Neural Networks for Classifying Fashion MNIST

Amanda Lee

March 13, 2020

Abstract

I build and train two neural networks to classify articles of clothing in the Fashion-MNIST. One is fully-connected, the other is a convolutional neural network. The CNN performs better on the test data than the fully-connected network.

Introduction

I will build two classifiers using the Fashion-MNIST dataset. The dataset contains images of the ten different classes of clothing, t-shirts, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots. The first classifier will use a fully-connected neural network. The second will use a convolutional neural network.

Background

Artificial Neural Networks

An artificial neural network is a computing device that is based on the structure of neurons in the brain. The network is arranged in layers with the first layer being the input layer and the last layer being the output layer. There can be layers in between called hidden layers. Each layer is made up of nodes called neurons. The neurons in the input and output layer correspond to the data points in the input and output. In this case, the input is greyscale values of a 28 by 28 pixel image, so the input layer has 784 neurons. The output is a number from 0 to 9 that corresponds with the class of clothing. Since there are 10 possible outputs, there are ten neurons in the output layer. The number of neurons in a layer is referred to as the width of the layer. In a feed-forward network, such as the two networks that will be developed, the outputs from each neuron are sent to the neurons in the next layer, with no outputs being sent to previous layers. The input of a neuron is first put into an activation function. The neural networks developed below will use two types of activation functions. The first is the rectified linear unit (ReLu). This is equal to:

$$\max(0, x) \tag{1}$$

The other activation function that will be used is softmax. Softmax is used to map the output of a network to a probability distribution over the output classes. This is important for the output layer. The softmax function is defined as:

$$\sigma(z)_i = \frac{e^{zi}}{\sum_{j=1}^{K} e^{zj}} \text{ for } i = 1, \ldots, K \text{ and } z = (z_1, \ldots, z_K) \in \mathbb{R}^K \tag{2}$$

The value from the activation function is then put through a weight assigned to the connection between neurons. There are also bias neurons that are added to every layer of a network. Each bias neuron has a value of one and the weight on its connection to another neuron allows the inputs to the neuron to be shifted accordingly to better match the activation function. The weights and biases are found through an optimization algorithm that tries to minimize a loss function that represents the distance between the actual class of the training images and the class that the network guesses. The loss function that will be used for this classification task is called cross-entropy loss. The function used is:

$$L = -\frac{1}{N} \sum_{j=1}^{N} \ln(p_j) + \lambda ||\vec{w}||_2^2 \tag{3}$$

With N being the number of data points, p_j being the probability returned by the model for the actual class of input j, $\vec{w}$ being the vector of weights, and λ being the regularization constant. The first term is the cross-entropy loss and the second term is for regularization, preventing overfitting by ensuring that the weights assigned don't get too large.

Gradient descent will be used to train the model, finding the weights and biases that minimize the above loss function. This method works by starting at an initial guess, calculating the derivative, or gradient, at that point, then moving in the opposite direction of the gradient for a certain amount of time. This is repeated until the steps are very close together, indicating that a minimum has been reached. The amount of time used for each step is the learning rate. The type of gradient descent used is stochastic gradient descent. This means that mini-batches, or small subsets of the data, are used to calculate the gradient in one step of gradient descent. Once the entire dataset had been used, one epoch has passed. Another addition to gradient descent is momentum, meaning that previous gradients will have an effect on the steps taken. This can help prevent getting stuck in local minimums. Another part of the optimization process is backpropagation. This means that to compute the gradients, the weights at the end of the model are considered first.

In a fully-connected neural network, each neuron is connected to every neuron in the neighboring layers. Convolutional neural networks (CNN) are based on the visual cortex and are not fully-connected. Small filters that can detect certain features, such as a horizontal line, are moved across and down the image. The feature that the filter looks for depends on the weights for each pixel or data point in the filter. These weights are determined through the same optimization process detailed above. The weighted sums from the same filter, sweeped across the image, are constructed into a matrix, called a feature matrix. High values in the feature matrix would indicate the presence of the feature that the filter is looking for Features in the image are unlikely to be in the same location for every object within the

same class. Pooling layers can be used to subsample the result of the convolutional layers. This is done with a filter that is usually 2x2 pixels and has a stride of two pixels. There are two ways of pooling. Average pooling calculates the average value within each section of the feature map. Maximum pooling calculates the maximum value within each section of the feature map. Near the end of a CNN, there is typically one or more dense layer where the neurons are fully connected.

Algorithm Implementation

The code for building both neural networks involves loading in the fashion-MNIST dataset, converting the values in the dataset to double precision numbers that range from zero to one, and taking a subsection of the training dataset to use as validation data. For the CNN, the convolutional layers required the data to have an extra axis, so an extra axis was added to the training and validation data.

For the fully-connected neural network, I built a simple model to start out with, then I adjusted the model in different ways, such as adding or taking away layers, adjusting the width of the layers, trying different activation functions, changing the regularization function and constant, and changing the learning rate. I kept the model that had the highest validation accuracy. The final model had four hidden layers, with 500, 400, 300, and 100 neurons. I used a learning rate of 0.0002 to train the model.

My process for building the CNN was similar. I started out with a basic model based on another CNN and took away or added convolutional and pooling layers to see the effect on the accuracy. In the final model, every convolutional layer had 64 filters with a size of 3 by 3 pixels. The model consisted of an initial convolutional layer, followed by a max pooling layer, then two convolutional layers and a max pooling layer. After that was a flatten layer to reshape to inputs for a dense layer of width 300 and finally, the output layer.

Results

For the fully connected neural network, it was difficult to find ways of improving on the network. The factor that had the largest impact on the accuracy of the classifier was the architecture of the network, such as adding or taking away dense layers and changing the widths of the layers. I was able to get the test accuracy to 87.36%, but not past 90%.

The CNN had a better accuracy of 91.8% on the test data. I found that the accuracy improved when I lowered the number of times there would be a convolutional layer followed by a pooling layer. Ultimately, I found that only two pooling layers were needed.

Both networks had some difficulty differentiating different types of tops (t-shirts, pullovers, coats, and long-sleeved shirts) and different types of shoes (sandals, sneakers, and ankle boots). The fully-connected network had more trouble than the CNN with differentiating between sandals and ankle boots. The CNN could classify sandals relatively well, possibly

	0	1	2	3	4	5	6	7	8	9
0	894	0	5	19	1	2	76	0	3	0
1	4	964	0	25	2	0	4	0	1	0
2	38	0	749	17	71	1	121	0	3	0
3	29	4	3	930	7	0	24	0	3	0
4	1	1	92	69	693	0	138	0	6	0
5	0	0	0	1	0	983	0	13	1	2
6	172	0	47	33	33	0	703	0	12	0
7	0	0	0	0	0	32	0	955	0	13
8	5	0	1	7	1	2	9	3	972	0
9	0	0	0	0	0	35	1	71	0	893

Figure 1: This is the confusion matrix for the fully-connected network on the test data.

because of its ability to learn certain features. Both networks had an easier time classifying trousers, likely due to their distinctive shape among the other fashion items.

Conclusion

The CNN was more effective than a fully-connected network for classifying images. This is likely because the CNN identifies certain features of the the classes in the dataset to look for, which is more similar to human perception of these images. The architecture of neural networks has a larger effect of the performance of the model than other factors, such as the learning rate or regularization constant. The ReLu activation function turned out to work the best for both neural networks, despite its weaknesses. Ultimately, both networks were unable to get an accuracy above 95%, likely because of the difficulty in differentiating between certain types of tops.

Appendix A

`tf.keras.datasets.fashion_mnist.load_data()` - Loads the fashion-MNIST data.

`functools.partial(func[,*args][, **keywords])` - Returns an object that when called will behave like the `func` specified. This is used to add layers to the model more easily.

`tf.keras.layers.Dense()` - Initializes a dense layer.

`tf.keras.layers.Conv2D()` - Initializes a 2D convolutional layer.

`tf.keras.layers.Flatten()` - Initializes a layer that flattens the input.

	0	1	2	3	4	5	6	7	8	9
0	860	0	16	17	4	1	93	0	9	0
1	1	987	0	8	1	0	1	0	2	0
2	13	1	885	9	53	0	38	0	1	0
3	15	3	7	946	15	0	13	0	1	0
4	0	0	44	38	870	0	47	0	1	0
5	0	0	0	0	0	987	0	8	0	5
6	86	1	66	30	73	0	729	0	15	0
7	0	0	0	0	0	12	0	967	0	21
8	2	1	1	3	3	3	1	2	984	0
9	0	0	1	0	0	4	0	30	0	965

Figure 2: This is the confusion matrix for the convolutional neural network on the test data.

`tf.keras.layers.MaxPool2D()` - Initializes a 2D max pooling layer.

```
tf.keras.Sequential.compile(loss="sparse_categorical_crossentropy",
    optimizer=tf.keras.optimizers.Adam(learning_rate= a), metrics=["accuracy"])
```
Configures the model for training with the specified loss function, optimizer, learning rate, and metrics to evaluate.

```
tf.keras.Sequential.fit(X_train, y_train, epochs=5,
    validation_data=(X_valid,y_valid))
```
Trains the model for 5 epochs using the specified training and validation data.

`y_pred = tf.keras.Sequential.predict_classes(X_test)` - Returns the model's class predictions of the test data.

Appendix B

Part I

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

# In[504]:


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


# In[505]:


# Load mnist data
fashion_mnist = tf.keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()


# In[506]:


# Convert integer values to floating point numbers and remove 5,000 images to use as validation 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:]


# In[564]:


from functools import partial

my_dense_layer = partial(tf.keras.layers.Dense, activation="relu", kernel_regularizer=tf.keras.regularizers.l2(0.0001))

model1 = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    my_dense_layer(500),
    my_dense_layer(400),
    my_dense_layer(300),
    my_dense_layer(100),
    my_dense_layer(10, activation="softmax")
])
```

```
49
50
51 # In[565]:
52
53
54 model1.compile(loss="sparse_categorical_crossentropy",
55                optimizer=tf.keras.optimizers.Adam(learning_rate=0.0002),
56                metrics=["accuracy"])
57
58
59 # In[566]:
60
61
62 history = model1.fit(X_train, y_train, epochs=5, validation_data=(X_valid,
       y_valid))
63
64
65 # In[567]:
66
67
68 pd.DataFrame(history.history).plot(figsize=(8,5))
69 plt.grid(True)
70 plt.gca().set_ylim(0,1)
71 plt.show()
72
73
74 # In[568]:
75
76
77 y_pred = model1.predict_classes(X_test)
78 conf_test = confusion_matrix(y_test, y_pred)
79 print(conf_test)
80
81
82 # In[569]:
83
84
85 np.sum(y_pred == y_test)/y_test.size
86
87
88 # In[570]:
89
90
91 fig, ax = plt.subplots()
92
93 # hide axes
94 fig.patch.set_visible(False)
95 ax.axis('off')
96 ax.axis('tight')
97
98 # create table and save to file
99 df = pd.DataFrame(conf_test)
100 ax.table(cellText = df.values, rowLabels = np.arange(10), colLabels = np.
       arange(10), loc = 'center', colLoc = 'center')
101 fig.tight_layout()
```

```
plt.savefig('conf_mat_part_1.png')


# In[ ]:
```

Part II

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

# In[2]:


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


# In[3]:


# Load mnist data
fashion_mnist = tf.keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()


# In[4]:


# Convert integer values to floating point numbers and remove 5,000 images to use as validation 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:]

# Add extra axis
X_train = X_train[..., np.newaxis]
X_valid = X_valid[..., np.newaxis]
X_test = X_test[..., np.newaxis]


# In[132]:


from functools import partial

my_dense_layer = partial(tf.keras.layers.Dense, activation="relu", kernel_regularizer=tf.keras.regularizers.l2(0.0002))
my_conv_layer = partial(tf.keras.layers.Conv2D, activation="relu", padding="same")
```

```
46
47 model2 = tf.keras.models.Sequential([
48     my_conv_layer(64,3,padding="same",input_shape=[28,28,1]),
49     tf.keras.layers.MaxPool2D(),
50     my_conv_layer(64,3),
51     my_conv_layer(64,3),
52     tf.keras.layers.MaxPool2D(),
53     tf.keras.layers.Flatten(),
54     my_dense_layer(300),
55     my_dense_layer(10, activation="softmax")
56 ])
57
58
59 # In[133]:
60
61
62 model2.compile(loss="sparse_categorical_crossentropy",
63                optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
64                metrics=["accuracy"])
65
66
67 # In[134]:
68
69
70 history = model2.fit(X_train, y_train, epochs=5, validation_data=(X_valid,
       y_valid))
71
72
73 # In[120]:
74
75
76 pd.DataFrame(history.history).plot(figsize=(8,5))
77 plt.grid(True)
78 plt.gca().set_ylim(0,1)
79 plt.show()
80
81
82 # In[137]:
83
84
85 y_pred = model2.predict_classes(X_test)
86 conf_test = confusion_matrix(y_test, y_pred)
87 print(conf_test)
88
89
90 # In[145]:
91
92
93 fig, ax = plt.subplots()
94
95 # hide axes
96 fig.patch.set_visible(False)
97 ax.axis('off')
98 ax.axis('tight')
99
```

```
100 # create table and save to file
101 df = pd.DataFrame(conf_test)
102 ax.table(cellText = df.values, rowLabels = np.arange(10), colLabels = np.
        arange(10), loc = 'center', colLoc = 'center')
103 fig.tight_layout()
104 plt.savefig('conf_mat_part_2.png')
105
106
107 # In[ ]:
```