optimizers-comparison-in-dl

March 2, 2024

[1]:
```
#IMPORTING LIBRARIES

import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import Dense, Dropout,Flatten
```

```
2024-03-02 06:26:07.111044: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-03-02 06:26:07.111192: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-03-02 06:26:07.257062: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
```

0.0.1 LOADING DATA

[2]:
```
(X_train,y_train),(X_test,y_test)=mnist.load_data()
```

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

0.0.2 DATA RESHAPE

[3]:
```
X_train=X_train.reshape(X_train.shape[0],28,28,1)
X_test=X_test.reshape(X_test.shape[0],28,28,1)
```

[4]:
```
X_train=X_train.astype('float32')
X_test=X_test.astype('float32')
X_train /=255
```

```
X_test /=255
y_train=tf.keras.utils.to_categorical(y_train)
y_test=tf.keras.utils.to_categorical(y_test)
```

0.0.3 BUILD OPTIMIZER CALL

[5]:
```
def build_optimizer(op):
    model=tf.keras.Sequential()
    model.add(tf.keras.Input(shape=(28,28,1)))
    model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), strides=1,␣
 ↪activation='relu'))
    model.add(tf.keras.layers.MaxPool2D())
    model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), strides=1,␣
 ↪activation='relu'))
    model.add(tf.keras.layers.Dropout(0.25))
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(128, activation='relu'))
    model.add(tf.keras.layers.Dense(256, activation='relu'))
    model.add(tf.keras.layers.Dropout(0.5))
    model.add(tf.keras.layers.Dense(10, activation='softmax'))
    model.compile(optimizer=op, loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
    return model
```

0.0.4 COMPARING EACH OPTIMIZER ACCURACY

[7]:
```
import os, gc

optimizers=['Adam', 'RMSprop','Adadelta', 'Adagrad',  'SGD']
opt_res=[]
model_res=[]
for i in optimizers:
    model=build_optimizer(i)
    print("Accuracy for: ",i)
    print("\n")
    history=model.fit(X_train,y_train, epochs=5, batch_size=64,verbose=1,␣
 ↪validation_data=(X_test, y_test))
    print("\n")
    gc.collect()
    model_res.append(history)
    opt_res.append(history.history['accuracy'])
```

```
Accuracy for:  Adam


Epoch 1/5
938/938              34s 34ms/step -
```

```
accuracy: 0.8167 - loss: 0.0992 - val_accuracy: 0.9815 - val_loss: 0.0103
Epoch 2/5
938/938            31s 33ms/step -
accuracy: 0.9837 - loss: 0.0110 - val_accuracy: 0.9893 - val_loss: 0.0067
Epoch 3/5
938/938            31s 33ms/step -
accuracy: 0.9885 - loss: 0.0076 - val_accuracy: 0.9889 - val_loss: 0.0070
Epoch 4/5
938/938            30s 32ms/step -
accuracy: 0.9920 - loss: 0.0058 - val_accuracy: 0.9900 - val_loss: 0.0066
Epoch 5/5
938/938            41s 32ms/step -
accuracy: 0.9937 - loss: 0.0042 - val_accuracy: 0.9912 - val_loss: 0.0057


Accuracy for:  RMSprop


Epoch 1/5
938/938            31s 32ms/step -
accuracy: 0.7934 - loss: 0.1095 - val_accuracy: 0.9810 - val_loss: 0.0106
Epoch 2/5
938/938            31s 33ms/step -
accuracy: 0.9780 - loss: 0.0140 - val_accuracy: 0.9865 - val_loss: 0.0077
Epoch 3/5
938/938            31s 33ms/step -
accuracy: 0.9868 - loss: 0.0089 - val_accuracy: 0.9900 - val_loss: 0.0061
Epoch 4/5
938/938            42s 34ms/step -
accuracy: 0.9911 - loss: 0.0061 - val_accuracy: 0.9915 - val_loss: 0.0053
Epoch 5/5
938/938            40s 32ms/step -
accuracy: 0.9939 - loss: 0.0044 - val_accuracy: 0.9900 - val_loss: 0.0063


Accuracy for:  Adadelta


Epoch 1/5
938/938            33s 34ms/step -
accuracy: 0.1001 - loss: 0.6806 - val_accuracy: 0.1069 - val_loss: 0.6311
Epoch 2/5
938/938            31s 33ms/step -
accuracy: 0.1005 - loss: 0.5991 - val_accuracy: 0.1278 - val_loss: 0.4308
Epoch 3/5
938/938            41s 33ms/step -
accuracy: 0.1077 - loss: 0.4143 - val_accuracy: 0.1536 - val_loss: 0.3345
Epoch 4/5
```

```
938/938             42s 34ms/step -
accuracy: 0.1176 - loss: 0.3604 - val_accuracy: 0.2520 - val_loss: 0.3248
Epoch 5/5
938/938             42s 35ms/step -
accuracy: 0.1237 - loss: 0.3522 - val_accuracy: 0.3556 - val_loss: 0.3196


Accuracy for:  Adagrad


Epoch 1/5
938/938             32s 33ms/step -
accuracy: 0.0961 - loss: 0.6229 - val_accuracy: 0.1029 - val_loss: 0.3300
Epoch 2/5
938/938             31s 33ms/step -
accuracy: 0.1181 - loss: 0.3461 - val_accuracy: 0.4434 - val_loss: 0.3171
Epoch 3/5
938/938             30s 31ms/step -
accuracy: 0.1546 - loss: 0.3326 - val_accuracy: 0.6627 - val_loss: 0.3044
Epoch 4/5
938/938             29s 31ms/step -
accuracy: 0.2225 - loss: 0.3183 - val_accuracy: 0.6934 - val_loss: 0.2851
Epoch 5/5
938/938             40s 30ms/step -
accuracy: 0.3294 - loss: 0.2986 - val_accuracy: 0.7023 - val_loss: 0.2536


Accuracy for:  SGD


Epoch 1/5
938/938             28s 29ms/step -
accuracy: 0.1162 - loss: 0.5056 - val_accuracy: 0.6243 - val_loss: 0.2982
Epoch 2/5
938/938             42s 30ms/step -
accuracy: 0.3258 - loss: 0.2965 - val_accuracy: 0.7799 - val_loss: 0.1752
Epoch 3/5
938/938             41s 30ms/step -
accuracy: 0.6496 - loss: 0.1944 - val_accuracy: 0.8570 - val_loss: 0.1079
Epoch 4/5
938/938             28s 30ms/step -
accuracy: 0.7635 - loss: 0.1414 - val_accuracy: 0.8869 - val_loss: 0.0815
Epoch 5/5
938/938             30s 32ms/step -
accuracy: 0.8178 - loss: 0.1143 - val_accuracy: 0.9006 - val_loss: 0.0687
```

0.0.5 PLOTTING OPTIMIZERS ACCURACY

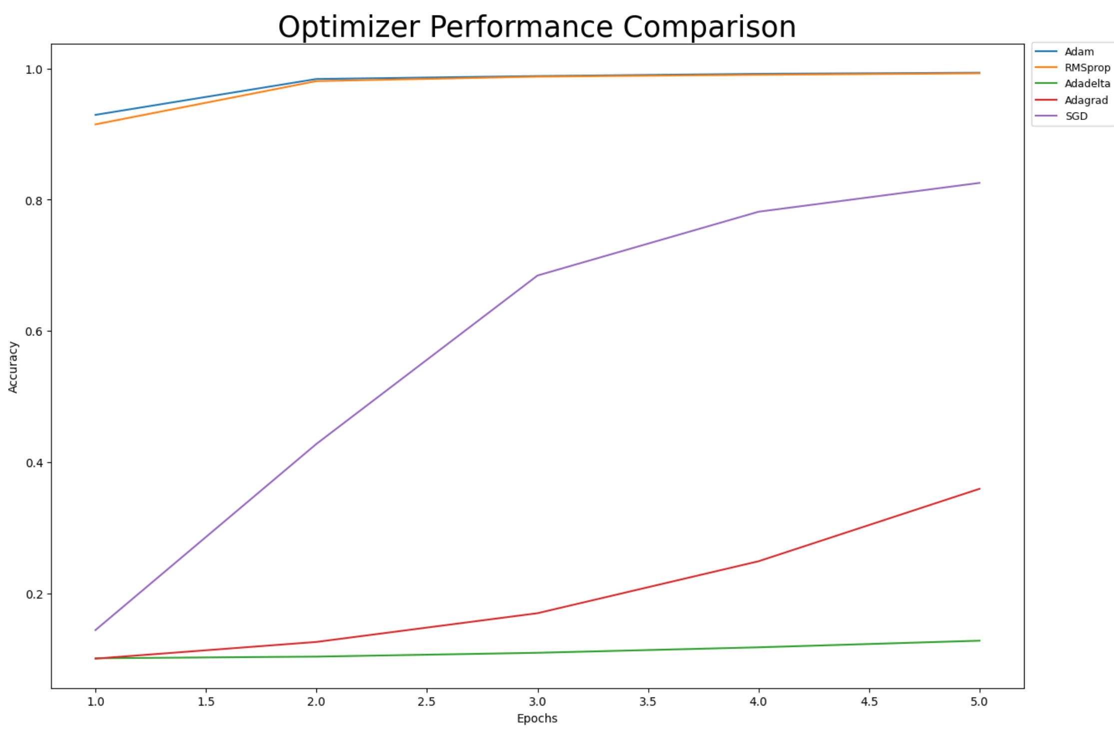
[8]:
```
import matplotlib.pyplot as plt

fully_nested = [list(zip(*[(ix+1,y) for ix,y in enumerate(x)])) for x in␣
 ↪opt_res]
names = ['sublist%d'%(i+1) for i in range(len(fully_nested))]

fig = plt.figure(figsize=(15,10))

for l in fully_nested:
    plt.plot(*l)

plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend(optimizers, fontsize=9, loc = 'upper right', bbox_to_anchor=(1.1, 1.
 ↪01))
plt.title("Optimizer Performance Comparison", fontsize=25)
plt.show()
```



Adam and RMSprop performed the best in terms of accuracy and loss reduction, with Adam being slightly better. Adadelta performed poorly, while Adagrad and SGD showed improvement over epochs but didn't reach the same level of performance as Adam and RMSprop within the given epochs.