Worksheet 2

Deep Learning: fully connected MNIST network

1. The goal is to train a fully connected network for the popular MNIST dataset. You will use the code fragments below as a guide to perform this task. Use the MIT code libraries found at: www.deeplearning.ai

```
%tensorflow_version 2.x
import tensorflow as tf

!pip install mitdeeplearning
import mitdeeplearning as mdl

import matplotlib.pyplot as plt
import numpy as np
import random
from tqdm import tqdm

#download and load the MNIST dataset
#The training set is made up of 28x28 grayscale images of handwritten digits.

mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = (np.expand_dims(train_images, axis=-1)/255.).astype(np.float32
    )
train_labels = (train_labels).astype(np.int64)
test_images = (np.expand_dims(test_images, axis=-1)/255.).astype(np.float32)
test_labels = (test_labels).astype(np.int64)

#use the Keras API and define the model using the Sequential class.
#First use a Flatten layer, which flattens the input so that it can be fed
    into the model.

#In this next block, define the fully connected layers of this simple work.
#Use a sigmoid activation

def build_fc_model():
  fc_model = tf.keras.Sequential([
      # First define a Flatten layer
      tf.keras.layers.Flatten(),

      # '''TODO: Define the activation function for the first fully connected
          (Dense) layer.'''
      tf.keras.layers.Dense(128, activation= '''TODO'''),

      # '''TODO: Define the second Dense layer to output the classification
         probabilities'''
```

```
      '''TODO: Dense layer to output classification probabilities'''
  ])
  return fc_model

model = build_fc_model()

#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#Compile the model with appropriate optimizer and loss function

model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=1e-1),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


#train the fully connected model


# Define the batch size and the number of epochs to use during training
BATCH_SIZE = 64
EPOCHS = 5

model.fit(train_images, train_labels, batch_size=BATCH_SIZE, epochs=EPOCHS)


#Use the evaluate method to evaluate the model on the test dataset!


'''TODO: Use the evaluate method to test the model!'''
test_loss, test_acc = # TODO

print('Test accuracy:', test_acc)
```

2. Compare the performance of other loss functions:

```
mse = tf.keras.losses.MeanSquaredError()

mape = tf.keras.losses.MeanAbsolutePercentageError()
```