Training Neural Networks to Categorize Clothing

Christina Smith

AMATH 482 - Winter 2020

.

Abstract

The objective of this exercise is to explore training both fully-connected and convolutional neural networks using the Fashion-MNIST data set. The testing accuracy is compared using a variety of hyperparameters.

I. Introduction

The Fashion-MNIST dataset contains images with 10 classifications. Top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle Boot. We will train a fully connected neural network and a convolutional neural network to predict classifications of new images. In order to do this, we will adjust hyperparameters of the models, including depth and width of the network, learning rate, activation functions and number of filters. We will evaluate the accuracy and the loss values that result from the models and look for information about errors using a confusion matrix.

II. Theoretical Background

In its most simple sense, we consider a neural network to consist of a vector of input node(s) $\mathbf{x}$ connected to a output node y by a vector of weighted edges $\mathbf{w}$. The value of these input nodes and their weighted edges is represented as

$$z = \mathbf{w}^T\mathbf{x} \tag{1}$$

The output node itself has a binary value, which is determined by a step function, our activation function σ. Note that in order to accommodate for thresholds that may not be at zero, we add another node, called the bias node. This node has a value of 1 and a weight of b to account for different thresholds. So the formula for the output node is as follows:

$$y = \sigma(z) = \sigma(\mathbf{w}^T\mathbf{x} + b) = \begin{cases} 0 & z < \text{threshold} \\ 1 & z \geq \text{threshold} \end{cases} \tag{2}$$

Building on this model, we have a perceptron, which is like our original network only now there are multiple inputs and the bias node and each node in $\mathbf{x}$ has a separate weight value associated with each output node. The formula for computing the output values of a perceptron is as follows:

$$\mathbf{y} = \sigma(\mathbf{A}\mathbf{x} + \mathbf{b}) \tag{3}$$

where $\mathbf{A}$ is a matrix of all of the $\mathbf{w}$ vectors.

There is also a multi-layered perceptron that uses the formula recursively to compute the final output from multiple input vectors.

We consider the original input $\mathbf{x}$ to be a set value and the output values that follow from that are determined by the weighted edges. We use a loss function to determine the accuracy of our output from the model versus the correct output. In order to do minimize this loss function, we want to use gradient descent. This requires that we are able to take a derivative of our activation functions. Since a step function is discontinuous, we must choose a different function. The activation function we will use in this exercise is the Rectified Linear Unit (ReLU) function which is the max of 0 or x.

We also want to evaluate the errors in our predictions using a cross entropy loss function:

$$L = -\frac{1}{N}\sum_{j=1}^{N} \ln(p_j) \tag{4}$$

For this function, we add a regularization factor $\lambda||\mathbf{w}||_2^2$, which ensures that the weights will not get too large (prevents overfitting).

In order to minimize our loss function we will use gradient descent in order to find the minimum values associated. The learning rate we choose affects how we traverse this domain of this function and a larger learning rate may result in less accuracy as then the gradient descent can overshoot the minimum.

III. Algorithm Implementation and Development

We import the data sets of images and labels and break the data into training sets, test sets, and validation sets and then convert the values to represent grey scale values.

For Part 1, we create a base layer which uses the ReLU function and a regularizer with λ set to 10^{-4}. We create a model with 10 layers that are 500 nodes wide, and a final layer that is 10 nodes wide. We then compile the model with a sparse crossentropy loss function, an Adam optimizer and a learning rate of 10^{-4}.

We then fit the model with 10 epochs.

For Part 2, we create convolutional layers with filters and kernels of 10 each and with padding set to 'valid'. We also set the pooling to average pooling with a size of 2.

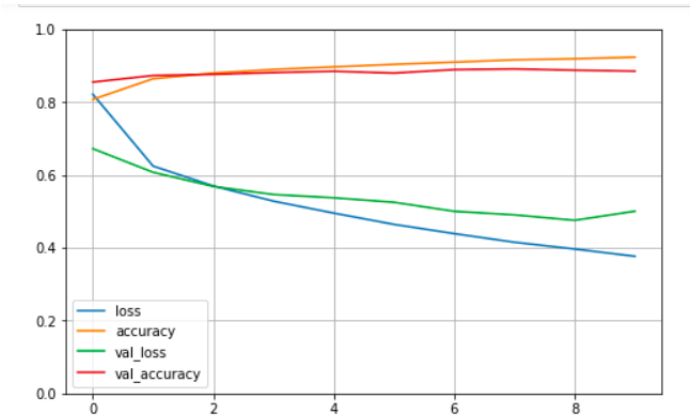


Figure 1: Fully Connected Network - Loss and Accuracy Graph

```
[[5075    3   18   28    3    3  408    0    5    0]
 [  22 5344    4   66    3    4    1    0    0    0]
 [ 101    0 4705   38  481    0  169    0    2    0]
 [  98    0    7 5284   77    1   30    0    1    1]
 [   4    0  255  188 4931    0  130    0    4    0]
 [   0    0    0    0    0 5501    0    2    0    4]
 [ 512    0  330   78  382    1 4195    0    8    1]
 [   0    0    0    0    0  414    0 4983    0   91]
 [  20    1    0   18    1   12    6    1 5450    1]
 [   0    0    0    0    0   11    0   87    0 5396]]
```

Figure 2: Fully Connected Network - Confusion Matrix

IV.Computational Results & Supplementary Plots

Part 1

See Figures 1 and 2.

Part 2

See Figures 3 and 4.

V.Summary and Conclusions

In the fully connected network, it appears the largest confusion happens between the T-shirt/top class and the shirt class. There was also notable confusion between the Pullover, Coat, and Shirt classes

In part 1, the model was able to achieve an accuracy of 88%.

The best results occurred when the scalar factor of the regularizer value was 10^{-4} and increasing the value to 10^{-2} decreased the validation accuracy by 10 %. (from about 87 % to 77 %.

Decreasing the learning rate from 10^{-4} to 10^{-5} decreased the validation accuracy by about 3 %.

In convolutional network, the models was able to achieve an accuracy of 98 %

The confusion matrix for this model shows that there were less errors overall, but the Dress class showed a lot of confusion with other classes.

Appendix A: Python Functions

tf.keras.Sequential() - used to create a model for a neural network. Takes in customizable layers.

tf.keras.layers.Dense() - customizable dense layers

tf.keras.layers.Conv2D() - customizable convolutional layers

compile() - set the loss function, optimizer and metrics

fit() - takes in training data, validation data and number of epochs to train the model.

Appendix A: Python Code

HW_05_p1_a

March 13, 2020

[61]:
```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import confusion_matrix

fashion_mnist = tf.keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

[62]:
```
# convert the data to doubles ranging from 0 to 1,
# and break into training and validation sets
X_valid = X_train_full[:5000]/255.0
X_train = X_train_full[5000:]/255.0
X_test = X_test/255.0

y_valid = y_train_full[:5000]
y_train = y_train_full[5000:]
```

[63]:
```
my_dense_layer = partial(tf.keras.layers.Dense, activation="relu", \
                         kernel_regularizer=tf.keras.regularizers.l2(0.0001))
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    # fully connected hidden layer
    my_dense_layer(700),
    my_dense_layer(700),
    my_dense_layer(700),
    my_dense_layer(700),
    my_dense_layer(700),
    my_dense_layer(700),
    my_dense_layer(700),
    my_dense_layer(700),
    my_dense_layer(700),
    my_dense_layer(700),
    # has 10 'neurons' because we have 10 classes
    # the softmax function gives us 10 probabilities
    tf.keras.layers.Dense(10,activation="softmax")
])
```

[64]:
```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              metrics=["accuracy"])
```

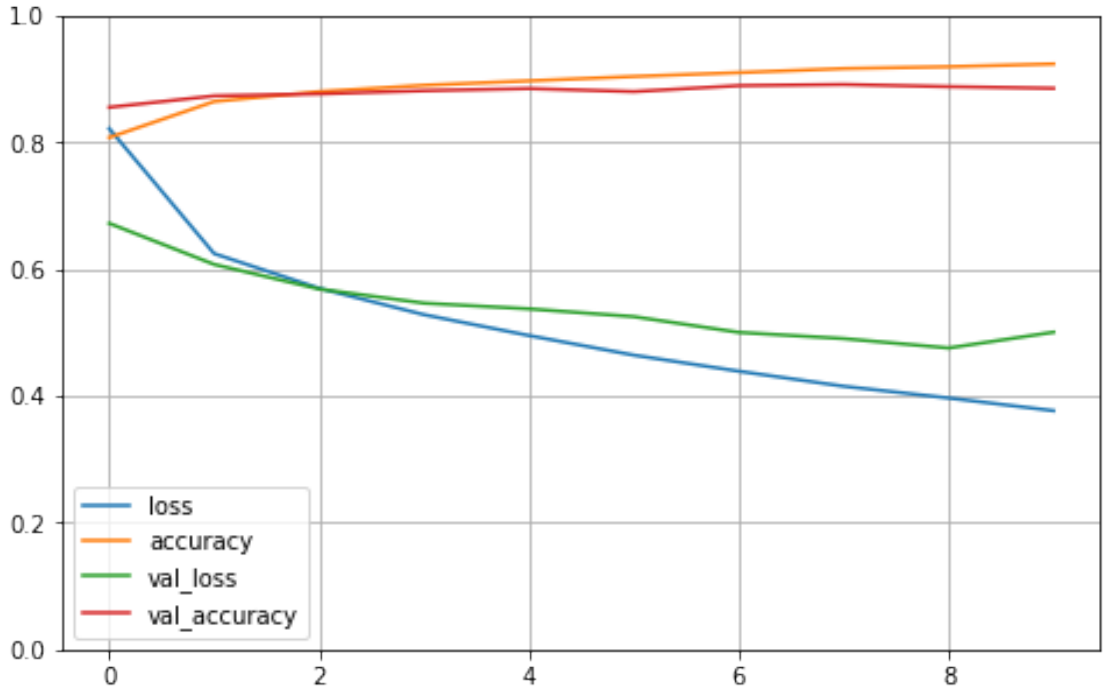
[65]:
```
# training the nn
history = model.fit(X_train, y_train, epochs=10,␣
 ↪validation_data=(X_valid,y_valid))
```

```
Train on 55000 samples, validate on 5000 samples
Epoch 1/10
55000/55000 [==============================] - 411s 7ms/sample - loss: 1.0295 -
accuracy: 0.8032 - val_loss: 0.8142 - val_accuracy: 0.8680
Epoch 2/10
55000/55000 [==============================] - 389s 7ms/sample - loss: 0.7938 -
accuracy: 0.8626 - val_loss: 0.8082 - val_accuracy: 0.8592
Epoch 3/10
55000/55000 [==============================] - 404s 7ms/sample - loss: 0.7140 -
accuracy: 0.8796 - val_loss: 0.7129 - val_accuracy: 0.8770
Epoch 4/10
55000/55000 [==============================] - 336s 6ms/sample - loss: 0.6529 -
accuracy: 0.8890 - val_loss: 0.6660 - val_accuracy: 0.8804
Epoch 5/10
55000/55000 [==============================] - 284s 5ms/sample - loss: 0.6042 -
accuracy: 0.8956 - val_loss: 0.6379 - val_accuracy: 0.8784
Epoch 6/10
55000/55000 [==============================] - 286s 5ms/sample - loss: 0.5566 -
accuracy: 0.9012 - val_loss: 0.5895 - val_accuracy: 0.8860
Epoch 7/10
55000/55000 [==============================] - 284s 5ms/sample - loss: 0.5147 -
accuracy: 0.9083 - val_loss: 0.5538 - val_accuracy: 0.8958
Epoch 8/10
55000/55000 [==============================] - 284s 5ms/sample - loss: 0.4775 -
accuracy: 0.9132 - val_loss: 0.5372 - val_accuracy: 0.8916
Epoch 9/10
55000/55000 [==============================] - 280s 5ms/sample - loss: 0.4469 -
accuracy: 0.9170 - val_loss: 0.5244 - val_accuracy: 0.8956
Epoch 10/10
55000/55000 [==============================] - 285s 5ms/sample - loss: 0.4215 -
accuracy: 0.9223 - val_loss: 0.5325 - val_accuracy: 0.8848
```

[40]:
```
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()

# plotting the confusion matrix
y_pred = model.predict_classes(X_train)
```

```
conf_train = confusion_matrix(y_train, y_pred)
print(conf_train)
```




```
[[5075    3   18   28    3    3  408    0    5    0]
 [  22 5344    4   66    3    4    1    0    0    0]
 [ 101    0 4705   38  481    0  169    0    2    0]
 [  98    0    7 5284   77    1   30    0    1    1]
 [   4    0  255  188 4931    0  130    0    4    0]
 [   0    0    0    0    0 5501    0    2    0    4]
 [ 512    0  330   78  382    1 4195    0    8    1]
 [   0    0    0    0    0  414    0 4983    0   91]
 [  20    1    0   18    1   12    6    1 5450    1]
 [   0    0    0    0    0   11    0   87    0 5396]]
```

[22]:
```
model.evaluate(X_test, y_test)
```

```
10000/1 [==============================================================================
==============================================================================
==============================================================================
==============================================================================
==============================================================================
==============================================================================
==============================================================================
==============================================================================
==============================================================================
==============================================================================
```

```
==============================================================================
==============================================================================
==============================================================================
==============================================================================
==============================================================================
==============================================================================
==============================================================================
==============================================================================
==============================================================================
==============================================================================
==============================================================================
==============================================================================
==========] - 1s 85us/sample - loss: 0.3969 - accuracy: 0.8299
```

[21]:
```
[0.4976480528831482, 0.8299]
```

[ ]:

[ ]:

[ ]:

HW_05_p2_a

March 13, 2020

[87]:
```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import confusion_matrix
```

[88]:
```
fashion_mnist = tf.keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

[89]:
```
X_valid = X_train_full[:5000] / 255.0
X_train = X_train_full[5000:] / 255.0
X_test = X_test / 255.0

y_valid = y_train_full[:5000]
y_train = y_train_full[5000:]

#55000x28x28x1 (because it's greyscale, if RGB would be x3)

X_train = X_train[..., np.newaxis]
X_valid = X_valid[..., np.newaxis]
X_test = X_test[..., np.newaxis]
```

[90]:
```
from functools import partial

my_dense_layer = partial(tf.keras.layers.Dense, activation="relu", \
                         kernel_regularizer=tf.keras.regularizers.l2(0.0001))
my_conv_layer = partial(tf.keras.layers.Conv2D, activation="relu",␣
 ↪padding="valid")

model = tf.keras.models.Sequential([
    # we don't have to flatten the images for conv layers
    my_conv_layer(5,10,padding="same",input_shape=[28,28,1]),
    my_conv_layer(5,10,padding="same",input_shape=[28,28,1]),
    my_conv_layer(5,10,padding="same",input_shape=[28,28,1]),
    tf.keras.layers.AveragePooling2D(2),
    my_conv_layer(16,5),
    tf.keras.layers.AveragePooling2D(2),
```

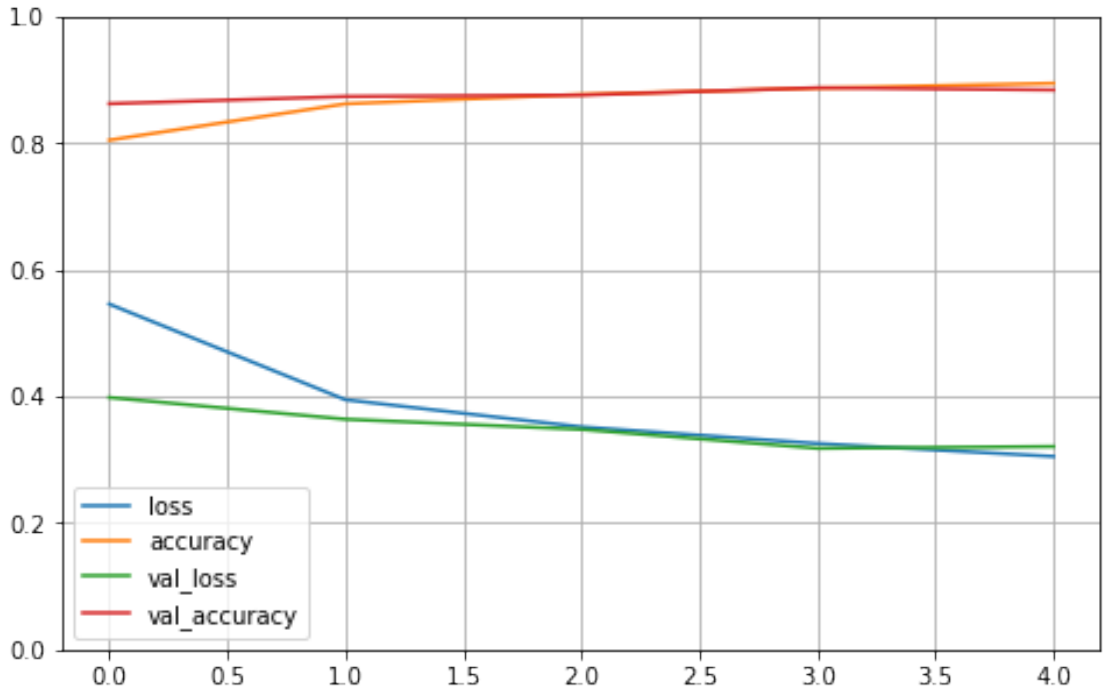
```
    # flatten before reverting to dense layers
    tf.keras.layers.Flatten(),
    my_dense_layer(84),
    my_dense_layer(10, activation="softmax")
])
```

[91]:
```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              metrics=["accuracy"])
```

[92]:
```
history = model.fit(X_train, y_train, epochs=5,␣
↪validation_data=(X_valid,y_valid))
```

```
Train on 55000 samples, validate on 5000 samples
Epoch 1/5
55000/55000 [==============================] - 2495s 45ms/sample - loss: 0.8188
- accuracy: 0.7010 - val_loss: 0.5979 - val_accuracy: 0.7866
Epoch 2/5
55000/55000 [==============================] - 604s 11ms/sample - loss: 0.5819 -
accuracy: 0.7885 - val_loss: 0.5240 - val_accuracy: 0.8148
Epoch 3/5
55000/55000 [==============================] - 326s 6ms/sample - loss: 0.5189 -
accuracy: 0.8132 - val_loss: 0.4996 - val_accuracy: 0.8238
Epoch 4/5
55000/55000 [==============================] - 325s 6ms/sample - loss: 0.4823 -
accuracy: 0.8278 - val_loss: 0.4687 - val_accuracy: 0.8332
Epoch 5/5
55000/55000 [==============================] - 327s 6ms/sample - loss: 0.4573 -
accuracy: 0.8373 - val_loss: 0.4613 - val_accuracy: 0.8380
```

[7]:
```
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```



[10]:
```
y_pred = model.predict_classes(X_train)
conf_train = confusion_matrix(y_train, y_pred)
print(conf_train)
```

```
[[5402    0    3    6    0    6   13    1   10    3]
 [   0 6153    3    9    0    0    0   10    3    1]
 [   1    4 5438   15    1    0    0    9    1    1]
 [   0    0    7 5612    0    5    0    3    5    6]
 [   0    5    2    0 5253    0    7    1    2   37]
 [   1    1    0   46    1 4905   26    0    4    3]
 [   1    6    1    0    1    1 5404    0    3    0]
 [   0    7    5   15   14    1    0 5656    3   14]
 [   1   13    6   33    5    4    4    1 5308   14]
 [   5    3    1   19    9    7    1    9    2 5398]]
```

[11]:
```
model.evaluate(X_test,y_test)
```

```
10000/10000 [==============================] - 2s 221us/sample - loss: 0.0699 -
accuracy: 0.9833
```

[11]:
```
[0.06994416005015373, 0.9833]
```

[8]:
```
y_pred = model.predict_classes(X_test)
conf_test = confusion_matrix(y_test, y_pred)
print(conf_test)
```

```
[[ 972    0    1    1    0    1    1    0    2    2]
 [   0 1133    1    1    0    0    0    0    0    0]
 [   1    3 1019    3    1    0    0    3    1    1]
 [   0    0    1 1005    0    0    0    1    0    3]
 [   0    0    4    1  964    0    2    2    0    9]
 [   1    0    0   23    1  863    1    1    0    2]
 [   8    3    1    1    4    4  935    0    2    0]
 [   1    4    8    4    0    0    0 1002    1    8]
 [   1    0    4    5    4    1    1    2  951    5]
 [   1    3    0    3   11    1    0    3    0  987]]
```

[9]:
```
fig, ax = plt.subplots()

# hide axes
fig.patch.set_visible(False)
ax.axis('off')
ax.axis('tight')

# create table and save to file
df = pd.DataFrame(conf_test)
ax.table(cellText=df.values, rowLabels=np.arange(10), colLabels=np.arange(10),␣
 ↪loc='center', cellLoc='center')
fig.tight_layout()
plt.savefig('conf_mat.pdf')
```

	0	1	2	3	4	5	6	7	8	9
0	972	0	1	1	0	1	1	0	2	2
1	0	1133	1	1	0	0	0	0	0	0
2	1	3	1019	3	1	0	0	3	1	1
3	0	0	1	1005	0	0	0	1	0	3
4	0	0	4	1	964	0	2	2	0	9
5	1	0	0	23	1	863	1	1	0	2
6	8	3	1	1	4	4	935	0	2	0
7	1	4	8	4	0	0	0	1002	1	8
8	1	0	4	5	4	1	1	2	951	5
9	1	3	0	3	11	1	0	3	0	987

[ ]:

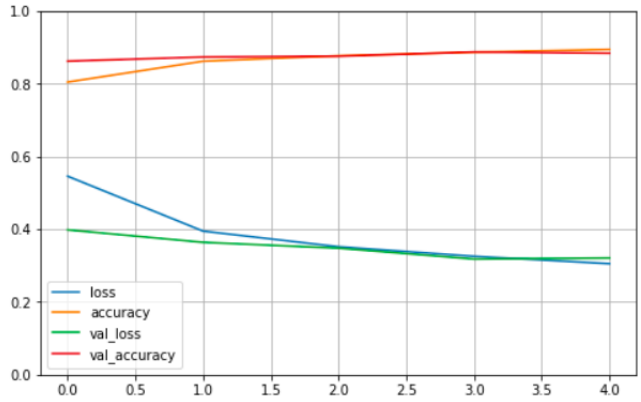


Figure 3: Convolutional Network - Loss and Accuracy Graph

```
[[5402    0    3    6    0    6   13    1   10    3]
 [   0 6153    3    9    0    0    0   10    3    1]
 [   1    4 5438   15    1    0    0    9    1    1]
 [   0    0    7 5612    0    5    0    3    5    6]
 [   0    5    2    0 5253    0    7    1    2   37]
 [   1    1    0   46    1 4905   26    0    4    3]
 [   1    6    1    0    1    1 5404    0    3    0]
 [   0    7    5   15   14    1    0 5656    3   14]
 [   1   13    6   33    5    4    4    1 5308   14]
 [   5    3    1   19    9    7    1    9    2 5398]]
```

Figure 4: Convolutional Network - Confusion Matrix