1.1 Assignment

a. Run Keras MNIST MLP Example

In [1]:

```
'''Trains a simple deep NN on the MNIST dataset.
Gets to 98.40% test accuracy after 20 epochs
(there is *a lot* of margin for parameter tuning).
2 seconds per epoch on a K520 GPU.
'''

from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import RMSprop

batch_size = 128
num_classes = 10
epochs = 20

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
60000 train samples
10000 test samples
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 512)               401920    
_________________________________________________________________
dropout (Dropout)            (None, 512)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656    
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130      
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
469/469 [==============================] - 5s 10ms/step - loss: 0.2466 - accu
racy: 0.9232 - val_loss: 0.1056 - val_accuracy: 0.9680
Epoch 2/20
469/469 [==============================] - 4s 9ms/step - loss: 0.1058 - accur
acy: 0.9687 - val_loss: 0.0757 - val_accuracy: 0.9784
Epoch 3/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0759 - accu
racy: 0.9772 - val_loss: 0.0704 - val_accuracy: 0.9790
Epoch 4/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0611 - accu
racy: 0.9815 - val_loss: 0.0814 - val_accuracy: 0.9777
Epoch 5/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0510 - accur
acy: 0.9858 - val_loss: 0.0734 - val_accuracy: 0.9811
Epoch 6/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0448 - accu
racy: 0.9863 - val_loss: 0.0686 - val_accuracy: 0.9829
Epoch 7/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0388 - accur
acy: 0.9878 - val_loss: 0.0846 - val_accuracy: 0.9821
Epoch 8/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0345 - accur
acy: 0.9898 - val_loss: 0.0784 - val_accuracy: 0.9817
Epoch 9/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0321 - accur
acy: 0.9909 - val_loss: 0.0866 - val_accuracy: 0.9832
Epoch 10/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0284 - accur
acy: 0.9918 - val_loss: 0.0882 - val_accuracy: 0.9829
Epoch 11/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0252 - accur
acy: 0.9925 - val_loss: 0.0998 - val_accuracy: 0.9843
Epoch 12/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0246 - accur
acy: 0.9927 - val_loss: 0.1062 - val_accuracy: 0.9834
Epoch 13/20
```

```
469/469 [==============================] - 4s 9ms/step - loss: 0.0244 - accur
acy: 0.9933 - val_loss: 0.0997 - val_accuracy: 0.9833
Epoch 14/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0217 - accur
acy: 0.9935 - val_loss: 0.1026 - val_accuracy: 0.9841
Epoch 15/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0187 - accur
acy: 0.9949 - val_loss: 0.1001 - val_accuracy: 0.9841
Epoch 16/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0199 - accur
acy: 0.9946 - val_loss: 0.1196 - val_accuracy: 0.9835
Epoch 17/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0191 - accur
acy: 0.9946 - val_loss: 0.1169 - val_accuracy: 0.9824
Epoch 18/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0195 - accur
acy: 0.9948 - val_loss: 0.1182 - val_accuracy: 0.9848
Epoch 19/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0186 - accur
acy: 0.9948 - val_loss: 0.1281 - val_accuracy: 0.9832
Epoch 20/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0164 - accur
acy: 0.9955 - val_loss: 0.1207 - val_accuracy: 0.9840
Test loss: 0.12066762894392014
Test accuracy: 0.984000027179718
```

b. Run PySpark Example

In [5]:

```
#
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

import sys
from random import random
from operator import add

from pyspark.sql import SparkSession


if __name__ == "__main__":
    """
        Usage: pi [partitions]
    """
    spark = SparkSession\
        .builder\
        .appName("PythonPi")\
        .getOrCreate()

    partitions = int(sys.argv[1]) if len(sys.argv) > 1 & sys.argv[1].isdigit()==T
    n = 100000 * partitions

    def f(_):
        x = random() * 2 - 1
        y = random() * 2 - 1
        return 1 if x ** 2 + y ** 2 <= 1 else 0

    count = spark.sparkContext.parallelize(range(1, n + 1), partitions).map(f).re
    print("Pi is roughly %f" % (4.0 * count / n))

    spark.stop()
```

```
Pi is roughly 3.134960
```

I added one logic to handle the partitions logic since argument is coming as String.

In [ ]: