Homework 5: Neural Networks

Maximilian Walter
AMATH 482
University of Washington
3/13/2020

This goal of this assignment is to compare the accuracy and efficacy of a fully connected versus a convolutional neural network. The networks are made in Python using the TensorFlow package and tested on the Fashion-MNST data set. The model is then assessed by looking at the validation and testing accuracy. For each model, many different hyperparameters were altered until a suitable model was found to compare. Overall, the CNN was more accurate, but both struggled with correctly classifying shirts, commonly confusing them with t-shirts and pullovers.

Introduction and Overview

This assignment is split into two parts, the first is to build a fully connected neural network using TensorFlow and the second was to build a convolutional neural network and compare the two. The data used to train these networks is the Fashion-MNST dataset found in TensorFlow. This is a collection of 60,000 images of clothing in 10 classes found below.

Label	**Description**
0	T-Shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle Boot

This is split into 55,000 training images and 5,000 validation images. Another set of 10,000 images are used for testing the trained model. For each model, I altered different hyperparameters as well as the number of layers and layer widths. I found that CNN was more accurate but required more computational power and time to complete each epoch.

Theoretical Background

Fully Connected Network

A neural network is an input layer followed by a series of hidden layers and finally an output layer. Each layer is connected by a series of weighted values. The layers themselves consist of nodes. What makes this network fully connected is that each node from one layer is connected to each node of the next layer. To find the value of these nodes, the input value is multiplied by the weight and then inputted into a non-linear function, known as an activation function.

$$N = f(x_n * w_n) \tag{1.1}$$

Additional terms are added to each layer called bias values. These are added into equation 1.1

$$N_n = f(\sum_{j=1}^{n}(x_j * w_j + b_j)) \tag{1.2}$$

There are many options for including non-linearity into this equation. For the assignment, I used the activation function, rectified linear unit (ReLu) for my hidden layers, and SoftMax for my output layer.

ReLu: A rough approximation of the function.

$$f(N) = \ln(1 + e^N) \tag{2.1}$$

Softmax

$$P(y = j|x) = \frac{e^{x^T * w_j}}{\sum_{k=1}^{K} e^{x^T * w_k}} \tag{2.2}$$

Convolutional Neural Network (CNN)

Similar to the fully connected network, CNN has many connected layers. What makes this model different is how it creates the input layer. The initial layer instead of taking the raw data as inputs creates tensors of a section of data. In simple terms, this looks at a small subset of an image, say a 4x4 square, instead of processing the entire image at once. This is then combined with pooling, which compresses the information found in the square and pushes it to the next node as a smaller set of data, in my case a 2x2 output. This new output is again convolved and pooled to create the input to the FCN. After this, the process is the same as in section one. Below is a visual representation from Medium.com1.

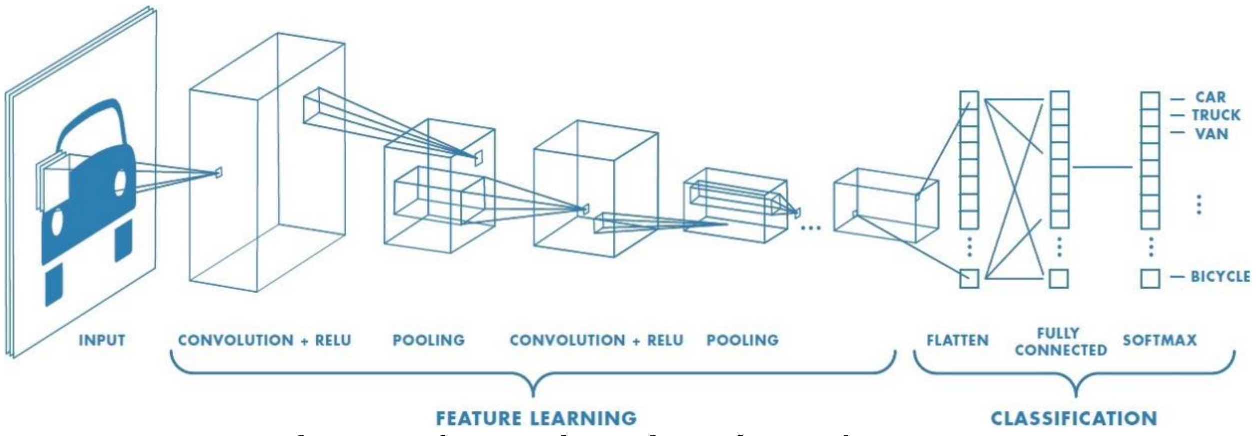


Figure 1. Visual process of a Convolutional neural network.

Algorithm Development

FCN:

The first part of this assignment was creating an FCN in Python. I used Jupyter notebooks to run the code which can be found on my Github. Most of the tools needed are in the TensorFlow package. Additional packages used were NumPy, pandas, matplotlib, and sklearn for data processing and plotting.

I began by loading the data. I am using the fashion-MNST dataset from TensorFlow. To better understand the data, I looked at the sizes and plotted the training and testing sets. I then separated the training data into 55,000 training images and 5,000 validation images. Because the data was in a uint8 format, I normalized it and converted it into a double by dividing by 255.0.

With my dataset prepared, I started building the model. Using the Keras tool, I began by flattening my data. For my dense layers, I used the ReLu activation function and the l2 regularizer. I set all my widths to 100 for my initial pass. My output layer used the softmax activation function and a width of 10. To compile my model, I used the sparse categorical cross-entropy loss parameter and the Adam optimizer with accuracy as my metric and an initial learning rate of 0.001. After I had the model set up, I trained it using the training and validation sets, testing with a baseline of 10 epochs. My initial results were low, with a validation accuracy of 83% and testing accuracy of 82%. I then began altering the following parameters.

- Network Depth
- Layer Width
- Activation Functions
- Learning Rate
- Optimizer
- Epochs

I found that as I decreased the learning rate, the model would produce a higher validation accuracy but would be overfitting and produce a low testing accuracy (~10%). The width of the layers improved both validation and testing accuracy. The depth provided marginal improvements to performance. Changing to the NAdam optimizer improved the model at times but also caused it to run much longer. I also included different models to randomize starting weights, trying the he_uniform initializer.

My final model had the following parameters.

- Input
  - Flatten: [28,28]
- Hidden Layers
  - Activation: ReLu
  - Regularizer: l2 (0.00001)
  - Three Layers
    - Width:300, 150, 10
- Output
  - Width:10
  - Activation: Softmax
- Compiler

    - Loss: Sparse Categorical Cross Entropy
    - Optimizer: Adam
        - Learning Rate: 0.0001
        - Metric: Accuracy
- Training
    - Epochs: 40

CNN

I did the same data preparation as in the FCN section with the only difference was adding a third dimension to the data so that each image became 28x28x1. This was done because the built-in tool requires this additional layer. The convolutional layers added a few more parameters to play around with.

- Filters
- Pooling Size
    - Size of the output
- Padding
    - This parameter adds 0's to the output to "pad" the ends.
- Kernel Size
    - Size of convolutional window
- Strides

I began by including two convolutional layers and 2 dense layers. My activation functions were initially tanh. I randomly assigned filter and kernel size and used the same padding so that the input resulted in the same size output. My pooling was set to 2. I found the validation accuracy to be better than the FCN but only marginally, resulting in ~88%. I then began playing around with the above parameters. Since I found some success with the ReLu activation function, I returned to using that for both my convolution and dense layers. I also included the HE initializer to both layer types. I reduced the number of dense layers to one hidden and one output. I added another convolutional layer increasing the filter by 2x each layer. My kernel size was set to (5,5) and I kept the original pooling of 2,2. I kept the activation function of my output layer as softmax. I found that this model required more time to compute, so I only used 20 epochs as I found that accuracy flattened around this value. My final model parameters are below

- Three Convolutional Layers
    - Activation: ReLu
    - Initializer: HE Uniform
    - Padding: Same for first, Valid for all others.
    - Pooling: Average (2,2)
    - Filter:
        - 32
        - 64
        - 128
    - Kernel Size: (5,5)
- One Dense Layer
    - Activation: ReLu

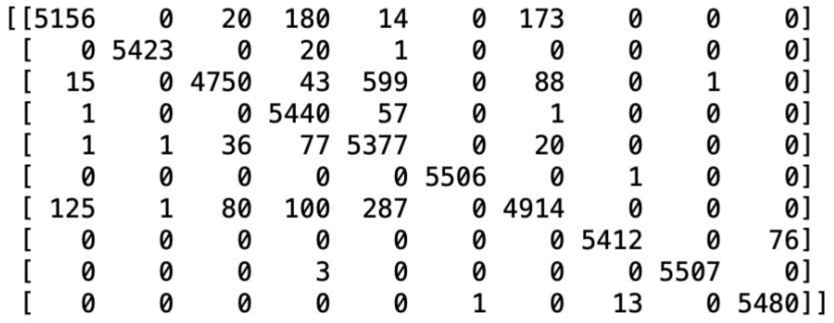
    - Initializer: HE Uniform
    - Regularizer: l2 (0.0001)
    - Width: 10
- Output
    - Activation: Softmax
    - Width: 10
- Compiler
    - Loss: Sparse Categorical Cross-Entropy
    - Optimizer: Adam
        - Learning Rate: 0.0001
        - Metric: Accuracy
- Training
    - Epochs: 2 0

Computational Results

FCN

These are the results using the parameters found under algorithm development.

Accuracy: 96.51% **Validation Accuracy:** 89.82% **Testing Accuracy:** 88.81%



```
[[5156    0   20  180   14    0  173    0    0    0]
 [   0 5423    0   20    1    0    0    0    0    0]
 [  15    0 4750   43  599    0   88    0    1    0]
 [   1    0    0 5440   57    0    1    0    0    0]
 [   1    1   36   77 5377    0   20    0    0    0]
 [   0    0    0    0    0 5506    0    1    0    0]
 [ 125    1   80  100  287    0 4914    0    0    0]
 [   0    0    0    0    0    0    0 5412    0   76]
 [   0    0    0    3    0    0    0    0 5507    0]
 [   0    0    0    0    0    1    0   13    0 5480]]
```

Figure 2. Confusion Matrix of FCN. It had the highest error classifying dresses and coats.

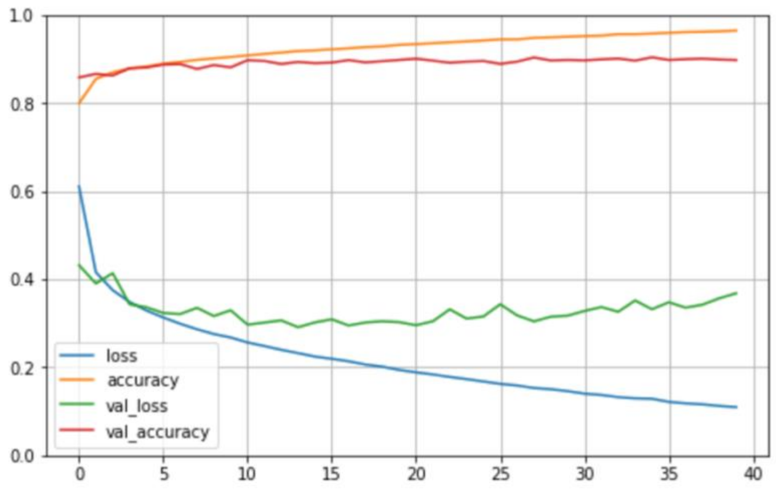


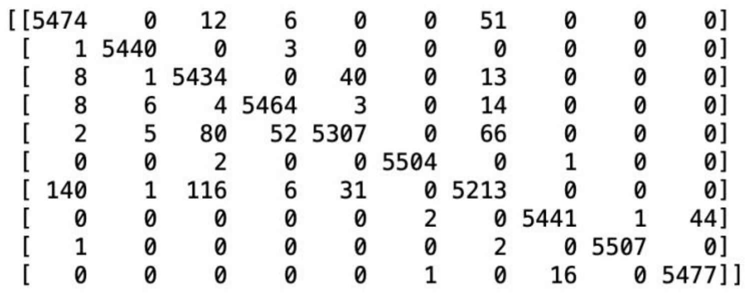
Figure 3. Plot of the model training. Validation loss separates and flattens after 5 epochs.

The model did well but struggled with classifying dresses and coats. Mostly predicting a coat as a pullover and dresses as t-shirts or pullovers. These are understandable as these pieces of clothing have a similar structure. It was interesting to see the validation loss flatten heavily after 5 epochs and separate from the model loss so drastically. The validation accuracy followed a similar trend but stayed close to the model accuracy.

CNN

These are the results using the parameters found under algorithm development.

Accuracy: 98.00% **Validation Accuracy:** 92.78% **Testing Accuracy:** 91.90%



```
[[5474    0   12    6    0    0   51    0    0    0]
 [   1 5440    0    3    0    0    0    0    0    0]
 [   8    1 5434    0   40    0   13    0    0    0]
 [   8    6    4 5464    3    0   14    0    0    0]
 [   2    5   80   52 5307    0   66    0    0    0]
 [   0    0    2    0    0 5504    0    1    0    0]
 [ 140    1  116    6   31    0 5213    0    0    0]
 [   0    0    0    0    0    2    0 5441    1   44]
 [   1    0    0    0    0    0    2    0 5507    0]
 [   0    0    0    0    0    1    0   16    0 5477]]
```

Figure 4. Confusion Matrix of CNN. The most common misclassification came from the shirt class.

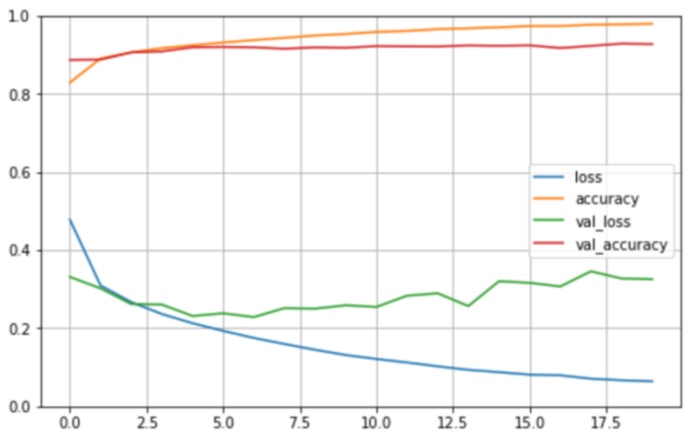


Figure 5. Plot of loss and accuracy from the CNN model. Large separation of validation loss and model loss over each epoch.

Summary and Conclusion

This model did achieve higher testing and validation accuracy, running into the most issue classifying shirts. Most commonly misclassifying shirts as T-shirts or Pullovers. This is similar to the results of the FCN. Both models do well with distinct items like shoes, sandals, and bags but struggle with more similar items such as shirts, t-shirts, and sweaters. The CNN model also takes longer to run than the FCN and sees a higher separation in validation and model loss. I ran into fewer issues overfitting the data when it came to the test data which I attribute to the reduced number of fully connected layers. Future work may include implementing drop-out layers or reduce the number of connections per layer.

Appendix A.

Packages:

- Numpy: Mathematical tools
- TensorFlow: Neural Network Package
- MatPlotLib: MATLAB style plotting commands and tools
- Pandas: Data manipulation package
- Sklearn: Package includes Confusion Matrix plotting

- Import tensorflow as tf: This command imports the TensorFlow package and calls it tf. This is done for all packages and renamed for ease of use.

- Np.newaxis*:* adds the additional axis to the data set. This is used to add the x1 for use in the CNN.

- tf.keras.models.Sequential: This is used to assemble my model by adding the layers I want to use sequentially in my code.

- tf.keras.layers.Dense: This pulls the fully connected layer function from the TensorFlow package.From here I can add the hyperparameters of my choice.

- tf.keras.layers.Conv2D: This is the convolutional layer from the TensorFlow package.

- Model.fit: This function trains my model taking in the number of epochs, training and validation data.

- Model.evaluate: Runs the test data on the model and reports accuracy.

Appendix B.

FCN

```
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()

X_train_full.shape

plt.figure()
for k in range(9):
    plt.subplot(3,3,k+1)
    plt.imshow(X_train_full[k], cmap ="gray")
    plt.axis('off')
plt.show()

y_train_full[:9]

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:]

X_train[:9]

from functools import partial

my_dense_layer = partial(tf.keras.layers.Dense, activation = "relu", kernel_regularizer =
tf.keras.regularizers.l2(0.00001))

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape = [28, 28]),
    my_dense_layer(300),
    my_dense_layer(150),
    my_dense_layer(100),
    my_dense_layer(10, activation = "softmax")
])

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

history = model.fit(X_train, y_train, epochs = 40, validation_data = (X_valid,y_valid))

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

y_pred = model.predict_classes(X_train)
conf_train = confusion_matrix(y_train,y_pred)
print(conf_train)

model.evaluate(X_test,y_test)
```

CNN

```
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()

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:]

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

from functools import partial

my_dense_layer = partial(tf.keras.layers.Dense, activation = "relu", kernel_initializer =
'he_uniform', kernel_regularizer = tf.keras.regularizers.l2(0.0001))
my_conv_layer = partial(tf.keras.layers.Conv2D, activation = 'relu', kernel_initializer =
'he_uniform', padding = "valid")

model = tf.keras.models.Sequential([
   my_conv_layer(32,(5,5),padding = "same", input_shape = [28,28,1]),
   tf.keras.layers.AveragePooling2D(2,2),
   my_conv_layer(64,(5,5)),
   tf.keras.layers.AveragePooling2D(2,2),
   my_conv_layer(128,(5,5)),
   tf.keras.layers.Flatten(),
   my_dense_layer(128),
   my_dense_layer(10, activation = "softmax")
])

model.compile(loss = "sparse_categorical_crossentropy",
             optimizer = tf.keras.optimizers.Adam(learning_rate=0.001),
             metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs = 20, validation_data = (X_valid,y_valid))

pd.DataFrame(history.history).plot(figsize=(8,5))
```

```
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()

y_pred = model.predict_classes(X_train)
conf_train = confusion_matrix(y_train,y_pred)
print(conf_train)

model.evaluate(X_test,y_test)
```

Reference

[1] Saha, S. (2018, December 17). A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way. Retrieved from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53