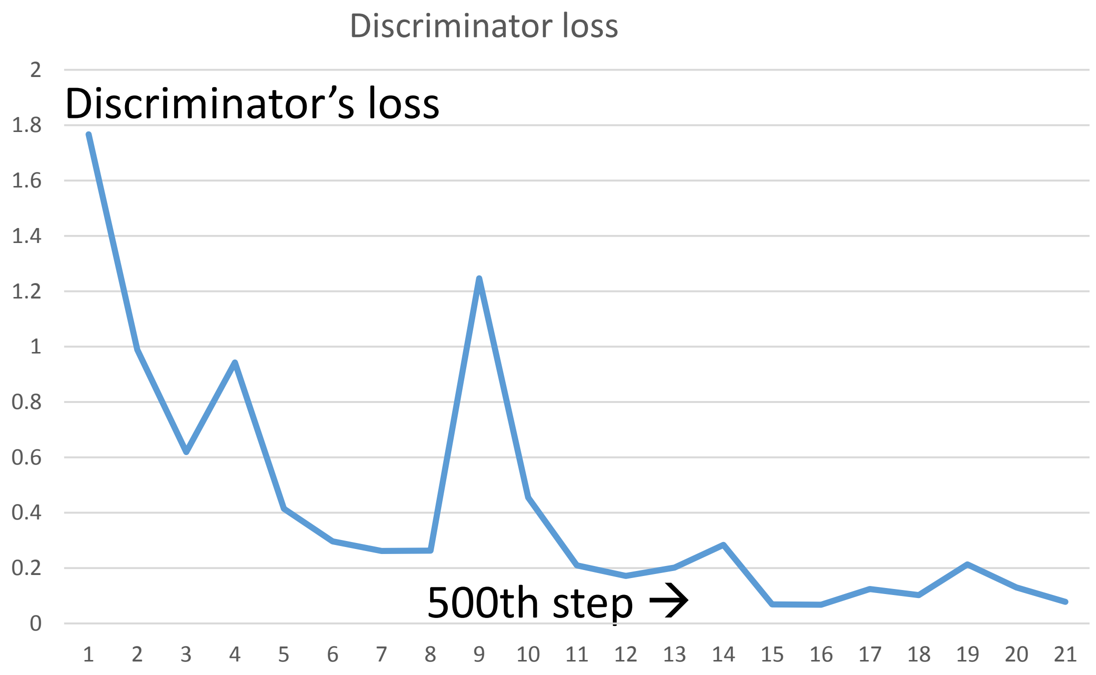
Source Code for: GANs DCGANs

GANs project: Analysis



Source Code for: GANs DCGANs

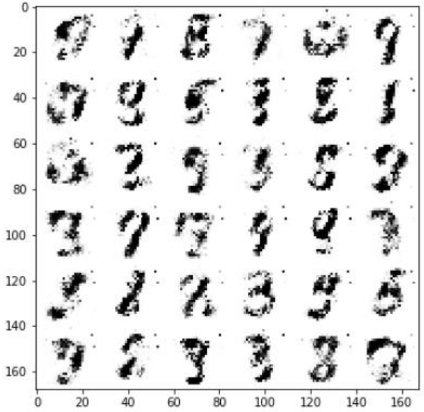
GANs project: Analysis

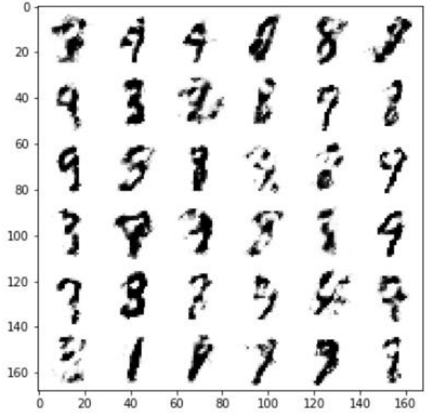


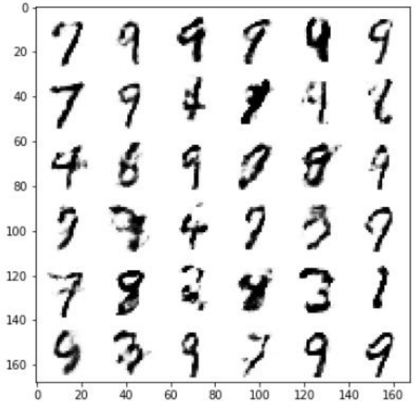
Source Code for: [GANs](#) [DCGANs](#)

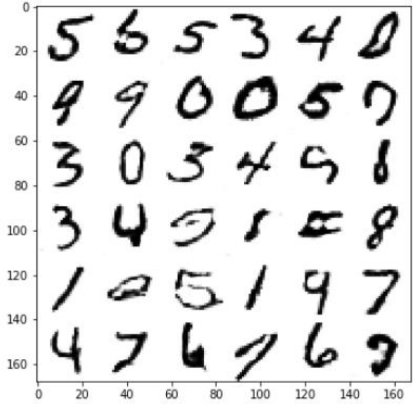
GANs project: Analysis

Step based evaluation:









GAN 10k

GAN 40k

GAN80k

DCGAN 10k

Source Code for: [GANs] [DCGANs]