11_GAN

1 Generative Adversarial Networks

GANs, or Generative Adversarial Networks, are a type of neural network architecture that allow neural networks to generate data. In the past few years, they've become one of the hottest subfields in deep learning, going from generating fuzzy images of digits to photorealistic images of faces.

GANs learn a probability distribution of a dataset by pitting two neural networks against each other. - One model, the **generator**, tries to create images that look very similar to the dataset. - The other model, the **discriminator**, tries to detect whether the images generated were fake or not.

Training generative adversarial networks involve two objectives: 1. The discriminator maximizes the probability of assigning the correct label to both training examples and images generated by the generator. 2. The generator minimizes the probability that the discriminator can predict that what it generates is fake. i.e the generator becomes better at creating fakes.

We'll be using the MNIST, a dataset of handwritten digits to implement GAN.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from tqdm import tqdm
from keras.layers import Input
from keras.models import Model, Sequential
from keras.layers.core import Dense, Dropout
from keras.layers.advanced_activations import LeakyReLU
from keras.datasets import mnist
from tensorflow.keras.optimizers import Adam
from keras import initializers
```

[2]:
```
np.random.seed(10)

# The dimension of our random noise vector.
random_dim = 100
```

[3]:
```
def load_minst_data():
    # load the data
    (x_train, y_train), (x_test, y_test) = mnist.load_data()
    # normalize our inputs to be in the range[-1, 1]
    x_train = (x_train.astype(np.float32) - 127.5)/127.5
    # convert x_train with a shape of (60000, 28, 28) to (60000, 784) so we have
    # 784 columns per row
    x_train = x_train.reshape(60000, 784)
    return (x_train, y_train, x_test, y_test)
```

The generator also needs random input vectors to generate images, and for this, we'll be using numpy.

The GAN Function

The GAN plays a minimax game, where the entire network attempts to optimize the function V(D,G). This is the equation that defines what a GAN is doing:

Now to anyone who isn't well versed in the math behind it, it looks terrifying, but the idea it represents is simple, yet powerful. It's just a mathematical representation of the two objectives as defined above.

The generator is defined by G(z), which converts some noise z we input into some data, like images.

The discriminator is defined by D(x), which outputs the probability that the input x came from the real dataset or not.

We want the predictions on the dataset by the discriminator to be as close to 1 as possible, and on the generator to be as close to 0 as possible. To achieve this, we use the log-likelihood of D(x) and 1-D(z) in the objective function. The log just makes sure that the closer it is to an incorrect value, the more it is penalized.

[4]:
```
def get_optimizer():
    return Adam(lr=0.0002, beta_1=0.5)

def get_generator(optimizer):
    generator = Sequential()
    generator.add(Dense(256, input_dim=random_dim,␣
 ↪kernel_initializer=initializers.RandomNormal(stddev=0.02)))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(512))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(1024))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(784, activation='tanh'))
    generator.compile(loss='binary_crossentropy', optimizer=optimizer)
    return generator
```

The generator is just a vanilla neural network model that takes a random input vector and outputs a 784-dim vector, which, when reshaped, becomes a 28*28 pixel image.

```
[5]: def get_discriminator(optimizer):
         discriminator = Sequential()
         discriminator.add(Dense(1024, input_dim=784,␣
      ↪kernel_initializer=initializers.RandomNormal(stddev=0.02)))
         discriminator.add(LeakyReLU(0.2))
         discriminator.add(Dropout(0.3))

         discriminator.add(Dense(512))
         discriminator.add(LeakyReLU(0.2))
         discriminator.add(Dropout(0.3))

         discriminator.add(Dense(256))
         discriminator.add(LeakyReLU(0.2))
         discriminator.add(Dropout(0.3))

         discriminator.add(Dense(1, activation='sigmoid'))
         discriminator.compile(loss='binary_crossentropy', optimizer=optimizer)
         return discriminator
```

The discriminator is another neural network that takes the output of the previous network, a 784-dimensional vector, and outputs a probability between 0 and 1 that it came from the training dataset.

Compiling it into a GAN

```
[6]: def get_gan_network(discriminator, random_dim, generator, optimizer):
         # We initially set trainable to False since we only want to train either the
         # generator or discriminator at a time
         discriminator.trainable = False
         # gan input (noise) will be 100-dimensional vectors
         gan_input = Input(shape=(random_dim,))
         # the output of the generator (an image)
         x = generator(gan_input)
         # get the output of the discriminator (probability if the image is real or␣
      ↪not)
         gan_output = discriminator(x)
         gan = Model(inputs=gan_input, outputs=gan_output)
         gan.compile(loss='binary_crossentropy', optimizer=optimizer)
         return gan
```

```
[7]: def plot_generated_images(epoch, generator, examples=100, dim=(10, 10),␣
      ↪figsize=(10, 10)):
         noise = np.random.normal(0, 1, size=[examples, random_dim])
         generated_images = generator.predict(noise)
         generated_images = generated_images.reshape(examples, 28, 28)

         plt.figure(figsize=figsize)
         for i in range(generated_images.shape[0]):
```

```
        plt.subplot(dim[0], dim[1], i+1)
        plt.imshow(generated_images[i], interpolation='nearest', cmap='gray_r')
        plt.axis('off')
    plt.tight_layout()
    plt.savefig('gan_generated_image_epoch_%d.png' % epoch)
```

We now compile both models into a single adversarial network, setting the input as a 100-dimensional vector, and the output as the output of the discriminator.

[8]:
```
def train(epochs=1, batch_size=128):

    #1 Get the training and testing data
    x_train, y_train, x_test, y_test = load_minst_data()
    # Split the training data into batches of size 128
    batch_count = x_train.shape[0] / batch_size

    #2. Build our GAN netowrk
    adam = get_optimizer()
    generator = get_generator(adam)
    discriminator = get_discriminator(adam)
    gan = get_gan_network(discriminator, random_dim, generator, adam)

    # 3
    for e in range(1, epochs+1):
        print('-'*15, 'Epoch %d' % e, '-'*15)
        for _ in tqdm(range(int(batch_count))):
            # 4. Get a random set of input noise and images
            noise = np.random.normal(0, 1, size=[batch_size, random_dim])
            image_batch = x_train[np.random.randint(0, x_train.shape[0],␣
↪size=batch_size)]

            # 5. Generate fake MNIST images
            generated_images = generator.predict(noise)
            X = np.concatenate([image_batch, generated_images])

            # 6. Labels for generated and real data
            y_dis = np.zeros(2*batch_size)
            # One-sided label smoothing
            y_dis[:batch_size] = 0.9

            #7. Train discriminator
            discriminator.trainable = True
            discriminator.train_on_batch(X, y_dis)

            #8. Train generator
            noise = np.random.normal(0, 1, size=[batch_size, random_dim])
            y_gen = np.ones(batch_size)
            discriminator.trainable = False
            gan.train_on_batch(noise, y_gen)

        if e == 1 or e % 20 == 0:
            plot_generated_images(e, generator)
```

[9]:
```
train(50, 128)
```

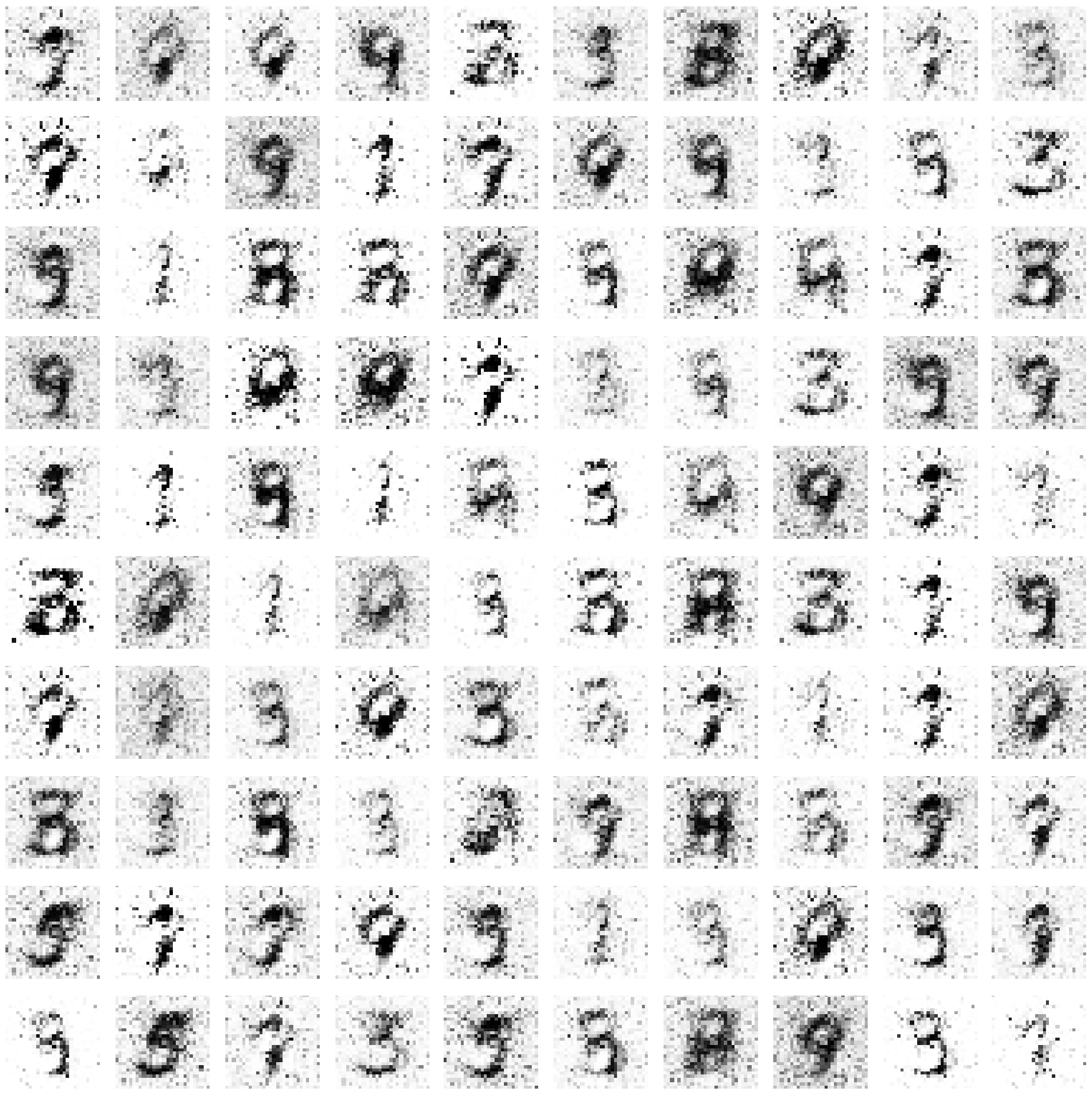
```
C:\Users\Dileep\anaconda3\lib\site-
packages\keras\optimizer_v2\optimizer_v2.py:355: UserWarning: The `lr` argument
is deprecated, use `learning_rate` instead.
  warnings.warn(

--------------- Epoch 1 ---------------

100%|
   | 468/468 [01:20<00:00,  5.82it/s]

--------------- Epoch 2 ---------------

100%|
   | 468/468 [01:13<00:00,  6.36it/s]

--------------- Epoch 3 ---------------

100%|
   | 468/468 [01:04<00:00,  7.22it/s]

--------------- Epoch 4 ---------------

100%|
   | 468/468 [01:01<00:00,  7.55it/s]

--------------- Epoch 5 ---------------

100%|
   | 468/468 [00:59<00:00,  7.91it/s]

--------------- Epoch 6 ---------------

100%|
   | 468/468 [01:01<00:00,  7.65it/s]

--------------- Epoch 7 ---------------

100%|
   | 468/468 [01:04<00:00,  7.27it/s]

--------------- Epoch 8 ---------------

100%|
   | 468/468 [01:04<00:00,  7.27it/s]

--------------- Epoch 9 ---------------
```

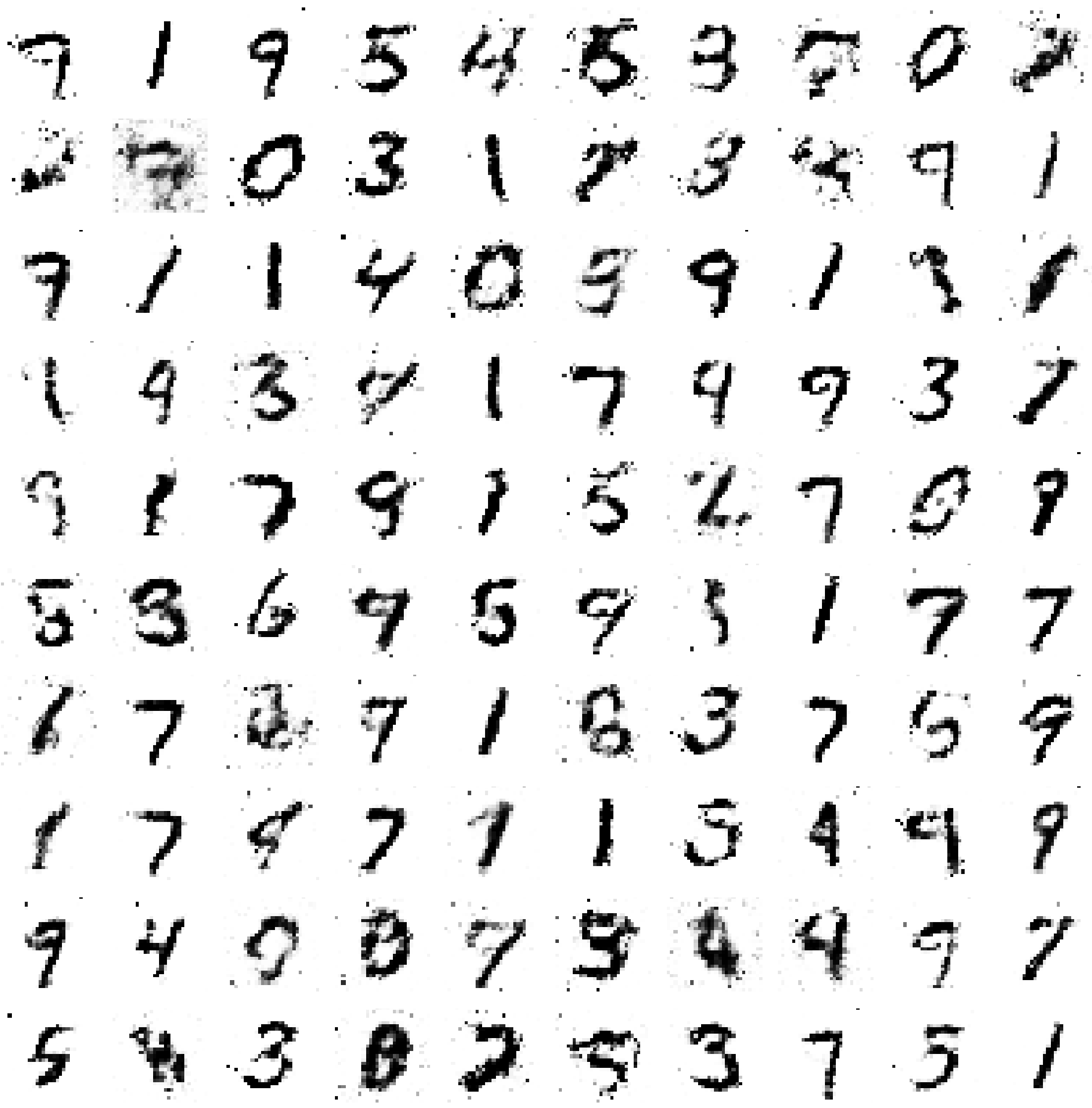
```
100%|
   | 468/468 [01:01<00:00,  7.55it/s]

--------------- Epoch 10 ---------------

100%|
   | 468/468 [01:00<00:00,  7.78it/s]

--------------- Epoch 11 ---------------

100%|
   | 468/468 [01:02<00:00,  7.49it/s]

--------------- Epoch 12 ---------------

100%|
   | 468/468 [01:00<00:00,  7.68it/s]

--------------- Epoch 13 ---------------

100%|
   | 468/468 [01:01<00:00,  7.67it/s]

--------------- Epoch 14 ---------------

100%|
   | 468/468 [01:01<00:00,  7.59it/s]

--------------- Epoch 15 ---------------

100%|
   | 468/468 [01:00<00:00,  7.69it/s]

--------------- Epoch 16 ---------------

100%|
   | 468/468 [01:00<00:00,  7.76it/s]

--------------- Epoch 17 ---------------

100%|
   | 468/468 [01:00<00:00,  7.71it/s]

--------------- Epoch 18 ---------------

100%|
   | 468/468 [01:00<00:00,  7.80it/s]

--------------- Epoch 19 ---------------

100%|
   | 468/468 [00:59<00:00,  7.82it/s]

--------------- Epoch 20 ---------------

100%|
   | 468/468 [00:59<00:00,  7.85it/s]

--------------- Epoch 21 ---------------
```

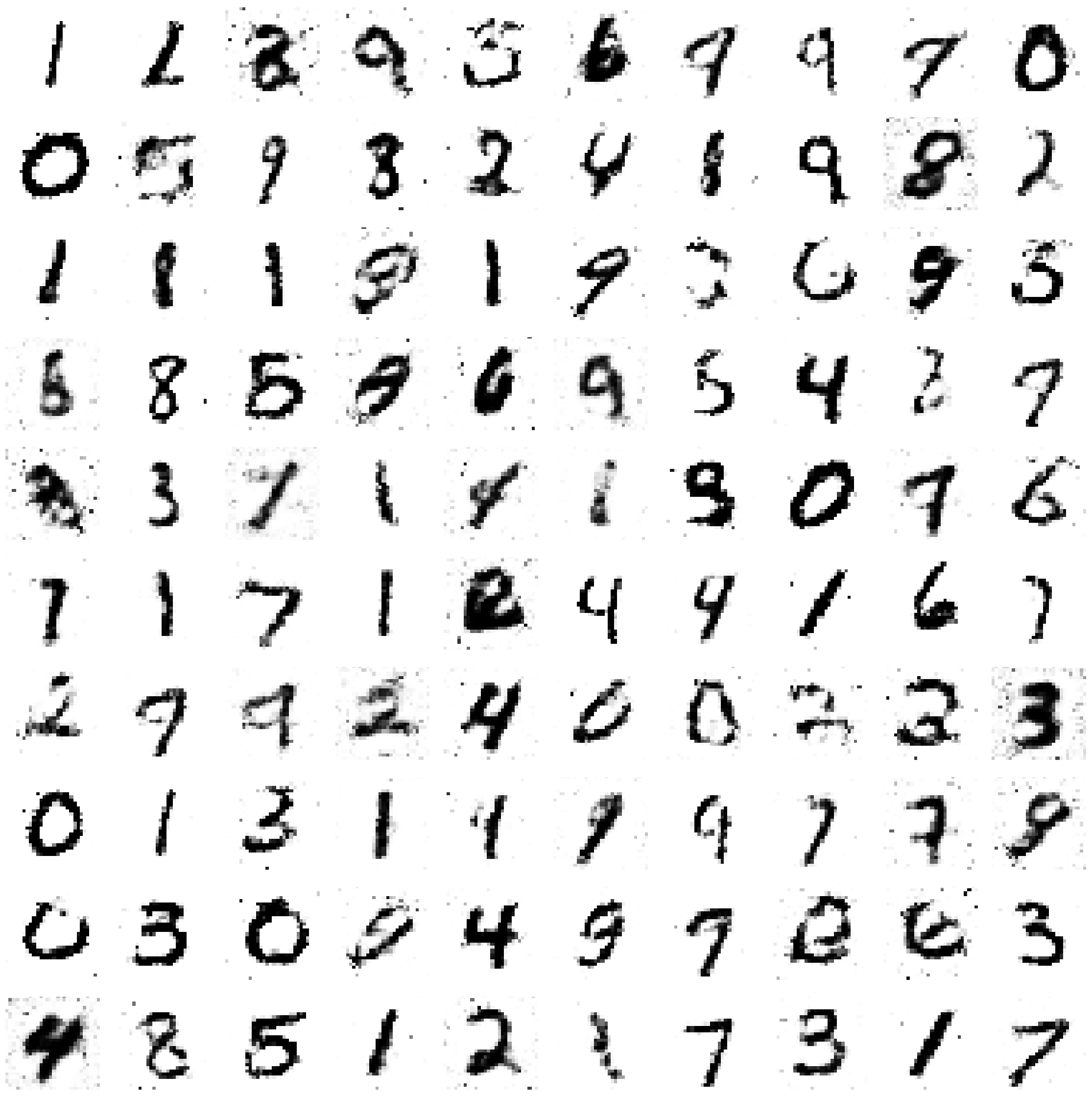
```
100%|
   | 468/468 [00:59<00:00,  7.89it/s]

--------------- Epoch 22 ---------------

100%|
   | 468/468 [00:58<00:00,  7.95it/s]

--------------- Epoch 23 ---------------

100%|
   | 468/468 [01:14<00:00,  6.30it/s]

--------------- Epoch 24 ---------------

100%|
   | 468/468 [01:11<00:00,  6.58it/s]

--------------- Epoch 25 ---------------

100%|
   | 468/468 [01:04<00:00,  7.28it/s]

--------------- Epoch 26 ---------------

100%|
   | 468/468 [01:04<00:00,  7.25it/s]

--------------- Epoch 27 ---------------

100%|
   | 468/468 [01:04<00:00,  7.20it/s]

--------------- Epoch 28 ---------------

100%|
   | 468/468 [01:19<00:00,  5.86it/s]

--------------- Epoch 29 ---------------

100%|
   | 468/468 [01:08<00:00,  6.83it/s]

--------------- Epoch 30 ---------------

100%|
   | 468/468 [01:27<00:00,  5.38it/s]

--------------- Epoch 31 ---------------

100%|
   | 468/468 [01:22<00:00,  5.68it/s]

--------------- Epoch 32 ---------------

100%|
   | 468/468 [01:26<00:00,  5.41it/s]

--------------- Epoch 33 ---------------
```

```
100%|
   | 468/468 [01:25<00:00,  5.49it/s]

--------------- Epoch 34 ---------------

100%|
   | 468/468 [01:12<00:00,  6.42it/s]

--------------- Epoch 35 ---------------

100%|
   | 468/468 [01:13<00:00,  6.39it/s]

--------------- Epoch 36 ---------------

100%|
   | 468/468 [01:11<00:00,  6.53it/s]

--------------- Epoch 37 ---------------

100%|
   | 468/468 [01:05<00:00,  7.10it/s]

--------------- Epoch 38 ---------------

100%|
   | 468/468 [01:06<00:00,  6.99it/s]

--------------- Epoch 39 ---------------

100%|
   | 468/468 [01:05<00:00,  7.13it/s]

--------------- Epoch 40 ---------------

100%|
   | 468/468 [01:07<00:00,  6.94it/s]

--------------- Epoch 41 ---------------

100%|
   | 468/468 [01:06<00:00,  7.01it/s]

--------------- Epoch 42 ---------------

100%|
   | 468/468 [01:06<00:00,  7.08it/s]

--------------- Epoch 43 ---------------

100%|
   | 468/468 [01:05<00:00,  7.13it/s]

--------------- Epoch 44 ---------------

100%|
   | 468/468 [01:04<00:00,  7.23it/s]

--------------- Epoch 45 ---------------
```

```
100%|
   | 468/468 [01:07<00:00,  6.93it/s]

--------------- Epoch 46 ---------------

100%|
   | 468/468 [01:21<00:00,  5.72it/s]

--------------- Epoch 47 ---------------

100%|
   | 468/468 [01:12<00:00,  6.49it/s]

--------------- Epoch 48 ---------------

100%|
   | 468/468 [01:08<00:00,  6.79it/s]

--------------- Epoch 49 ---------------

100%|
   | 468/468 [01:06<00:00,  7.04it/s]

--------------- Epoch 50 ---------------

100%|
   | 468/468 [01:08<00:00,  6.80it/s]
```







1. First, we load the data and split the data into several batches to feed into our model
2. Here we just initialize our GAN network based on the methods defined above
3. This is our training loop, where we run for the specified number of epochs.
4. We generate some random noise and take out some images from our dataset
5. We generate some images using the generator and create a vector X that has some fake images and some real images
6. We create a vector Y which has the "correct answers" that corresponds to X, with the fake images labeled 0 and the real images labeled 0.9. They're labeled 0.9 instead of 1 because it helps the GAN train better, a method called one-sided label smoothing.
7. We need to alternate the training between the discriminator and generator, so over here, we update the discriminator
8. Finally, we update the discriminator.

[ ]:
```
```