assignment01_MuleyTushar

December 2, 2021

Name: Muley, Tushar Assignment: Week1-Assignment 1.1 Examples mnist_mlp and Pyspark

Example: mnist_mlp

[1]:

```python
'''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()
```

```python
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])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
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 11ms/step - loss: 0.2454 -
accuracy: 0.9234 - val_loss: 0.1155 - val_accuracy: 0.9644
```

```
Epoch 2/20
469/469 [==============================] - 5s 10ms/step - loss: 0.1030 -
accuracy: 0.9685 - val_loss: 0.0830 - val_accuracy: 0.9752
Epoch 3/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0737 -
accuracy: 0.9780 - val_loss: 0.0793 - val_accuracy: 0.9774
Epoch 4/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0593 -
accuracy: 0.9822 - val_loss: 0.0803 - val_accuracy: 0.9784
Epoch 5/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0493 -
accuracy: 0.9854 - val_loss: 0.0757 - val_accuracy: 0.9801
Epoch 6/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0448 -
accuracy: 0.9875 - val_loss: 0.0801 - val_accuracy: 0.9805
Epoch 7/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0376 -
accuracy: 0.9890 - val_loss: 0.0944 - val_accuracy: 0.9812
Epoch 8/20
469/469 [==============================] - 4s 10ms/step - loss: 0.0332 -
accuracy: 0.9898 - val_loss: 0.0959 - val_accuracy: 0.9811
Epoch 9/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0321 -
accuracy: 0.9906 - val_loss: 0.0869 - val_accuracy: 0.9827
Epoch 10/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0292 -
accuracy: 0.9916 - val_loss: 0.0924 - val_accuracy: 0.9826
Epoch 11/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0279 -
accuracy: 0.9922 - val_loss: 0.0899 - val_accuracy: 0.9827
Epoch 12/20
469/469 [==============================] - 4s 10ms/step - loss: 0.0261 -
accuracy: 0.9928 - val_loss: 0.0996 - val_accuracy: 0.9811
Epoch 13/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0256 -
accuracy: 0.9934 - val_loss: 0.1012 - val_accuracy: 0.9830
Epoch 14/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0225 -
accuracy: 0.9940 - val_loss: 0.1180 - val_accuracy: 0.9815
Epoch 15/20
469/469 [==============================] - 4s 10ms/step - loss: 0.0228 -
accuracy: 0.9936 - val_loss: 0.1134 - val_accuracy: 0.9828
Epoch 16/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0227 -
accuracy: 0.9942 - val_loss: 0.1130 - val_accuracy: 0.9827
Epoch 17/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0199 -
accuracy: 0.9945 - val_loss: 0.1196 - val_accuracy: 0.9844
```

```
Epoch 18/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0190 -
accuracy: 0.9946 - val_loss: 0.1136 - val_accuracy: 0.9837
Epoch 19/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0203 -
accuracy: 0.9947 - val_loss: 0.1448 - val_accuracy: 0.9824
Epoch 20/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0184 -
accuracy: 0.9956 - val_loss: 0.1211 - val_accuracy: 0.9838
Test loss: 0.12112750113010406
Test accuracy: 0.9837999939918518
```

Example Pyspark

[3]:

```python
#
# 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 else 2
```

```
partitions = 2
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 3.142900
```

[ ]: