```
'''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])
```

```
 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: 669706 (2.55 MB)
Trainable params: 669706 (2.55 MB)
```

```
Epoch 4/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0567 - accuracy: 0.9822 - val_loss: 0.0810 - val_accuracy: 0.9760
Epoch 5/20
469/469 [==============================] - 6s 14ms/step - loss: 0.0472 - accuracy: 0.9849 - val_loss: 0.0657 - val_accuracy: 0.9804
Epoch 6/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0397 - accuracy: 0.9877 - val_loss: 0.0755 - val_accuracy: 0.9788
Epoch 7/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0343 - accuracy: 0.9890 - val_loss: 0.0646 - val_accuracy: 0.9822
Epoch 8/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0299 - accuracy: 0.9906 - val_loss: 0.0676 - val_accuracy: 0.9821
Epoch 9/20
469/469 [==============================] - 10s 20ms/step - loss: 0.0254 - accuracy: 0.9922 - val_loss: 0.0742 - val_accuracy: 0.9789
Epoch 10/20
469/469 [==============================] - 6s 14ms/step - loss: 0.0237 - accuracy: 0.9924 - val_loss: 0.0745 - val_accuracy: 0.9815
Epoch 11/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0199 - accuracy: 0.9935 - val_loss: 0.0700 - val_accuracy: 0.9823
Epoch 12/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0187 - accuracy: 0.9941 - val_loss: 0.0642 - val_accuracy: 0.9848
Epoch 13/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0167 - accuracy: 0.9946 - val_loss: 0.0704 - val_accuracy: 0.9849
Epoch 14/20
469/469 [==============================] - 7s 15ms/step - loss: 0.0166 - accuracy: 0.9948 - val_loss: 0.0754 - val_accuracy: 0.9842
Epoch 15/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0154 - accuracy: 0.9948 - val_loss: 0.0724 - val_accuracy: 0.9849
Epoch 16/20
469/469 [==============================] - 7s 14ms/step - loss: 0.0140 - accuracy: 0.9954 - val_loss: 0.0831 - val_accuracy: 0.9831
Epoch 17/20
469/469 [==============================] - 7s 14ms/step - loss: 0.0127 - accuracy: 0.9959 - val_loss: 0.0783 - val_accuracy: 0.9847
Epoch 18/20
469/469 [==============================] - 7s 14ms/step - loss: 0.0124 - accuracy: 0.9961 - val_loss: 0.0783 - val_accuracy: 0.9841
Epoch 19/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0108 - accuracy: 0.9962 - val_loss: 0.0778 - val_accuracy: 0.9846
Epoch 20/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0097 - accuracy: 0.9967 - val_loss: 0.0856 - val_accuracy: 0.9850
Test loss: 0.08555874228477478
Test accuracy: 0.9850000143051147
```

```
import sys
from random import random
from operator import add

from pyspark.sql import SparkSession

if __name__ == "__main__":
    spark = SparkSession.builder.appName("PythonPi").getOrCreate()

    # Set the number of partitions directly
    partitions = 4  # Adjust this value as needed
    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).reduce(add)
    print("Pi is roughly %f" % (4.0 * count / n))

    spark.stop()
```

```
Pi is roughly 3.136680
```

✓ 5s completed at 10:58 AM ● ×