Homework 5: Neural Networks for Classifying Fashion MNIST

Jiayi Wang

Abstract

In this report, we will see how artificial neural networks are being used as a machine learning tool in real life to classify fashion items. We will perform different neural network architectures with different hyperparameters and try to achieve the best accuracy on the validation data in order to classify ten fashion items.

Introduction and Overview

As a specific application for neural networks, suppose we will work with an analogous data set, and there are images of 10 different classes of fashion items in the file called **fashion_mist.mat**. There are 60,000 training images in the array X_train and 10,000 test images in the array X_test, each of which is 28 × 28 pixels. The labels are contained in the vectors y_train and y_test. The values in these vectors are numbers from 0 to 9, but they correspond to the 10 classes in the following way: 0 for t-shirt/top, 1 for trouser, 2 for pullover, 3 for dress, 4 for coat, 5 for sandal, 6 for shirt, 7 for sneaker, 8 for bag, and 9 for ankle boot. In the first and second part of this problem, we will train a fully-connected neural network and a convolutional neural network respectively to classify the images in the data set. We would experience different hyperparameters to achieve the best accuracy on the validation data in both cases. At last, we will do the comparison between these two neural networks.

Theoretical Background

- **Artificial neural networks (ANN)**

  Artificial neural networks (ANN) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules.

  - *Components of ANN*

    <u>*Neurons:*</u> ANNs are composed of artificial neurons which retain the biological concept of neurons, which receive input, combine the input with their internal state (activation) and an optional threshold using an activation function, and produce output using an output function. The initial inputs are external data, such as images and documents. The ultimate outputs accomplish the task, such as recognizing an object in an image. One activation function that has been used a lot is the hyperbolic tangent function $\sigma(x) = tanh(x)$. It has the same shape as the logistic function by goes from -1 to 1. You could refer to both functions as sigmoid functions.

The activation function that is probably the most popular today is the rectified linear unit (ReLU) which is $\sigma(x) = max(0, x)$. So applying the ReLU activation function to a layer means you zero out all the entries that are negative and leave the positive entries alone.

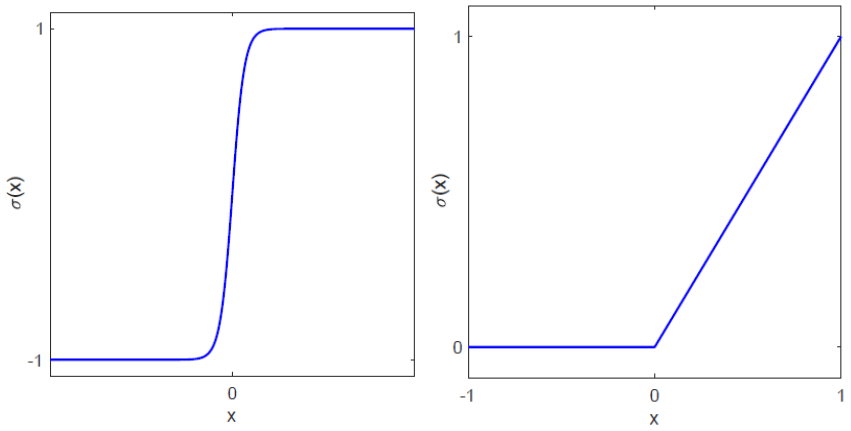


Figure 1: Hyperbolic tangent (left) and ReLU (right)

Connections and weights: The network consists of connections, each connection providing the output of one neuron as an input to another neuron. Each connection is assigned a weight that represents its relative importance. A given neuron can have multiple input and output connections.

Propagation function: The propagation function computes the input to a neuron from the outputs of its predecessor neurons and their connections as a weighted sum. A bias term can be added to the result of the propagation.

- *Hyperparameter*

  A hyperparameter is a constant parameter whose value is set before the learning process begins. The values of parameters are derived via learning. Examples of hyperparameters include learning rate, the number of hidden layers and batch size. The values of some hyperparameters can be dependent on those of other hyperparameters. [1]

- **Fully connected neural network**

  A fully connected neural network consists of a series of fully connected layers. A fully connected layer is a function from $\mathbb{R}^m$ to $\mathbb{R}^n$. Each output dimension depends on each input dimension. [2]

- **Convolutional neural network (CNN)**

  A Convolutional Neural Network is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer

1 Cited from https://en.wikipedia.org/wiki/Artificial_neural_network
2 Cited from https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html

neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features.[3]

Algorithm Implementation and Development

Before coding, we first would like to import the relevant libraries, which includes: numpy, tensorflow, matplotlib.pyplot, pandas, and confusion_matrix.

Next, we declare a variable called *fashion_mnist* and set it to an object on the MNIST dataset. We then load the data and output four things which are called: *X_train_full, y_train_full, X_test,* and *y_test.* The variable *X_train_full* contains 60000 images and *X_test* contains 10000 images each with 28 pixels by 28 pixels. *y_train_full* and *y_test* are one dimensional arrays that contain integers corresponding to the fashion items which we talked about in the introduction.

As we know, validation data plays the role of test data while we are trying to adjust aspects of our network and choose the right model. It can also help us figure out when to stop training our network, so we would like to spit our training data into training and validation data. To do so, we remove the first 5,000 images from our training data to use as validation data, so we end up with 55,000 training examples in an array *X_train* and 5,000 validation examples in an array *X_valid*. We also remove the first 5,000 labels from our training labels to use as validation labels. Since the numbers in the arrays *X_train*, *X_valid*, and *X_test* are integers from 0 to 255, so we will then convert them to floating point numbers between 0 and 1 by dividing each by 255.0.

In the first part, we would build a fully-connected neural network. To build our neural network model, we will use the command *tf.keras.model.Sequential()*. Inside of this command, we will put our layers. We will first flatten out each of our images into a one-dimensional array by using *tf.keras.layers.Flatten()*. Then we will use *tf.keras.layers.Dense()* to add fully-connected layers; inside of the parentheses, we can put the number of neurons, activation functions, activation functions, regularization parameters, etc. which we will experience later. We will set the output layer to have 10 neurons since there are ten fashion items that need to be classified and we will feed them through a "softmax" activation function.

3 Cited from http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

To change our training options, we will use *model.compile()*. Inside of the parentheses, we are able to adjust the learning rate, optimizer, and matrics.

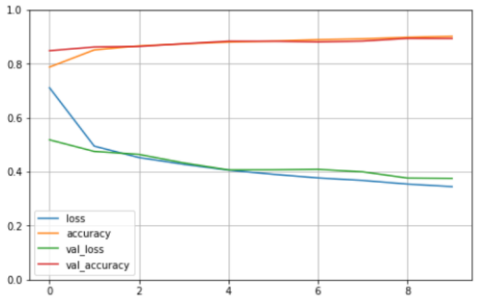
To begin the training process, we will feed the training data, validation data and the number of epochs to the command *model.fit()*. To get more information, we can plot the training and validation loss and accuracy by using the command *pd.DataFrame()*. We can also plot the confusion matrix by first getting the prediction of our training data by using *model.predict_classes()*, then with the help of *confusion_matrix()*, and *print()*, we are able to get the confusion matrix of our training data.

After we get the accuracy which meets our satisfaction, we may evaluate our testing data by using *model.evaluate()* and use the same way to plot the confusion matrix of our testing data.

In part two, since we will build a convolutional neural network, we will change the dimension of each image to be 28×28×1 by adding another column in *X_train*, *X_valid*, and *X_test*. Then we will change our model by using the template of some successful neural network model, for example: AlexNet, GoogleNet, VGGNet, etc to see if we can get better accuracy compared to the model in part one.

Computational Results

In part 1, by running our initial model, which has two fully connected hidden layers: one with 300 neurons, one with 100 neurons and both with ReLU activation function, we got the 88.74% accuracy on the validation data. We then tried other activation functions, such as “selu,” “sigmoid,” “tanh,” none of the accuracy exceed 90%. We then tried to adjust the learning rate, we found out larger learning rate will result in overfitting. We also tried different regularizers: l1, l1_l2, different optimizer: Adamax, Adadelta, Adagrad, etc, none of them gave us better accuracy than our initial model. At last, we tried to adjust the number and the width of the layers and finally decided to have four hidden layers with number of neurons: 300, 200, 64, 20 which are gradually decreasing from 784 to 10. The accuracy on the validation data is 89.46%. Although it is not beating the 90% expected accuracy, this is the best we can do so far. The values of parameters can be found in the appendix B and figure 1 shows the plot of training and validation loss and accuracy (left) and confusion matrix of the training data (right). From the left plot, the accuracy of our training data and validation data are relatively similar. We had a little bit overfitting at the end, but at least we didn’t see any increasement in our validation loss. From the confusion matrix, we can see our model had trouble to identify among 0,2, 4 and 6, which represents t-shirt/top, pullover, coat, and shirt respectively, which is understandable.



	0	1	2	3	4	5	6	7	8	9
0	4750	8	58	167	9	1	509	0	41	0
1	11	5370	2	53	3	1	3	0	1	0
2	47	10	4735	80	304	0	295	0	25	0
3	71	51	19	5176	92	1	74	0	15	0
4	11	19	563	235	4352	0	300	0	32	0
5	0	0	0	0	0	5417	0	74	6	10
6	584	6	401	162	249	0	4041	0	64	0
7	0	0	0	0	0	29	0	5396	6	57
8	6	5	14	12	3	4	15	8	5443	0
9	0	1	0	0	0	11	0	215	3	5264

Figure 1: Plot of training and validation loss and accuracy with fully-connected neural network(left)

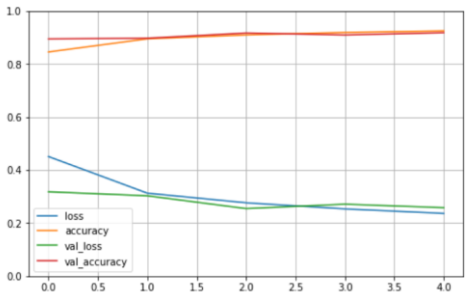
Confusion matrix for the test data with fully-connected neural network (right)

The accuracy of our final model on the test data is 87.98%. Below is the confusion matrix for the test data, we can see our model had trouble to identify among 0, 2, 4 and 6, which represents t-shirt/top, pullover, coat, and shirt respectively, which is understandable and match the result from the training data.

	0	1	2	3	4	5	6	7	8	9
0	824	4	13	38	2	0	106	0	13	0
1	2	970	0	21	4	0	3	0	0	0
2	15	1	835	18	71	0	56	0	4	0
3	21	14	6	912	16	0	27	0	4	0
4	0	3	131	60	741	0	60	0	5	0
5	0	0	0	1	0	965	0	21	2	11
6	122	4	95	35	61	0	665	0	18	0
7	0	0	0	0	0	18	0	966	1	15
8	3	1	3	7	2	2	3	3	976	0
9	0	0	0	0	0	6	1	49	0	944

Figure 2: Confusion matrix for the test data with fully-connected neural network

In part 2, we were inspired by the neural network LeNet-5 and decided to experience other successful ones, for example: AlexNet, GoogleNet, and VGGNet. Due to complexity of some networks mentioned above , we decided to refer to AlexNet which first uses the combination of convolutional layers and maximum pooling layers, then flatten through the fully connected layers, and we will do some adjustment on top of it. Detailed model and hyperparameter chosen are included in the appendix B. This time, the accuracy on the validation data is 92.54% which is way better than our model in part 1. From the left plot below, the accuracy of our training data and validation data are relatively similar. We had a little bit overfitting at the end, but at least we didn't see any increasement in our validation loss and everything tends to flatten out. From the confusion matrix below, we can see the result definitely has been improved compared to that in part 1; for example, way less 2 (pullover) are misclassified as 6 (shirt).



	0	1	2	3	4	5	6	7	8	9
0	5065	0	41	65	14	1	349	0	8	0
1	3	5363	1	63	6	0	4	0	4	0
2	74	3	4885	23	328	0	183	0	0	0
3	56	2	8	5075	277	0	77	0	4	0
4	5	4	171	39	5116	0	174	0	3	0
5	0	1	0	0	0	5388	0	105	1	12
6	493	2	199	66	274	0	4466	0	7	0
7	0	0	0	0	0	3	0	5460	0	25
8	6	0	3	2	5	0	6	14	5474	0
9	0	0	0	0	0	13	0	253	1	5227

Figure 3: Plot of training and validation loss and accuracy with convolutional neural network(left)

Confusion matrix for the test data with convolutional neural network (right)

The accuracy of our final model on the test data is 91.41%. Below is the confusion matrix for the test data.

	0	1	2	3	4	5	6	7	8	9
0	884	0	15	15	6	1	74	0	5	0
1	1	974	0	15	4	0	4	0	2	0
2	17	0	874	6	55	0	48	0	0	0
3	11	1	9	894	52	0	30	0	3	0
4	1	0	39	9	920	0	29	0	2	0
5	0	0	0	0	0	970	0	24	0	6
6	121	1	51	20	84	0	714	0	9	0
7	0	0	0	0	0	2	0	993	0	5
8	3	0	1	5	2	1	1	5	982	0
9	0	0	1	0	0	4	0	59	0	936

Figure 4: Confusion matrix for the test data with convolutional neural network

Compared to our model in part 1, the accuracy on the validation data has been improved. Our new model can better distinguish among 0, 2, 4 and 6, which represents t-shirt/top, pullover, coat, and shirt respectively.

Summary and Conclusions

As we can see, hyperparameters play an important role in the neural network and directly affect the performance. According to our results, we can also clearly see that our fully-connected network can hardly beat the 90% accuracy on the validation data, while convolution neural networks can easily do so. This is the reason why convolution neural networks are being applied ubiquitously for variety of learning problems, especially for image classification problems.

Appendix A. Python functions used and brief implementation explanation

- *tf.keras.datasets* loads the dataset
- *tf.keras.Model*. groups layers into an object with training and inference features
- *tf.keras.layers.Dense* regular densely-connected NN layer
- *tf.keras.layers.Conv2D* 2D convolution layer (e.g. spatial convolution over images)
- *tf.keras.layers.MaxPool2D* Max pooling operation for spatial data.
- *tf.keras.layers.Flatten* Flattens the input. Does not affect the batch size.
- *tf.keras.optimizers.Adam* Optimizer that implements the Adam algorithm.
- *predict_classes* generates class predictions for the input samples.
- *confusion_matrix* computes confusion matrix to evaluate the accuracy of a classification.[4]

4 Cited from https://www.tensorflow.org/

Appendix B. Python codes

```
### Part 1
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:]

from functools import partial

my_dense_layer=partial(tf.keras.layers.Dense,activation="relu",kernel_regularizer=tf.kera
s.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(64),
    my_dense_layer(20),
    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=10,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()

# create confusion matrix for the training data
y_pred=model.predict_classes(X_train)
conf_train=confusion_matrix(y_train,y_pred)

fig, ax = plt.subplots()

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

df = pd.DataFrame(conf_train)
ax.table(cellText=df.values, rowLabels=np.arange(10), colLabels=np.arange(10),
loc='center', cellLoc='center')
fig.tight_layout()

# accuracy on test data
model.evaluate(X_test,y_test)

# create confusion matrix for the test data
y_pred=model.predict_classes(X_test)
```

```
conf_test=confusion_matrix(y_test,y_pred)
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),
loc='center', cellLoc='center')
fig.tight_layout()



### Part 2
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_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, input_shape=(28,28,1)),
    my_conv_layer(32,3),
    tf.keras.layers.MaxPool2D(2,2),
    tf.keras.layers.MaxPool2D(2,2),
    tf.keras.layers.Flatten(),
    my_dense_layer(100),
    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=5, 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()
```

```
# create confusion matrix for the training data
y_pred = model.predict_classes(X_train)
conf_train = confusion_matrix(y_train, y_pred)
fig, ax = plt.subplots()

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

df = pd.DataFrame(conf_train)
ax.table(cellText=df.values, rowLabels=np.arange(10), colLabels=np.arange(10),
loc='center', cellLoc='center')
fig.tight_layout()

# accuracy on test data
model.evaluate(X_test,y_test)

# create confusion matrix for the test data
y_pred = model.predict_classes(X_test)
conf_test = confusion_matrix(y_test, y_pred)
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),
loc='center', cellLoc='center')
fig.tight_layout()
```