Open in Colab

ENGS 108 Fall 2020 Assignment 7

Due November 18, 2020 at 11:59PM on Canvas

Instructors: George Cybenko

TAs: Chase Yakaboski

Rules and Requirements

1. You are only allowed to use Python packages that are explicity imported in the assignment notebook or are standard (bultin) python libraries like random, os, sys, etc, (Standard Bultin Python libraries will have a Python.org documentation). For this assignment you may use:
    - numpy
    - pandas
    - scikit-learn
    - matplotlib
    - tensorflow
2. All code must be fit into the designated code or text blocks in the assignment notebook. They are indentified by a **TODO** qualifier.
3. For analytical questions that don't require code, type your answer cleanly in Markdown. For help, see the Google Colab Markdown Guide.

In [ ]:
```
'''Import Statements'''
import tensorflow as tf
import numpy as np
import tqdm.notebook as tq
import os
import PIL
from tensorflow.keras import layers
import time
import matplotlib.pyplot as plt
```

Problem 1: MNIST GANs

Use the MNIST dataset to train 10 Generative Adversarial Networks to create hand writtencharacters using the MNIST samples. You can build on the Matlab example or use any othercode as a starting point. Submit your code for the GAN you used.

> **Part 1** Load in the MNIST dataset and seperate the dataset into 10 seperate datasets for each number class, i.e. 0, 1, 2, ..., 9.

In [ ]:
```
# Load data
(train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()

#TODO: Seperate the training data using the train labels into 10 seperate datasets.

#TODO: Then build a TF dataset for each of the seperated out numpy datasets.
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

> **Part 2** Make the generator and the discriminator of the GAN. Feel free to use this tutorial for building the generator and discriminator. Maybe change the architecture a little bit tho so it's not just straight plagiarism.

In [ ]:
```
#TODO: Build a function that builds the generator network.
def make_generator_model():
  return model
#TODO: Build a function that builds the discriminator network.
def make_discriminator_model():
  return model
```

> **Part 3** Define your loss functions for the models.

In [ ]:
```
#TODO: Make your generator and discriminiator loss functions and assign a optimizer for your generator and discriminator.
# This method returns a helper function to compute cross entropy loss
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

def discriminator_loss(real_output, fake_output):
  return loss

def generator_loss(fake_output):
  return loss
```

> **Part 4** Using the training step function provided build a training loop and train 10 GANs using your 10 datasets so that each GAN is trained on a seperate dataset. After training, you should have 10 GANs, where each GAN will generate one respective number, i.e. 0, 1, 2, ..., 9. **Make sure to enable GPU acceleration for faster training. Go to *Edit > Notebook Settings > Hardware Accelerator* and click GPU.** *Note: You'll have to rerun your notebook after you do this.*

In [ ]:
```
# Some code to help save checkpoints in case your notebook crashes
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
```

In [ ]:
```
noise_dim = 100

# Provided training step function taken from cited tutorial but modified a little. Pay attention to the modification!
def train_step_fn():
  @tf.function
  def train_step(images, generator, discriminator, generator_optimizer, discriminator_optimizer):
      noise = tf.random.normal([BATCH_SIZE, noise_dim])

      with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)

        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)

        gen_loss = generator_loss(fake_output)
        disc_loss = discriminator_loss(real_output, fake_output)

      gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
      gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

      generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
      discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))
  return train_step
```

In [ ]:
```
#TODO: Finish implementing this train function. It doesn't have to return anything
def train(dataset, epochs, generator, discriminator, generator_optimizer, discriminator_optimizer, checkpoint):
  for epoch in tq.tqdm(range(epochs), desc='Completed Epochs'):
    start = time.time()

    #TODO: Implement training step

    # Save the model every 15 epochs
    if (epoch + 1) % 15 == 0:
      checkpoint.save(file_prefix = checkpoint_prefix)
```

In [ ]:
```
#TODO: Finish the training loop.
EPOCHS = 50
gans = []

for dataset in tf_datasets:
  #TODO: Define your optimizers, i.e. change the Nones to something.
  generator_optimizer = None
  discriminator_optimizer = None
  # Instaniate your models
  gen = make_generator_model()
  discrim = make_discriminator_model()
  # Reinitialize the train_step function
  train_step = train_step_fn()
  # Put in your checkoutpoint for each gan
  checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
                                   discriminator_optimizer=discriminator_optimizer,
                                   generator=gen,
                                   discriminator=discrim)
  # Train the model
  train(dataset, EPOCHS, gen, discrim, generator_optimizer, discriminator_optimizer, checkpoint)
  # Save the generator in the gans list
  gans.append(gen)
```

> **Part 5** Using a random seed as the generator input, plot the generated number from each of the ten GANs. Use the same seed for each GAN.

In [ ]:
```
#TODO: Initialize a random seed as input into each gan.

#TODO: Go through each GAN, output the result using the seed and plot the result.
```

Problem 2: MNIST RCO

Use the 10 trained generators to create the following application which we will call RCO(reverse of OCR).

- Your RCO will read a string of digits and will output the corresponding GAN generated samples of handwritten digits as a single png image of the string of digits.

In [ ]:
```
#TODO: Implement the following function
def rco(string_of_numbers):
  generated_numbers = []
  for num in string_of_numbers:
    #TODO: Convert the number to an int
    #TODO: Index your trained GANs list with that int
    #TODO: Generate a image using your seed from Part 5.
    #TODO: Append that img to generated_numbers list

  #TODO: Now go through your generated numbers a create a matplotlib plot of all the numbers in the string.
  # Maybe increase figure size if too small.
  fig = plt.figure(15, 15)
  for i, gen_num in enumerate(generated_numbers):
    plt.subplot(1, len(generated_numbers), i+1)
    #TODO: Implement the image show.
    plt.imshow()
    #TODO: Maybe turn off the axis
  plt.show()
```