Homework 5: Neural Networks for Classifying Fashion MNIST

Yilin Li

March 13 2020

Abstract

In this project, I built a classifier for the Fashion-MNIST data set of 10 different classes of fashion items. I used two types of neural network to classify the images in the Fashion-MNIST data set. Additionally, I adjusted the neural network classifier by changing the hyperparameters to try to achieve the best accuracy I could get, and then used those parameters to build my final model on the test data. In the end, I checked the accuracy and the confusion matrix for the test data.

Sec.I. Introduction and Overview

Fashion-MNIST is a data set that contains images of 10 different classes of fashion items. In this project, I used two types of neural network to classify the images in the Fashion-MNIST data set. To be specific, in Part I, I trained a fully-connected neural network; in Part II, I repeated the same procedure as I did in Part I but used a convolutional neural network. I tried several different neural network architectures with different hyperparameters to try to achieve the best accuracy I could get and used those parameters to train a final model to use on the test data and report my results with the accuracy and the confusion matrix for the test data.

The Theoretical Background section introduces the theorems used in the project.

The Algorithm Implementation and Development section lists the algorithms that were implemented in this project and how they were developed.

The Computational Results section shows the computational results of Python and analyzes it.

The Summary and Conclusion section summarizes the project and makes a conclusion.

The Python functions and implementations used in this project were attached in Appendix A. The Python codes were attached in Appendix B.

Sec.II. Theoretical Background

Information was gathered from lectures and Wikipedia.

Artificial Neural Network (ANN)

The ANN approach was originally used for solving problems in the same way that a human brain would. Over time, the ANN approach was used more to performing specific tasks, such as computer vision, speech recognition, machine translation, social network filtering, and playing board and video games. The components of ANNs are **Neurons**, which receive input, combine the input with their internal state and an optional threshold using an activation function, and produce outputs accomplish the task, **Connections and Weights**, which provide the output of one neuron as an input to another neuron, and **Propagation function**, which computes the input to a neuron from the outputs.

The neurons are organized into multiple layers. Another necessary component is **Hyper-parameter**, which is a constant parameter that should be set before the learning process begins.

Last but not the least, **learning**, which is the adaptation of the network to better handle a task by considering sample observations. The following figure shows the theorem of the ANN.

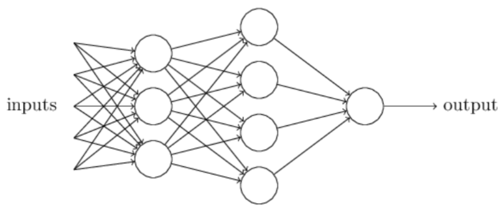


Figure 1: An Artificial Neural Network

The Fully-connected Neural Network

A fully-connected Neural Network consists of a series of fully connected layers, which is a function from R_m to R_n. Each output dimension depends on each input dimension.

The mathematical form of a fully connected network is: let x $\in \mathbb{R}^m$ represents the input to a fully-connected layer. Let $y_i \in \mathbb{R}$ be the i^{th} output from the fully-connected layer. Then $y_i \in \mathbb{R}$ is computed as:

$$y_i = \delta(w(1 * 1) + ... + w(m * m)) \tag{1}$$

The Convolutional Neural Network (CNN)

The convolutional Neural Network (CNN) is a class of deep neural networks that is most commonly applied to analyzing visual imagery. They are also known as *shift invariant* or *space invariant artificial neural network.* They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, and financial time series.

The convolutional layer is the core building of a CNN. The parameters of the layer consist of a set of learnable kernels, which have a small receptive field, but extend through the full depth of the input volume.

In order to deal with high-dimensional inputs such as images, the CNN exploit spatially local correlation by enforcing a sparce local connectivity pattern between neurons of adjacent layers, which means that each neuron is connected to only a small region of the input volume, The connections of the CNN are local in space, but always extend along the entire depth of the input volume.

Sec.III. Algorithm Implementation and Development

Part I. The Fully-connected Neural Network

- Import the corresponding library and package.
- Load the Fashion_MNIST data.
- Plot the first ten images of the fashion items in row k (NOT necessary).
- Before begin, preprocess the data:
  - Removing $5,000$ images from the training data to use as validation data
  - Ending up with $55,000$ training examples in an array **X_train** (the training data itself is $55,000 \times 28 \times 28$) and $5,000$ validation examples in an array **X_valid**.
  - Convert the numbers in the arrays **X_train**, **X_valid**, and **X_test** to floating point numbers between 0 and 1 by dividing each by 255.0.
- Define neural network, the model
  - Firstly, we want to flatten out the input layer.
  - Use **tf.keras.layer.Dense** to create three dense layers (try different number of layers and width of the layers here, but always ended with 10 probabilities because there are 10 classes that we want to classify).
  - Determine the activation functions that we will use.
  - Determine the **regularization parameters** that we will use: small 2-norm for the weights (small constant means add a little bit, conversely, big means add a lot).
- Decide some options for training.
  - Determine theright way to classify.
  - Specify an optimizer (large **learning_rate** means change a lot every time, conversely, small means change a little bit each time.)
  - Check the accuracy.
- Train the neural network:
  - Fit our model to the data (The epochs are the forward pass plus the backward pass for all training samples, add more to do more passes).

- Plot training and validation losses and accuracies. Check the overfitting (training loss much lower than validation loss), if there is no overfitting, it is safe for us to widen the layers to see whether we can get a better accuracy.
- When we satisfy with the accuracy, use those hyperparameter values to train one final model.
- Print out the confusion matrix for the training data (the rows of the confusion matrix represents the prediction, and the columns represents what the actual labels are).
- Check the accuracy of the final model on the test data.
- Print out the confusion matrix of the test data and save as a *png.* file.

Part II. The Convolutional Neural Network

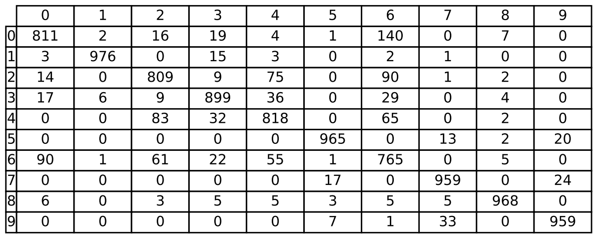
There are many similarities between the fully-connected neural network and the convolutional neural network. Therefore, this part only shows the differences and additions compare to Part I.

- Change the training data **X_train** from $55,000 \times 28 \times 28$ to $55,000 \times 28 \times 28 \times 1$ by adding one extra axis to **X_train**, **X_valid**, and **X_test**.
- Define neural network, the model
  - Instead of starting by flattening the output, go straight to the 2-dimentional convolutional layer (because the image is 2D) by determining the number of filters for the layer, the size of the filter, the padding option, the activation function, and the input size ($28 \times 28 \times 1$).
  - Create the average pooling layer.
  - Create three total convolutional layers, and set the **padding** valid.
  - Use **tf.keras.layers.Flatten** to heading the convolutional layers into the dense layers and match the activation type of the dense layer to the convolutional layer (which is "tanh").

Sec.IV. Computational Results

Part I. The Fully-connected Neural Network

Originally, I have a) the depth of the network: 3; b) the width of the layers: 300, 100, 10; c) the learning rate: 0.001; d) the regularization parameters: 0.0001; e) the activation functions: "relu"; f) the optimizer: Adam; and g) the epochs: 5. I got the validation accuracy as 88.82%. Firstly, I changed the activation function to "tanh", which is what we used in the convolutional neural network, and I got 87.5% as the accuracy, which was lower than the original one. Then, I changed the epochs to 10, which made the accuracy a little bit higher as 87.5%. I changed the activation function back to "relu", and add one more layer between the first layer and the third layer with the width 200. This made the accuracy higher as 89.06%, and I increased the epochs to 15, this brought the accuracy to 89.56% which was close to 90%. However, when I increased the epochs to 20, it got worse with the accuracy decreased to 87.96%. Since it took pretty long time to get the result, I changed the learning rate to 0.01, which made the accuracy went back to 88%. At last, I tried to add one more layer between the third one and the fourth one, and I got the accuracy as 87.04%. Therefore, I decided to use the hyperparameters when I got 89.56%. The following is the confusion matrix for the test data:



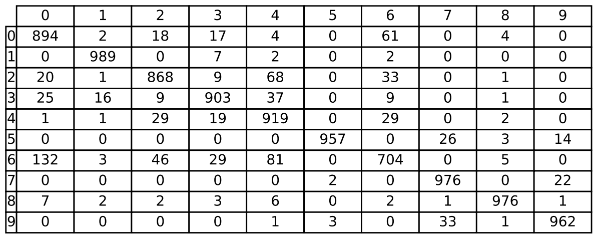
	0	1	2	3	4	5	6	7	8	9
0	811	2	16	19	4	1	140	0	7	0
1	3	976	0	15	3	0	2	1	0	0
2	14	0	809	9	75	0	90	1	2	0
3	17	6	9	899	36	0	29	0	4	0
4	0	0	83	32	818	0	65	0	2	0
5	0	0	0	0	0	965	0	13	2	20
6	90	1	61	22	55	1	765	0	5	0
7	0	0	0	0	0	17	0	959	0	24
8	6	0	3	5	5	3	5	5	968	0
9	0	0	0	0	0	7	1	33	0	959

Figure 2: The Confusion matrix for the test data

The accuracy of my final model on the test data was 89.29%. From the confusion matrix we can see that the biggest mistake is the 140 on [0,6]. The row represents the prediction and the column represents what the actual labels are. The 0 class is "T-shirt", and the 6th class is "shirt", and they are really similar to each other, so this makes sense that the network got confused on them.

Part II. The Convolutional Neural Network

Originally, I have a) the number of filters: 6, 16, 12; b) the kernel size: (5,5); c) the default stride which is (1,1); d) the padding options "valid"; e) the pool sizes for pooling layers: 2. The accuracy of the validation data was 88.92%. First of all, I changed the number of filters to 64, 64, 120, the kernel size to 4, and increased the epochs to 10. I got the accuracy of my validation data as 89.4%. Since the improvement was not pretty significant, I changed the number of filters to 120, 120, 120 and the epochs to 12. It took pretty long time to do the last one, so I changed the learning rate to 0.01. However, it still took about 10 minutes and I got a pretty bad accuracy around 80%. I changed the number of the filters to 32, the kernel size to (3,3), increased the number of the dense layer to 100, and changed the pooling type from *AvePooling* to *MaxPooling*. This time I got the accuracy increased to 83% which was still pretty low. At last, I changed the activation functions from "tanh" to "relu" and the learning rate back to 0.001. Finally, I got my accuracy of the validation data as 91.44%. Therefore, I decided to use the hyperparameters when I got this accuracy. The following is the confusion matrix for the test data:



	0	1	2	3	4	5	6	7	8	9
0	894	2	18	17	4	0	61	0	4	0
1	0	989	0	7	2	0	2	0	0	0
2	20	1	868	9	68	0	33	0	1	0
3	25	16	9	903	37	0	9	0	1	0
4	1	1	29	19	919	0	29	0	2	0
5	0	0	0	0	0	957	0	26	3	14
6	132	3	46	29	81	0	704	0	5	0
7	0	0	0	0	0	2	0	976	0	22
8	7	2	2	3	6	0	2	1	976	1
9	0	0	0	0	1	3	0	33	1	962

Figure 3: The Confusion matrix for the test data

The accuracy of my final model on the test data was 91.48%. From the confusion matrix we can see that the biggest mistake is the 132 on [6,0]. The row represents the prediction and the column represents what the actual labels are. The 0 class is "T-shirt", and the 6th class is "shirt", and they are really similar to each other, so this makes sense that the network got confused on them.

Sec.V. Summary and Conclusions

In this project, I used the fully-connected neural network and the convolutional neural network to classify the images of 10 different classes of fashion items. I found that the learning rate could influence the accuracy a lot. Similar hyperparameters could give really different result. Additionally, I found that the more different that the images are, the easier the network could classify. For me, it seems that I could relatively got a higher accuracy by using the convolutional neural network compared to the fully-connected neural network. However, both of these two neural network got really confused on classifying the "T-shirt" images from the "Shirt" images or conversely.

Appendix A

- **tf.keras.datasets.fashion_mnist** Loads the fashion_MNIST dataset.
- **.shape** The shape property is usually used to get the current shape of an array.
- **figure()** is to create a figure object.
- **tf.keras.layers.Dense** Just your regular densely-connected NN layer.
- **tf.keras.models.Sequential** Linear stack of layers.
- **tf.keras.layers.Flatten** Flattens the input. Does not affect the batch size.
- **tf.keras.optimizers.Adam** Optimizer that implements the Adam algorithm.

- **model.compile** Configures the model for training.
- **model.fit** Trains the model for a fixed number of epochs (iterations on a dataset).
- **pd.DataFrame** a 2-dimensional labeled data structure with columns of potentially different types.
- **model.predict_classes** Generate class predictions for the input samples.
- **confusion_matrix** Compute confusion matrix to evaluate the accuracy of a classification.
- **model.evaluate** Evaluates the model on a data generator.
- **tf.keras.layers.Conv2D** 2D convolution layer (e.g. spatial convolution over images).
- **keras.layers.AveragePooling2D()** Average pooling operation for spatial data.

Appendix B

Part I. The Fully-connected Neural Network

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

# In[15]:



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


# In[16]:


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


# In[17]:


X_train_full.shape


# In[18]:

```

```

# Plot the first images of fashion items in row k (Not necessary)
plt.figure()
for k in range(10):
    plt.subplot(2,5,k+1)
    plt.imshow(X_train_full[k], cmap='gray')
    plt.axis('off')
plt.show()


# In[19]:


# Try
y_train_full[:10]


# In[20]:


# Preprocess the data

# Convert X_train, X_valid, and X_test to floating point numbers between 0 and 1 by dividing each by 255.0
# Remove 5,000 images from the training data to use as validation data
X_valid = X_train_full[:5000] / 255.0
# End up with 55,000 training examples in an array X_train
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[21]:


# Part I

from functools import partial

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]),
    my_dense_layer(300),
    my_dense_layer(200),
    my_dense_layer(100),
    my_dense_layer(10, activation="softmax")
])


# In[30]:
```

```python


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


# In[31]:


history = model.fit(X_train, y_train, epochs=15, validation_data=(X_valid, y_valid))


# In[32]:


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


# In[33]:


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


# In[34]:


model.evaluate(X_test,y_test)


# In[35]:


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


# In[37]:


fig, ax = plt.subplots()

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

```python

# 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_mat2.pdf')


# In[ ]:
```

Part II. The Convolutional Neural Network

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

# In[11]:


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


# In[12]:


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


# In[13]:


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[57]:


from functools import partial

```

```python
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([
    my_conv_layer(32,3,padding="same",input_shape=[28,28,1]),
    tf.keras.layers.MaxPooling2D(2),
    my_conv_layer(64,3),
    tf.keras.layers.Flatten(),
    my_dense_layer(100),
    my_dense_layer(10, activation="softmax")
])


# In[58]:


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


# In[59]:


history = model.fit(X_train, y_train, epochs=12, validation_data=(X_valid,
    y_valid))


# In[60]:


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


# In[61]:


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


# In[62]:


model.evaluate(X_test,y_test)


# In[64]:
```

```python


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


# In[65]:


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_mat33.pdf')


# In[ ]:
```