Day 20

Date 29 June 2024

Daily Report

Today's training session was based on GANs(Generative Adversarial Nets).

Today's Topic

GANs (Generative Adversarial Nets)

Componenet of GANs:-

- Generator:- generative network is typically a convolutional neural network (with deconvolution layers). This network takes some noise vector and outputs an image. When training the generative network, it learns which areas of the image to improve/change so that the discriminator would have a harder time differentiating its generated images from the real ones.
- Discriminator:- discriminator network is usually a convolutional neural network (since GANs are mainly used for image tasks) which assigns a probability that the image is real.

The generative network keeps producing images that are closer in appearance to the real images while the discriminative network is trying to determine the differences between real and fake images. The ultimate goal is to have a generative network that can produce images which are indistinguishable from the real ones.

Library used

- matplotlib - used for plotting
- tensorflow, Keras - used for backend library
- tqdm - used to show fancy bar for each epoch

Steps for GANs

1. Import Libraries

```
import os
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm

from keras.layers import Input
from keras.models import Model, Sequential
from tensorflow.keras.layers import Dense, Dropout, LeakyReLU
from tensorflow.keras.datasets import mnist
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import initializers
```

2. Initial Configurations for a Machine Learning model

```
# Let Keras know that program is using tensorflow as our backend engine
os.environ["KERAS_BACKEND"] = "tensorflow"

# To make sure that program can reproduce the experiment and get the same results
np.random.seed(10)

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

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-3c8a270a31c4> in <cell line: 2>()
      1 # Let Keras know that program is using tensorflow as our backend engine
----> 2 os.environ["KERAS_BACKEND"] = "tensorflow"
      3
      4 # To make sure that program can reproduce the experiment and get the same
results
      5 np.random.seed(10)

NameError: name 'os' is not defined
```

3. Load the data using MNIST

```
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)
```

4. Discriminator and Generator Network

```
# You will use the Adam optimizer
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.RandomNor
    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

def get_discriminator(optimizer):
    discriminator = Sequential()
    discriminator.add(Dense(1024, input_dim=784, kernel_initializer=initializers.RandomNorma
    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
```

Double-click (or enter) to edit

```
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



# Create a wall of generated MNIST images
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)
    plt.close()
```

```
    def train(epochs=1, batch_size=128):
        # Get the training and testing data
```