Fully-Connected and Convolutional Neural Networks

Makenna Barton - bartonmj@uw.edu

March 2020

Abstract This report outlines the techniques and procedures necessary to create, train, and test a Fully-Connected Neural Network and a Convolutional Neural Network for image classification.

1 Introduction and Overview

The data set of images that will be classified in this report are the following fashion items: T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot. They are matched with a tag, the numbers 0-9 respectively. Before making either neural network, the data from the Fashion-MNIST data set must be processed. A neural network needs separate data for training, validation, and testing. I take 5,000 images and make them our validation set and leave 55,000 to train with. The testing data comes in a set of 10,000 images. All of the data is scaled to between 0 and 1 as to convert the images to gray scale.

2 Theoretical Background

The neural networks created to solve this classification problem are all backed by neuroscience concepts and ideas. The connections and structure of the networks are modeled after neurons in the brain, aligned in an ordered, layered structure, the activation of one neuron triggering the activation of the next. This is applied to neural network models through the building blocks that make up every model: an input layer, hidden layers, weights and biases, an activation function, and an output layer. These elements make up all feed-forward multi-layer perceptron networks. Weights on inputs are similar to coefficients on inputs in linear regression, they work in the same way. The bias term can be connected to the added on variable in linear regression that shifts the function. Activation functions decide the threshold at which a neuron will be activated and the strength of the output. Activation functions map the sumed weights of the input to the output and are often written in a very familiar way:

$$\vec{y} = \sigma(A\vec{x} + \vec{b}) \tag{1}$$

where $\vec{y}$ is the outputs, A is a matrix of all the wights, $\vec{x}$ are the inputs and $\vec{b}$ are the biases. Common activation functions are the hyperbolic tangent or rectified linear unit (ReLU).

Neural networks can be deep, have many parallel layers, or wide, have many neurons per layers. Networks where every neuron from the previous layer is connected to everything in the next layer is called a Fully-Connected Neural Network. When training fully-connected networks, a phenomenon called overfitting, same as in regression, can appear and skew the accuracy of the model. Overfitting is when the model is no longer a generalizable model, it has been overly fit to the data that was used to train it. This can be overcome by adjusting the parameters of each layer within the model. In addition to the depth and width of a network and the activation function used, other parameters the affect the function of a network are the learning rate and the regularizer. The learning rate determines how much the weights of the system are updated during training, it controls how the model adapts to the problem, directly impacting the time it takes to train the model. The regluarizer helps to limit overfitting and adjust the loss function.

Looking at more complex neural networks such as convolutional networks, the structure gets more complex as well. The foundation of these networks is the idea that each neuron has a receptive field, a "search area". By looking at an image in smaller "search areas" you can get more detailed information about what is going on in each part of the image. Convolutional layers are made up of feature maps, a feature map being a representation of the original image through a filter applied in each receptive field of the neuron. The receptive fields are slid across the image to make up the entire image. The distance you move the field is called the stride. Because there can be many feature maps to one convolutional layer and each feature map has its own filter, the data can get very large. This is when it becomes very helpful to sub sample the image. This can be done in 2 ways, max pooling or average pooling. These methods are very self explanatory as the max pooling method chooses the max value from the receptive field to represent that portion of the image and the average pooling method takes the average of the pixel values in the receptive field to represent that portion of the image.

3 Algorithm Implementation and Development

For part 1, the fully connected neural network, I tried a variety of parameters to try to increase the accuracy of the model. My first attempt was to make the network wider, adding more neurons per hidden layer (500 and 300). This resulted in 89% overall accuracy in 15 epochs. The main confusion from this approach was between sneakers and sandals. I then tried networks that were less wide and more deep or used an L1 regularizer instead of L2 or had a learning rate of .0005. All versions of the model led to an accuracy rate of 87-88% over 15 epochs. Next I tried increasing the depth of the model again, 6 hidden layers with 650, 550, 450, 350, 250, 100 neurons respectively, using an L2 regularizer

Fully Connected Model Loss and Accuracy over Epochs

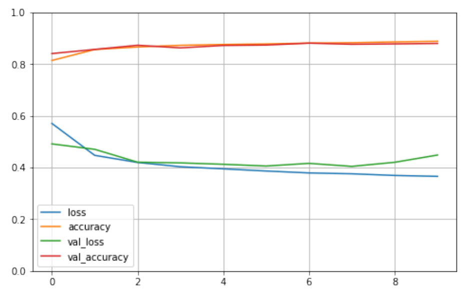


Figure 1: How the accuracy and loss for the final model varied at each epoch

(.0001) and with a learning rate of .0001 but this time with 25 epochs. The result was an accuracy level of 88% and the loss appeared to increase near the end. Although this was the highest overall accuracy, the validity accuracy was decreasing in the last few epochs, and I believe this was due to overfitting. To account for this I simplified the model and was able to achieve a validity accuracy of 88.02% and an overall accuracy of 88.85%. This model had 2 hidden layers, the first with 500 neurons and the second with 250, used a ReLU activation function, L2 regularizer (.0001) and an Adam optimizer with a learning rate of .00001. Figure 1 shows a graph of the losses and accuracy over the 10 epochs.

For part 2, created the convolutional neural network, I started with the outline for a standard LeNet-5 neural network. The first layer was a convolutional layer with 6 filters, each with a kernel size of 5. This layer had zero padding to allow all of the original data to be sampled. Next was an sub-sampling layer, an average pooling layer that down scaled the image by 2. Then another convolutional layer with a 5x5 filter, but this time had 16 filters. Next, another average pooling layer followed by the last convolutional layer, with 120 filters flattened and fully connected to the final hidden layer, a dense layer with 84 units. Over 5 epochs, this structure led me to a 89.37% overall accuracy. Looking to improve, I made the first major adjustment of switching to a max pooling sub-sampling layer instead of average pooling. This led to a huge improvement in overall accuracy over 10 epochs, 93.65%. I tried changing the regularizer and learning rate and saw no real improvements in accuracy. I then increased the number of filters in each convolutional layer to 10, 20, and 150 respectively. This change in structure resulted in 91.38% validity accuracy and 94.92% overall accuracy. I thought I may be able to see greater improvement from increasing the number of filters so I raised them to 20, 50, and 200 respectively. Over 10 epochs, with a learning rate of 0.001, an L2 regularizer(0.0001), and using the max pooling method on the sub sampling layers, I achieved a validity accuracy of 91.56% and an overall accuracy of 97.08%. Figure 2 shows the levels of loss and accuracy for the final convolutional model.

4 Computational Results

After achieving 88.02% accuracy in the validation data for the fully connected model, I used the model on the test data. The model correctly classified the images 86.59% of the time. The confusion matrix is shown in Figure 2. Most of the confusion in this model was having t-shirts classified as shirts.

Using the convolutional model that achieved 91.56% validation accuracy, I tested the model on the testing data. The evaluated data was classified with 90.75% accuracy. Based on the confusion matrix in Figure 4, this model has the most trouble distinguishing a t-shirt/top(0) from a shirt(6), note the high values in the coordinates (0,6) and (6,0).

5 Summary and Conclusions

Overall, working to classify these images showed me the power of neural networks and the benefits of both fully connected networks and convolutional networks. The fully connected network was significantly simpler to both understand and adjust parameters on. The simplicity of the network came to light as the network was quick to over-fit and wasn't able to classify images as accurately. The convolutional network was significantly more complex to build, with many more parameters to adjust. Due to the sophistication of the network, finely tuning the parameters to find the best combination for this problem was much more challenging than the fully connected network. The complexity of the network paid off in its extremely high accuracy rates. Once the model was built and parameters were selected, it converged to very high accuracy rates in very few epochs, unlike the fully connected network.

6 Appendix A - Python functions used and brief implementation explanation

`tf.keras.models.Sequential`- this function initializes a sequential stack of layers for a neural network model
`tf.keras.layers.Dense`- this creates a densely connected layer
`tf.keras.layers.Conv2D`- this function creates a convolutional network layer
`tf.keras.layers.MaxPooling2D`- this function performed the sub sampling of the network, specifically the max pooling method
`model`- any of the model function compile the layers into an object that can be trained, tested, and evaluated

Convolutional Model Loss and Accuracy over 10 Epochs

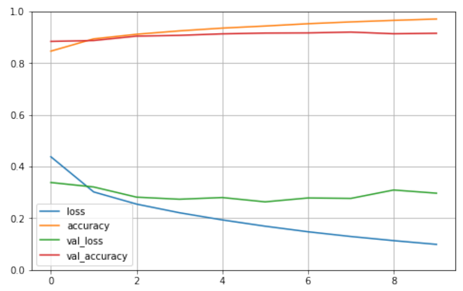
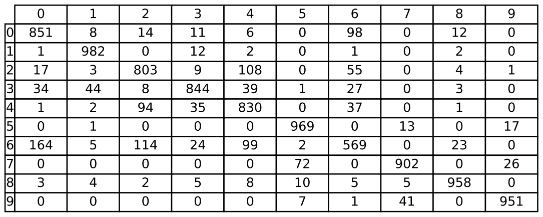


Figure 2: How the accuracy and loss for the final convolutional model varied between each iteration

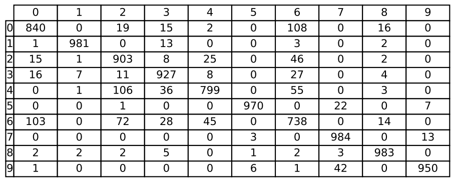
Confusion Matrix of Test Data on Fully Connected Model



	0	1	2	3	4	5	6	7	8	9
0	851	8	14	11	6	0	98	0	12	0
1	1	982	0	12	2	0	1	0	2	0
2	17	3	803	9	108	0	55	0	4	1
3	34	44	8	844	39	1	27	0	3	0
4	1	2	94	35	830	0	37	0	1	0
5	0	1	0	0	0	969	0	13	0	17
6	164	5	114	24	99	2	569	0	23	0
7	0	0	0	0	0	72	0	902	0	26
8	3	4	2	5	8	10	5	5	958	0
9	0	0	0	0	0	7	1	41	0	951

Figure 3: The index of the columns and rows correlate to the label of each clothing description. Each coordinate (row, col) describes how the image was classified by the model and what its true label is.

Confusion Matrix of Test Data on Convolutional Model



	0	1	2	3	4	5	6	7	8	9
0	840	0	19	15	2	0	108	0	16	0
1	1	981	0	13	0	0	3	0	2	0
2	15	1	903	8	25	0	46	0	2	0
3	16	7	11	927	8	0	27	0	4	0
4	0	1	106	36	799	0	55	0	3	0
5	0	0	1	0	0	970	0	22	0	7
6	103	0	72	28	45	0	738	0	14	0
7	0	0	0	0	0	3	0	984	0	13
8	2	2	2	5	0	1	2	3	983	0
9	1	0	0	0	0	6	1	42	0	950

Figure 4: Each coordinate (row, col) describes how the image was classified by the model and what its true label is.

7 Appendix B - Python Codes

Python Code

Part 1:

```python
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import confusion_matrix


# In[2]:

```

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


# In[3]:


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()


# In[4]:


y_train_full[:9]


# In[5]:


#Pre-processing - to double and scaled and splitting to training and validation
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:]


# In[6]:


from functools import partial

my_dense_layers = partial(tf.keras.layers.Dense, activation="relu", kernel_regul


model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    my_dense_layers(500),
    my_dense_layers(250),
    my_dense_layers(10, activation="softmax")
])


# In[7]:


model.compile(loss="sparse_categorical_crossentropy",
              optmizer=tf.keras.optimizers.Adam(learning_rate=0.00001),
              metrics=["accuracy"])


# In[8]:


history = model.fit(X_train, y_train, epochs=10, validation_data=(X_valid, y_val


# In[9]:


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


# In[10]:


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


# In[10]:


model.evaluate(X_test, y_test)


# In[11]:


y_pred = model.predict_classes(X_test)
conf_test = confusion_matrix(y_test, y_pred)
```

```
print(conf_test)


# In[12]:


fig, ax = plt.subplots()

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

df = pd.DataFrame(conf_test)
ax.table(cellText=df.values, rowLabels=np.arange(10), colLabels=np.arange(10), l
fig.tight_layout()
plt.savefig('NN_conf_mat.pdf')


# In[ ]:
```

Part 2:

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import confusion_matrix


# In[2]:


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


# In[3]:


#Pre-processing - to double and scaled and splitting to training and validation
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]


# In[4]:


from functools import partial

my_dense_layer = partial(tf.keras.layers.Dense, activation="tanh", kernel_regula
my_conv_layer = partial(tf.keras.layers.Conv2D, activation="tanh", padding="vali

model = tf.keras.models.Sequential([
    my_conv_layer(20,5,padding="same",input_shape=[28,28,1]),
    tf.keras.layers.MaxPooling2D(2),
    my_conv_layer(50,5),
    tf.keras.layers.MaxPooling2D(2),
    my_conv_layer(200,5),
    tf.keras.layers.Flatten(),
    my_dense_layer(84),
    my_dense_layer(10, activation="softmax")
])


# In[5]:


model.compile(loss="sparse_categorical_crossentropy",
             optmizer=tf.keras.optimizers.Adam(learning_rate=0.001),
             metrics=["accuracy"])


# In[6]:


history = model.fit(X_train, y_train, epochs=10, validation_data=(X_valid, y_val


# In[7]:
```

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


# In[62]:


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


# In[9]:


model.evaluate(X_test, y_test)


# In[10]:


y_pred = model.predict_classes(X_test)
conf_test = confusion_matrix(y_test, y_pred)
print(conf_test)


# In[11]:


fig, ax = plt.subplots()

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

df = pd.DataFrame(conf_test)
ax.table(cellText=df.values, rowLabels=np.arange(10), colLabels=np.arange(10), l
fig.tight_layout()
plt.savefig('CNN_conf_mat.pdf')


# In[ ]:
```