Neural Networks for Classifying Fashion MNIST

Khoa Vo

March 14, 2020

Abstract

This project will discuss and build Neuron Networks for classifying Fashion MNIST dataset. We will classify the $10,000$ images of fashion items into 10 different classes, with two type of Neuron Networks, which are Fully-connected and Convolutional. The accuracy rate will be reported to see which architecture of Neuron Networks work for Fashion MNIST dataset.

1 Introduction and Overview

We humans are astoundingly good at making sense of what our eyes are showing us. It is effortlessly for us to recognize what items are in front of us, but nearly all that work is done unconsciously, therefore, we usually do not appreciate how tough pattern recognition problems are.

The difficulty of this problem comes when we implement a computer program to classify the patterns. For example, we can easily recognize whether it is a t-shirt or a jumper by looking at the materials or the style of it. However, it is not that easy to express those "rules" algorithmically for a program to classify those two items. Computer needs different approaches than our brain to recognize those patterns. One of those is applying neural networks, in which, we train the system with examples called training set to infer rules for recognition.

This project will go through a process of building neural networks for classifying Fashion MNIST, which is to classify fashion items into 10 classes, including t-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag and ankle boot. We will investigate two network architectures, fully-connected and convolutional neural network, and report the accuracy rate on each architecture.

2 Theoretical Background

2.1 Perceptron

Perceptron is an artificial neutron, which weights the evidence to make decisions. It takes a several binary inputs, $x_1, x_2, ..., x_n$ and produces one binary output. We have weights, $w_1, w_2, ..., w_n$, according to the inputs. If the sum of the product of input and its according weight is greater than a defined threshold, the output is 1, otherwise, the output is 0. Based on threshold idea, in this paper, activation functions are used to determine the outputs.

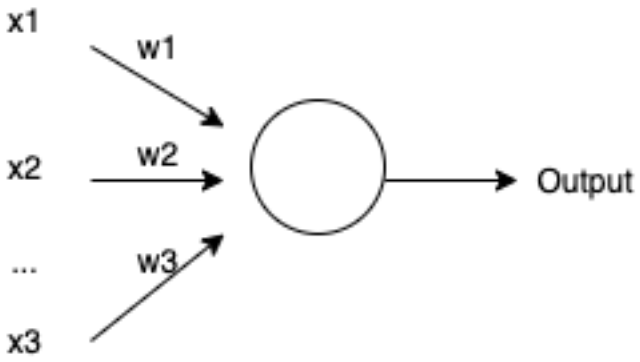


Figure 1: Caption

2.2 Neural network

Neural networks is a multi-layer perceptron and can have more than one output. There are many different architectures for building neural networks. In this project, we will experiment fully-connected and convolutional neural network.

2.2.1 Fully-connected Neural Network

Fully-connected neural network has each layer neuron connected to every neuron in the previous layer, and each connection has it's own weight. This is a totally general purpose connection pattern and makes no assumptions about the features in the data.

2.2.2 Convolutional Neural Network

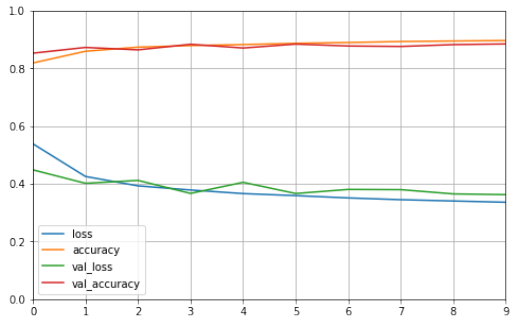
a convolutional layer each neuron is only connected to a few nearby (aka local) neurons in the previous layer, and the same set of weights (and local connection layout) is used for every neuron. This connection pattern only makes sense for cases where the data can be interpreted as spatial with the features to be extracted being spatially local (hence local connections only OK) and equally likely to occur at any input position (hence same weights at all positions OK) Convolutional neural network has layer neural only connected to nearby neurons in the previous layer, the same set of weights is used for each neuron.

3 Algorithm Implementation and Development

In this project, we will use Keras, a high-level neural networks library which is running on the top of TensorFlow.

3.1 Data Processing

After the load fashion_mnist data, we have $60,000$ samples for training and $10,000$ for testing. Besides, training set and testing set, we also need a validation set to estimate of model skill while tuning model's hyperparameters, such learning rate, depth width, regulazation parameters, activative functions, or optimizer. We will use the first $5,000$ samples of training for validation set, and the rest for training set. We also need to normalize our data by dividing all the data by 255.0.



(a) The loss and accuracy rate for training and validation sets

	0	1	2	3	4	5	6	7	8	9
0	826	1	11	51	5	1	92	0	13	0
1	1	937	1	50	5	0	6	0	0	0
2	20	0	823	18	73	0	65	0	1	0
3	16	2	12	912	32	0	22	0	4	0
4	0	0	150	33	768	0	47	0	2	0
5	0	0	0	0	0	948	0	34	1	17
6	136	0	102	41	70	0	639	0	12	0
7	0	0	0	0	0	14	0	934	0	52
8	3	0	6	5	4	2	1	6	973	0
9	0	0	0	0	0	8	1	28	0	963

(b) Confusion Matrix

Figure 2: Results for 1 layer - width of 128 Fully-connected Neuron Network

3.2 Part I: Fully-connected neuron network

In this part, we will experiment the fully-connected neuron network with only one layer, multiple layers with the same width, and the pyramid-like structure of layers.

After having the data processed, we first need to flatten our data and then create layers by using "tf.keras.layers.Dense" with some arguments. I specify regularization parameters is 0.0001, the activation function for layers are 'relu', except the last layer is 'softmax', the optimization is 'adam', and the learning rate is 0.001. Finally, we use "tf.keras.Sequential" to stack a list of layers.

3.3 Part II: Convolutional neuron network

In this part, we will experiment the convolutional neuron network with only one layer and multiple layers.

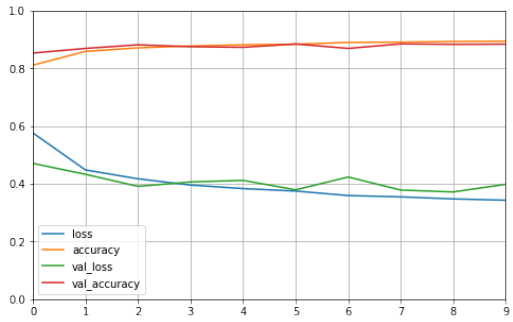
The process is quitesimilar to the previous part, but in order to use the convolutional layers in Tensorflow we need to add new axis, which makes our data become 55000x28x28x1 instead of 55000x28x28. We use "tf.keras.layers.Conv2D" to create a convolution kernel that is convolved with the layer input to produce a tensor of outputs. We need to provide the keyword argument 'input_shape' for first layer of the model. Max pooling layer is used for this part. All other parameter is similar to previous part.

4 Computational Results

4.1 Part I

4.1.1 One Layer Fully-connected NN

For 1 layer Fully-connected Neuron Network with the width of 128, we got the accuracy of 88.48% on validation data and 87.48% on testing data. Figure 2.a indicates that there is no overfitting in this model since the loss and validation rate is quite similar. Figure 2.b shows that the program failed to classify t-shirt, pullover, and coat the most. It makes sense since those items have some similar features.

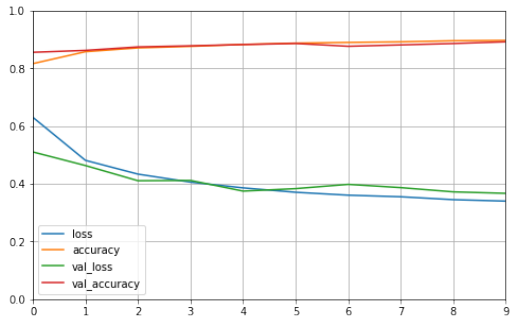


(a) The loss and accuracy rate for training and validation sets

	0	1	2	3	4	5	6	7	8	9
0	794	1	7	109	3	1	78	0	7	0
1	0	963	3	27	2	1	2	0	2	0
2	16	0	747	27	122	2	83	1	2	0
3	6	5	7	952	8	1	16	0	5	0
4	0	0	80	69	805	0	43	0	3	0
5	0	0	0	1	0	957	0	31	1	10
6	113	2	68	82	76	0	643	0	16	0
7	0	0	0	0	0	9	0	955	0	36
8	2	0	4	7	5	1	0	5	975	1
9	0	0	0	0	0	12	0	27	1	960

(b) Confusion Matrix

Figure 3: Results for multiple-layers with the same width of 128 Fully-connected Neuron Network



(a) The loss and accuracy rate for training and validation sets

	0	1	2	3	4	5	6	7	8	9
0	856	2	6	18	4	2	108	0	4	0
1	3	973	0	17	3	0	4	0	0	0
2	18	0	759	13	124	0	86	0	0	0
3	28	11	5	870	48	0	33	0	5	0
4	1	0	68	21	844	0	66	0	0	0
5	0	0	0	0	0	941	0	28	0	31
6	141	2	76	21	71	0	681	0	8	0
7	0	0	0	0	0	8	0	935	0	57
8	11	0	1	4	6	8	13	4	953	0
9	0	0	0	0	0	3	1	20	0	976

(b) Confusion Matrix

Figure 4: Results for Pyramid-structure layers width of 324 128 48 Fully-connected Neuron Network

4.1.2 Multiple-layers with same width Fully-connected NN

For multiple-layers Fully-connected Neuron Network with same width of 128, we got the accuracy of 88.40% on validation data and 87.51% on testing data. The accuracy is quite similar to 1 layer neuron network. Figure 3.a indicates that there is no overfitting in this model since the loss and validation rate is quite similar. Figure 3.b shows that the program failed most on classifying t-shirt and pullover as shirt.

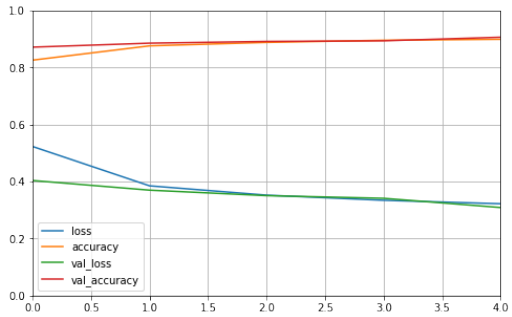
4.1.3 Pyramid-structure layers Fully-connected NN

For multiple-layers Fully-connected Neuron Network with same width of 128, we got the accuracy of 87.04% on validation data and 87.23% on testing data. The accuracy on validation data is a bit smaller than previous models, but the accuracy is quite similar. Figure 4.a indicates that there is no overfitting in this model since the loss and validation rate is quite similar. Figure 4.b shows that this model failed to classify between shirt and t-shirt the most.

4.2 Part II

4.2.1 One Layer Convolutional NN

For one layer Convolutional Neuron Network, we got the accuracy of 90.64% on validation data and 89.55% on testing data. The accuracy rate is higher than all model of fully-connected

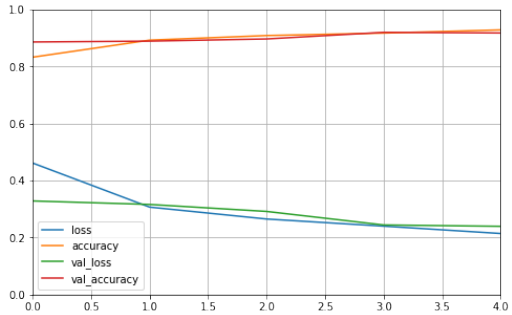


(a) The loss and accuracy rate for training and validation sets

	0	1	2	3	4	5	6	7	8	9
0	826	0	13	24	5	1	117	0	14	0
1	2	974	0	16	4	0	1	0	3	0
2	16	1	838	10	67	0	63	0	5	0
3	16	6	8	918	22	0	24	0	6	0
4	2	1	78	34	826	0	54	0	5	0
5	0	0	0	1	0	981	0	11	1	6
6	99	3	75	32	60	0	708	0	23	0
7	0	0	0	0	0	32	0	947	0	21
8	3	1	3	5	2	3	2	4	977	0
9	0	0	0	0	0	6	1	33	0	960

(b) Confusion Matrix

Figure 5: Results for one layers Convolutional Neuron Network



(a) The loss and accuracy rate for training and validation sets

	0	1	2	3	4	5	6	7	8	9
0	818	0	11	22	6	2	135	0	6	0
1	2	983	0	10	2	0	2	0	1	0
2	17	1	887	10	40	0	45	0	0	0
3	11	3	8	943	16	0	18	0	1	0
4	1	0	48	40	838	0	73	0	0	0
5	0	0	0	0	0	977	0	9	0	14
6	75	1	62	32	58	0	768	0	4	0
7	0	0	0	0	0	6	0	956	0	38
8	5	0	2	3	2	2	1	4	980	1
9	0	0	0	0	0	5	0	22	0	973

(b) Confusion Matrix

Figure 6: Results for multiple layers Convolutional Neuron Network

neuron networks we did in the previous part. Figure 5.a indicates that there is no overfitting in this model since the loss and validation rate is quite similar. Figure 5.b shows that this model failed to classify between shirt and t-shirt the most.

4.2.2 Multiple-Layers Convolutional NN

For multiple layers Convolutional Neuron Network, we got the accuracy of 91.78% on validation data and 91.23% on testing data. This is the model that has the highest accuracy. Figure 6.a indicates that there is no overfitting in this model since the loss and validation rate is quite similar. Figure 6.b shows that this model failed to classify between shirt and t-shirt the most.

5 Summary and Conclusions

Neuron Networks can be applied to do recognition problems. In this project, we go through the process of building Neuron Networks and experiment with different architectures. We can see that Convolutional Neuron Networks perform better than Fully-connected Neuron Network. One factor that could affect the result is how we choose other parameters in model, such as optimizer, activation functions, etc. If we can collect more data for training, it will potentially increase the accuracy rate for classification.

Appendix A

tf.keras.datasets.fashion_mnist: Loads the Fashion-MNIST dataset.
tf.keras.models.Sequential: Linear stack of layers.
tf.keras.layers.Flatten: Flatten input data.
tf.keras.layers.Dense: Just your regular densely-connected NN layer.
tf.keras.layers.MaxPooling2D: Max pooling operation for spatial data.
np.newaxis: Expand the dimensions of the resulting selection by one unit-length dimension.
model.predict_classes: Predict classes' value.
model.compile: Configures the model for training.
model.fit: Trains the model for a fixed number of epochs.
confusion_matrix: return the confusion matrix.

Appendix B

Part I

```
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:]

    my_dense_layer = partial(tf.keras.layers.Dense, a="relu",
        kernel_regularizer=tf.keras.regularizers.l2(0.00011))

    model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    my_dense_layer(128),
    my_dense_layer(128),
    my_dense_layer(128),
    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=10,
        validation_data=(X_valid,y_valid))
    model.evaluate(X_test,y_test)
    y_pred = model.predict_classes(X_test)
    conf_test = confusion_matrix(y_test, y_pred)
    print(conf_test)
    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_mat.pdf')
```

Part II

```
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,padding="same",input_shape=[28,28,1]),
    tf.keras.layers.MaxPooling2D(2),
    my_conv_layer(64,3),
    tf.keras.layers.MaxPooling2D(2),
    my_conv_layer(128,3),
    tf.keras.layers.Flatten(),
    my_dense_layer(64),
    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))
model.evaluate(X_test,y_test)
y_pred = model.predict_classes(X_test)
conf_test = confusion_matrix(y_test, y_pred)
print(conf_test)
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_mat.pdf')
```