Appendix

Generative Adversarial Networks (GAN)

GANs, one of the most important developments in deep learning, were introduced in a NIPS 2014 paper written by a group of researchers at the University of Montreal led by Ian Goodfellow. The GAN model is a way to have two neural netwoks compete with each other. The generator turns random noise into imitations of the data in an attempt to fool the adversarial or discriminator; the adversarial then takes samples from generator model and training data to distinguish genuine data from forgeries created by the generator model. GANs have become widely used in the field of image generation tasks. Real life applicatons of GANs range from text, image, video generation and text-to-image synthesis.

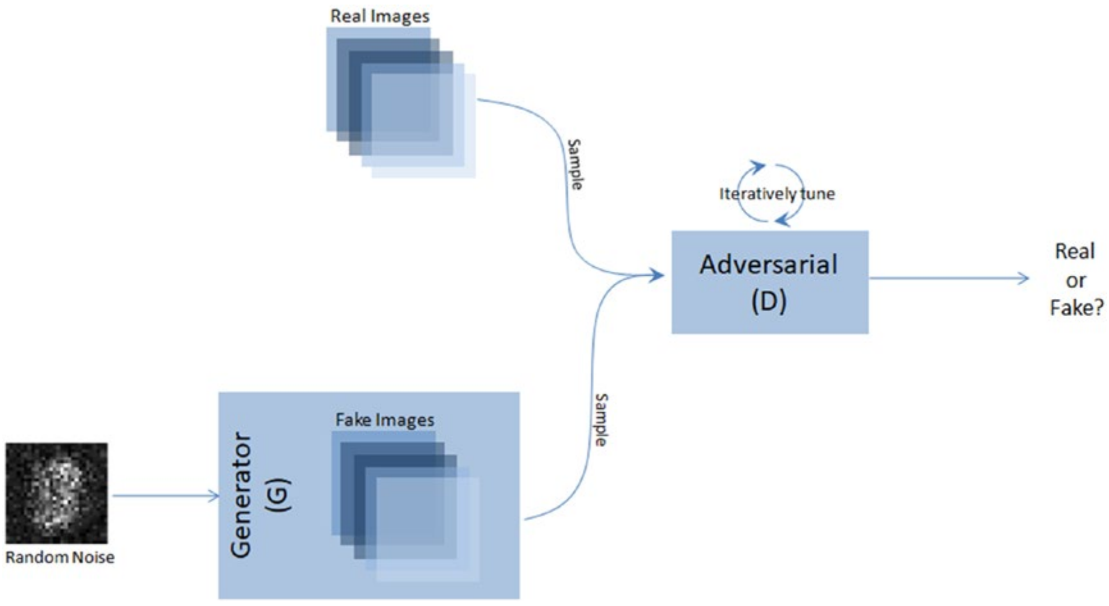


Figure A-1. Generative Adversarial Networks (GAN)

In the following equation, training is a join minimax game where G generates real-looking images to fool an adversarial model and D tries to distinguish between real and fake images by iteratively tuning the model.

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

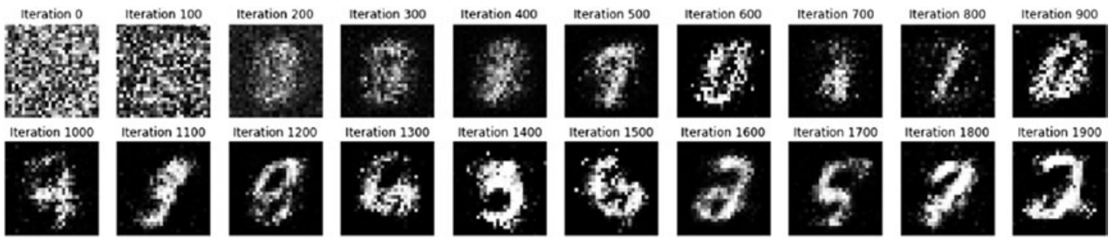
$D_{\Theta d}$ will try to maximize objective such that *D(x)* is close to real 1 and *D(G(z))* is close to fake 0. $G_{\Theta d}$ will try to minimize the objective function such that *D(G(z))* is close to 1. The adversarial is fooled in to thinking generated *G(z)* is real.

Listing A-1. Example code for q-learning

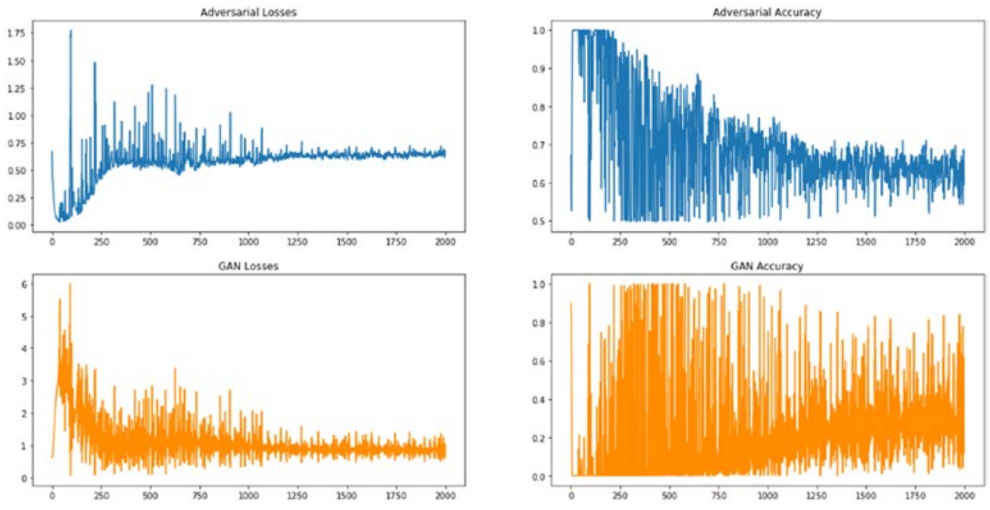
```
import numpy as np
np.random.seed(2017)

from keras.datasets import mnist
from keras.models import Sequential
from keras.models import Model
from keras.layers import Dense, LeakyReLU, BatchNormalization
from keras.optimizers import Adam
from keras import initializers
from tqdm import tnrange

# data
(X_train, y_train), (X_test, y_test) = mnist.load_data()

print('X_train shape:', X_train.shape)

# reshaping the inputs
X_train = X_train.reshape(60000, 28*28)
# normalizing the inputs (-1, 1)
X_train = (X_train.astype('float32') / 255 - 0.5) * 2

print('X_train reshape:', X_train.shape)

# latent space dimension
latent_dim = 100

# imagem dimension 28x28
img_dim = 784
```

```
init = initializers.RandomNormal(stddev=0.02)

# Generator network
generator = Sequential()

# Input layer and hidden layer 1
generator.add(Dense(128, input_shape=(latent_dim,),
kernel_initializer=init))
generator.add(LeakyReLU(alpha=0.2))
generator.add(BatchNormalization(momentum=0.8))

# Hidden layer 2
generator.add(Dense(256))
generator.add(LeakyReLU(alpha=0.2))
generator.add(BatchNormalization(momentum=0.8))

# Hidden layer 3
generator.add(Dense(512))
generator.add(LeakyReLU(alpha=0.2))
generator.add(BatchNormalization(momentum=0.8))

# Output layer
generator.add(Dense(img_dim, activation='tanh'))

# prints a summary representation of your model
generator.summary()

# Adversarial network
adversarial = Sequential()

# Input layer and hidden layer 1
adversarial.add(Dense(128, input_shape=(img_dim,), kernel_
initializer=init))
adversarial.add(LeakyReLU(alpha=0.2))

# Hidden layer 2
adversarial.add(Dense(256))
adversarial.add(LeakyReLU(alpha=0.2))

# Hidden layer 3
adversarial.add(Dense(512))
adversarial.add(LeakyReLU(alpha=0.2))

# Output layer
adversarial.add(Dense(1, activation='sigmoid'))

# Optimizer
optimizer = Adam(lr=0.0004, beta_1=0.5)

adversarial.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['binary_accuracy'])

# prints a summary representation of your model
adversarial.summary()

adversarial.trainable = False

gan_model = Sequential()
gan_model.add(generator)
gan_model.add(adversarial)
gan_model.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['binary_accuracy'])

# prints a summary representation of your model
gan_model.summary()

def train(batch_size=256, train_steps=2000):
    adversarial_losses = []
    gan_losses = []
    sample_images = []

    for i in tnrange(train_steps):

        # Select a random sample from the training data
        images_train = X_train[np.random.randint(0, X_train.shape[0],
        size=batch_size)]

        # Create noise as input for the generator to generate the fake images
        noise = np.random.normal(loc=0, scale=1, size=(batch_size, 100))
        images_fake = generator.predict(noise)

        # Create input by concatenate both real and fake images and
        assigning the respective labels
        input_data = np.concatenate((images_train, images_fake))
        input_labels = np.ones([2*batch_size, 1])
        input_labels[batch_size:, :] = 0
        adversarial_loss = adversarial.train_on_batch(input_data, input_
        labels)

        # Train the adversarial model to generate more realistic images
        input_labels = np.ones([batch_size, 1])
        noise = np.random.normal(loc=0, scale=1, size=(batch_size, 100))
        gan_loss = gan_model.train_on_batch(noise, input_labels)

        adversarial_losses.append(adversarial_loss)
        gan_losses.append(gan_loss)

        if i % 100 == 0:
            noise = np.random.normal(loc=0, scale=1, size=(batch_size,
            100))
            fake_images = generator.predict(noise)
            sample_images.append(fake_images[0])

    return adversarial_losses, gan_losses, sample_images

adversarial_losses, gan_losses, sample_images = train()

plt.figure(figsize=(20,4))

for i, fake_image in enumerate(sample_images, 0):
    plt.subplot(2, 10, i+1)
    plt.imshow(np.reshape(fake_image, (28, 28)), cmap='gray')
    plt.title("Iteration %d" % (i * 100))
    plt.axis('off')

plt.figure(figsize=(20,10))

plt.subplot(2,2,1)
plt.plot(np.array(adversarial_losses)[:, 0])
plt.title("Adversarial Losses")

plt.subplot(2,2,2)
plt.plot(np.array(adversarial_losses)[:, 1])
plt.title("Adversarial Accuracy")
plt.subplot(2,2,3)
plt.plot(np.array(gan_losses)[:, 0], color='darkorange')
plt.title("GAN Losses")

plt.subplot(2,2,4)
plt.plot(np.array(gan_losses)[:, 1], color='darkorange')
plt.title("GAN Accuracy")

----- output ------
```

Evolution of generated images



Training Histroy



The area of GAN has grown rapidly in recent years. Here is a link that lists various GAN versions based on the research paper referenced at the beginning of this appendix. https://github.com/hindupuravinash/the-gan-zoo.