AMATH482 HOMEWORK 5, NEURAL NETWORKS

CIHAN MERTOZ

Abstract. In this report we are building two different neural networks to analyze Fashion-MNIST data set. Then we will share the accuracy of our algorithms to predict the classes.

1. Introduction and Overview

In the first part of the assignment we are building a fully connected neural network. For both parts we are using an open-source machine learning tool called TensorFlow. Additionally to build the neural networks we are using Keras which is also an open-source library. Unlike the last four assignments, we are using Python instead of MATLAB. In the data set there are 10 different clothing classes with 60,000 images. Before we do the analysis, we are splitting the dataset into 55,000 training images and 5,000 validation images. Additionally, we have 10,000 images for testing. Each image is size $(28x28)$.

2. Theoretical Background

2.1. **Fully Connected Neural Nets.** The most important mathematical tool we are using in this assignment is called a neural network. It is basically a network that takes input data and analyzes with additional layers and the produces an output. Each layer has nodes and the nodes are connected to each other. Depending on the network, each connection has a an associated weight and a bias. We can fit these biases and weight into a vector using Linear Algebra and put all the weights into a matrix A. Biases are basically are inherent properties associated with each neuron and we can represent them as b. Putting all these in to a familiar equation:

$$Ax + b \tag{1}$$

Date: March 2020.

2.2. **Convolutional Neural Network.** Unlike fully connected networks, CNN is not fully connected. This is good in some cases because prevents over-fitting of the data which causes predictions to be comparatively worse. Basically, CNN neurons are layer on top of each other. Each layer recognizes a more complex feature about the data.

3. Algorithm Development

In this assignment we have two different parts. They are almost exactly the same except for the model used to train the data. Before we train the model, we prepared our data. After importing the necessary libraries in Python, we uploaded the data to Jupyter Notebook. After that we normalized our data. To train our model we used the Sequential model from Keras. Before adding layers, we flattened our data using the Flatten() command. For the fully connected network we added 4 layers using the dense command whereas for CNN we added one convolutional layer using Conv2D command and then we added two more layers. After that we fitted our models to our data and calculated the accuracy rates.
Before we decided on our final model we tried many different things. Firstly, we adjusted the number of layers in each case. For CNN, I added another convolutional layer however, there wasn't a big change in accuracy therefore we got rid of it. We also played with the optimizer, setting it to 0.0001 and 0.1. Our current value did better than both of those cases. Therefore we decided on keeping it at 0.001.

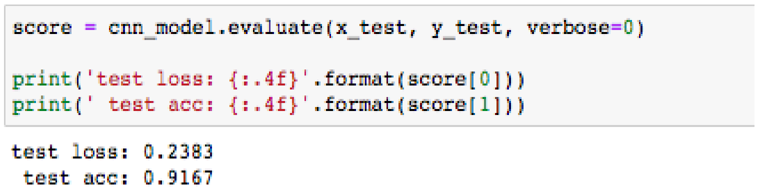
4. Computational results

4.1. **Full.** In fully connected neural network, we got test accuracy 89.41 percent.

4.2. **CNN.** In convolutional network, we got test accuracy 91.67 percent.

5. Conclusion

After training the same data with the two different methods, we saw that CNN is doing better than Fully Connected Neural Networks. However, the time it takes to train a CNN is almost 10 times more than Fully Connected Neural Networks. So if we want faster results with decent accuracy Full Connected Networks seem to be the better option

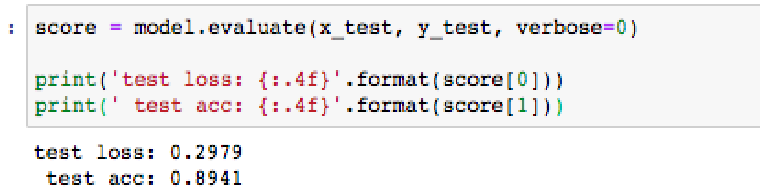


```
score = cnn_model.evaluate(x_test, y_test, verbose=0)

print('test loss: {:.4f}'.format(score[0]))
print(' test acc: {:.4f}'.format(score[1]))
```

```
test loss: 0.2383
 test acc: 0.9167
```

FIGURE 1. CNN,Results

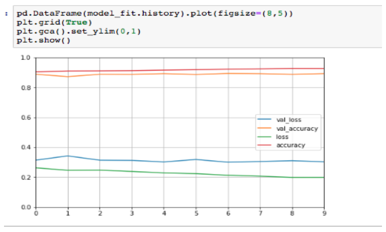


```
score = model.evaluate(x_test, y_test, verbose=0)

print('test loss: {:.4f}'.format(score[0]))
print(' test acc: {:.4f}'.format(score[1]))
```

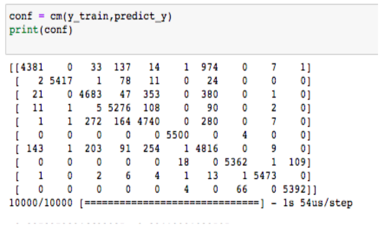
```
test loss: 0.2979
 test acc: 0.8941
```

FIGURE 2. Fully Connected, Results



```
pd.DataFrame(model_fit.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```

FIGURE 3. Fully Connected,Plot



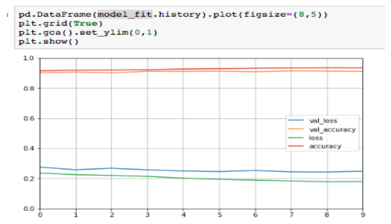
```
conf = cm(y_train,predict_y)
print(conf)
```

```
[[4381    0   33  137   14    1  974    0    7    1]
 [   2 5417    1   78   11    0   24    0    0    0]
 [  21    0 4683   47  353    0  380    0    1    0]
 [  11    1    5 5276  108    0   90    0    2    0]
 [   1    1  272  164 4740    0  280    0    7    0]
 [   0    0    0    0    0 5500    0    4    0    0]
 [ 143    1  203   91  254    1 4816    0    9    0]
 [   0    0    0    0    0   18    0 5362    1  109]
 [   1    0    2    6    4    1   13    1 5473    0]
 [   0    0    0    0    0    4    0   66    0 5392]]
10000/10000 [==============================] - 1s 54us/step
```

FIGURE 4. Fully Connected,Confusion Matrix

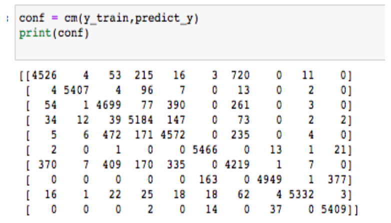
6. Appendix A

tf.keras.layers.Sequential = Creating Layers
tf.keras.layers.Dense = Creating connected Neural Networks
tf.keras.layers.Conv2D = 2D Convolution Layer



```
pd.DataFrame(model_fit.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```

Figure 5. CNN,Plot



```
conf = cm(y_train,predict_y)
print(conf)
```

```
[[4526    4   53  215   16    3  720    0   11    0]
 [   4 5407    4   96    7    0   13    0    2    0]
 [  54    1 4699   77  390    0  261    0    3    0]
 [  34   12   39 5184  147    0   73    0    2    2]
 [   5    6  472  171 4572    0  235    0    4    0]
 [   2    0    1    0    0 5466    0   13    1   21]
 [ 370    7  409  170  335    0 4219    1    7    0]
 [   0    0    0    0    0  163    0 4949    1  377]
 [  16    1   22   25   18   18   62    4 5332    3]
 [   0    0    0    2    0   14    0   37    0 5409]]
```

Figure 6. CNN,Confusion Matrix

7. Appendix B

```
# Part 1
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropou
from keras.optimizers import Adam
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix as cm
train_df = pd.read_csv('~/Downloads/fashionmnist/fashion-mnist_train.
test_df =  pd.read_csv('~/Downloads/fashionmnist/fashion-mnist_test.c
train_data = np.array(train_df, dtype='float32')
test_data = np.array(test_df, dtype='float32')

x_train = train_data[:, 1:] / 255
y_train = train_data[:, 0]

x_test = test_data[:, 1:] / 255
y_test = test_data[:, 0]
```

```
x_train, x_validate, y_train, y_validate = train_test_split(
    x_train, y_train, test_size=0.08335, random_state=12345,
)
im_rows = 28
im_cols = 28
batch_size = 512
im_shape = (im_rows, im_cols, 1)

x_train = x_train.reshape(x_train.shape[0], *im_shape)
x_test = x_test.reshape(x_test.shape[0], *im_shape)
x_validate = x_validate.reshape(x_validate.shape[0], *im_shape)

print('x_train shape: {}'.format(x_train.shape))
print('x_test shape: {}'.format(x_test.shape))
print('x_validate shape: {}'.format(x_validate.shape))
x_train shape: (54999, 28, 28, 1)
x_test shape: (10000, 28, 28, 1)
x_validate shape: (5001, 28, 28, 1)
model = Sequential([
    Flatten(),
    Dense(400, activation='relu'),
    Dense(50, activation='relu'),
    Dense(20, activation='relu'),
    Dense(10, activation='softmax')
])
model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=Adam(lr=0.001),
    metrics=['accuracy']
)
model_fit = model.fit(
    x_train, y_train, batch_size=batch_size,
    epochs=10, verbose=1,
    validation_data=(x_validate, y_validate),
)
Train on 54999 samples, validate on 5001 samples
Epoch 1/10
54999/54999 [==============================] - 2s 44us/step - loss: 0
Epoch 2/10
```

```
54999/54999 [==============================] - 2s 44us/step - loss: 0
Epoch 3/10
54999/54999 [==============================] - 3s 46us/step - loss: 0
Epoch 4/10
54999/54999 [==============================] - 2s 44us/step - loss: 0
Epoch 5/10
54999/54999 [==============================] - 2s 45us/step - loss: 0
Epoch 6/10
54999/54999 [==============================] - 2s 43us/step - loss: 0
Epoch 7/10
54999/54999 [==============================] - 2s 44us/step - loss: 0
Epoch 8/10
54999/54999 [==============================] - 3s 49us/step - loss: 0
Epoch 9/10
54999/54999 [==============================] - 2s 44us/step - loss: 0
Epoch 10/10
54999/54999 [==============================] - 3s 47us/step - loss: 0
score = model.evaluate(x_test, y_test, verbose=0)

print('test loss: {:.4f}'.format(score[0]))
print(' test acc: {:.4f}'.format(score[1]))
test loss: 0.2979
 test acc: 0.8941
pd.DataFrame(model_fit.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()

predict_y = model.predict_classes(x_train)
conf = cm(y_train,predict_y)
print(conf)
[[4381    0   33  137   14    1  974    0    7    1]
 [   2 5417    1   78   11    0   24    0    0    0]
 [  21    0 4683   47  353    0  380    0    1    0]
 [  11    1    5 5276  108    0   90    0    2    0]
 [   1    1  272  164 4740    0  280    0    7    0]
 [   0    0    0    0    0 5500    0    4    0    0]
 [ 143    1  203   91  254    1 4816    0    9    0]
 [   0    0    0    0    0   18    0 5362    1  109]
 [    1    0    2    6    4    1   13    1 5473    0]
 [    0    0    0    0    0    4    0   66    0 5392]]
10000/10000 [==============================] - 1s
```