Neural Networks

Yuanheng (Jonathan) Zhao

March 12nd 2020

Abstract

This report will explore the concepts of neural networks and convolutional neural networks. Based on the popular dataset, Fashion MNIST, we will try to build classifiers to classify 10 different categories of fashion items.

Sec I: Introduction and Overview

Nowadays, machine learning has become widely applied in conducting a variety of jobs of predictions, classifications, recognitions, detections, and recommendations. Some familiar applications may be face recognition technology, traffic predictions, and stock market prediction.

In this report, we will explore the neural network and convolutional neural network, and build models based on Tensorflow Keras in Python. The training data and testing data are from FASHION_MNIST, a popular set containing images of fashion items. We will try different parameters and structures of the models, and evaluate the results.

Sec II: Theoretical Background

2.1 Neural Network

A neural network is just a different model that we can use to relate all our x and y variables.

The basic components building up the neural network are called neurons, or nodes, and it will be intuitive to consider them with some biological circumstances. They are like the cells making up the brains of artificial intelligence. Neurons send out electrical signals through the synapses to other neurons. If a neuron receives enough electrical signals from other neurons, it will become “activated” and fire its own electrical impulse to others. Although a single neuron behaves in a relatively simple way, the collective behavior of all of the neurons can lead to very complex behavior, and this is true in both biological and machine learning fields.

In a neuron network, each neuron inputs and outputs a number and each connection has a weight associated with it. Say that the first layer of neurons are the input layer and they just output the same number they input. For the output of the neuron, we can calculate a weighted sum of the inputs as:

$$z = w_1x_1 + w_2x_2 + w_3x_3 \tag{2.1.1}$$

where x_1, x_2, and x_3 are signals and w_1, w_2, and w_3 are weights of connections.

And we can write this sum using vectors:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \quad w = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}, \quad z = w^T x = w_1x_1 + w_2x_2 + w_3x_3. \tag{2.1.2}$$

Then we will say that the output neuron is active if this sum is greater than some threshold. The simplest activation function is:

$$\sigma(z) = \begin{cases} 0, & z < \text{ threshold} \\ 1, & z \geq t \text{ hreshold} \end{cases} \tag{2.1.3}$$

To account for shifting the threshold, we can add an extra neuron to the input layer called a bias neuron. In terms of the formula this amounts to adding a constant term b.

$$y = \sigma(w_1x_1 + w_2x_2 + w_3x_3 + b) \tag{2.1.4}$$

The multilayer perceptron typically paired with a sigmoid or ReLU activation function, is what we would refer to as a feed-forward neural network because everything flows in one direction from the inputs to the outputs. For a given layer, every neuron is connected to every neuron in the previous layer. We call those types of layers fully-connected or dense. When all of the layers are of that type, we say that it is a fully-connected network. The number of neurons in each layer is called the width of the layer. The number of layers is called the depth of the network. You may hear the terminology of deep learning or deep neural networks. This just refers to neural networks with many layers.

The process of finding the weights and biases that minimize the loss function is known as training the network. One of the biggest problems with neural networks is overfitting. Overfitting tends to occur when you have a lot of parameters in comparison to the number of data points you have. For neural networks, every weight and bias is a parameter. So if we see that our neural network is performing much better on training data than test data, we can try to make some adjustments to your network (such as changing the depth of the network or the width of your layers). Essentially, we are making adjustments to the model. The standard procedure is to split our data into three parts: training data, validation data, and test data. Validation data plays the role of test data while we are trying to adjust aspects of your network and choose the right model. It can also help us figure out when to stop training our network. The test data is used at the very end to determine the final performance of our model.

2.2 Convolutional Neural Network

Convolutional neural networks have become one of the most influential innovations in the field of computer vision. As a basis model of deep learning, it incorporates multiple number of hidden layers which use convolution in place of general matrix multiplication in at least one of them.

The basic structure of a convolutional neural network (CNN), is built up by convolutionary and pooling layers, fully connected layers, and output layers. We will talk about some common concepts happening in the flow of work. Image a light sliding over and scanning all the areas of an input image. We consider the light as a filter and the areas scanned by the filter the receptive fields. An receptive field incorporates the information of the activity of the neurons changes depending on the orientation of the light. We use the same filter to calculate all of the neurons in a layer and we call the layer of neurons the feature map. For a single convolutional layer we have multiple feature maps. Each one has its own filter. But we consider all of them to be one convolutional layer. Stride is the distance to slide the receptive field. If we increase stride then the size of the feature map is decreased. Pooling layers are referred to as a downsampling layer. It is also referred to as a downsampling, or subsampling layer, with max pooling and average pooling options. One common advantage of convolutional layers is that it can capture the same features even if the image is shifted (in other words, the pixels are changed a lot but the features as a whole do not change so much).

LeNet-5, as one of the most famous CNNs, is constituted by convolutions, subsampling, convolutions, subsampling, full connections, full connections, and Gaussian connections. Other popular CNNs include AlexNet, GoogleNet, and VGGNet.

Sec III: Algorithm Implementation & Development

In the following, we will develop our models in Python, using the package Keras in Tensorflow.

First of all, we want to load the data from the popular Fashion-MNIST. By using the built-in method load_data in the package tf.keras.datasets.fashion_mnist, we load 60000 images of fashion items (28x28 pixels) into the training set, and 10000 images into the testing set. Then we split the training set into two portions, one with 55000 training examples in an array X_train and one with 5000 validation examples in an array X_valid. Since the numbers representing the images in the training and testing arrays are integers from 0 to 255, we convert them to floating point numbers between 0 and 1 by dividing each by 255.0.

We will use the standard ReLU activation function by setting the activation parameter as “relu” in fully-connected layers. And we will create a regularizer that applies an L2 regularization penalty, which is computed as the sum of the squared value of the coefficients.Then we build the model with a Sequential pipeline for both parts of our experiment.

For Part I, we build the Sequential model by adding the following layers:

1) Flatten layer (input shape = 28x28)
2) Fully-connected layer (Dense layer) with 300 nodes
3) Fully-connected layer (Dense layer) with 100 nodes
4) Output layer with 10 classes

For Part II, we adjust the model by adding and removing some layers as the following:

1) Convolutional layer with 256 filters, kernel size = 3x3, and zero padding
2) MaxPooling layer with pool size = 2x2
3) Convolutional layer with 128 filters, kernel size = 3x3, and no padding
4) MaxPooling layer with pool size = 2x2
5) Flatten layer
6) Fully-connected layer (Dense layer) with 128 nodes
7) Output layer with 10 classes

We configure the model for training by setting parameters in the function compile. We will apply sparse_categorical_crossentropy as our loss function, or objective function. We make the learning rate as 0.0001 for part I and 0.001 for part II.

When training finished, we will test the model by our testing set and print out the confusion matrix (error matrix), which indicates the performance of the algorithm.

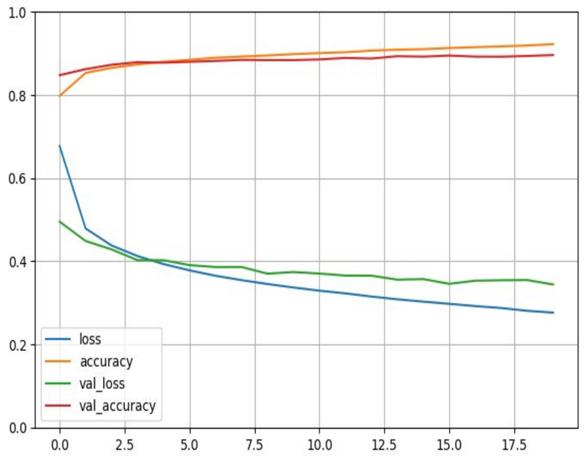
Sec IV: Computational Results & Conclusion

In part I, we first set 20 epochs and run the model. The stats we finally get are:

loss: 0.2796 - accuracy: 0.9199 - val_loss: 0.3562 - val_accuracy: 0.8876 (for training set)
loss: 0.3942 - accuracy: 0.8771 (for testing set)

And then we set 50 epochs and train the model. The stats we finally get are:

loss: 0.1865 - accuracy: 0.9575 - val_loss: 0.3563 - val_accuracy: 0.8970 (for training set)
loss: 0.3921 - accuracy: 0.8896 (for testing set)



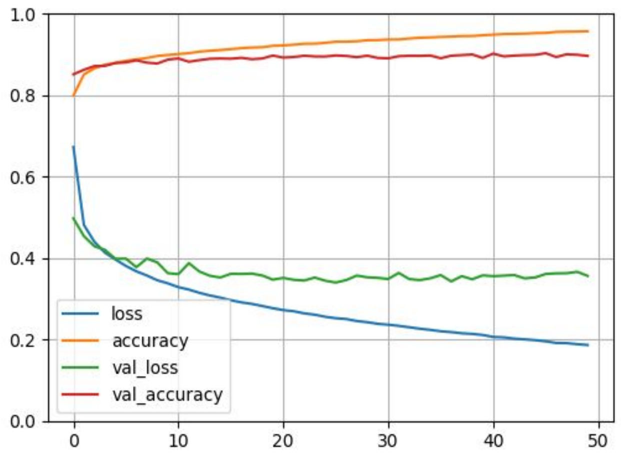


Figure 1: Part I, accuracy & loss during 20 epochs   *Figure 2: Part I, accuracy & loss during 50 epochs*

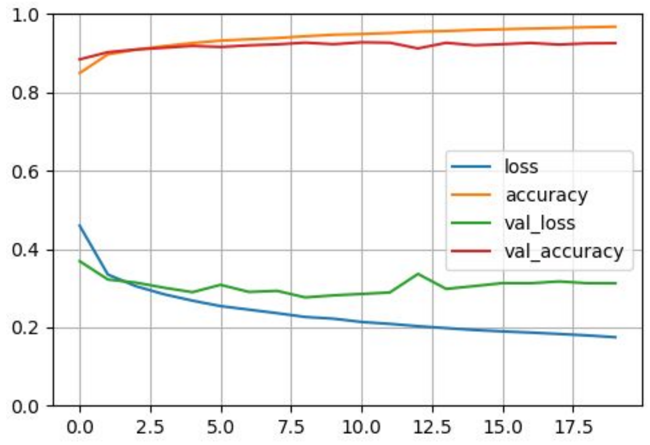


Figure 3: Part II, accuracy and loss during 20 epochs

From figures 1 and 2, we can see that the accuracy of the training set is always increasing. The accuracy for training data even approaches to 95.75 percent at the 50th epochs. However, the accuracy for the validation data and for the testing data is not increasing as that much. We finally get an accuracy of about 89% for the testing data.

In part II, we apply the CNN model and set 20 epochs. (It takes a really long time...actually not that long). The stats we get for the testing set is: **loss: 0.3387 - accuracy: 0.9212**. We notice that the accuracy for training data is increasing to 98% at the 20th epoch, while the accuracy for validation data seems to remain the same at latter epochs, and the loss of validation data even increases. We think this is caused by the overfitting and bias of our model on the training data set, even if we have assigned the regularizers.

	0	1	2	3	4	5	6	7	8	9
0	862	1	15	34	4	0	80	0	4	0
1	3	963	0	29	2	0	2	0	1	0
2	14	0	834	19	68	1	64	0	0	0
3	20	2	11	934	14	0	17	0	2	0
4	1	0	95	58	757	0	88	0	1	0
5	0	0	0	1	0	964	0	19	1	15
6	126	0	86	41	43	0	696	0	8	0
7	0	0	0	0	0	16	0	957	0	27
8	6	1	5	9	3	3	10	4	959	0
9	0	0	0	0	0	5	1	35	0	959

Figure 4: part I - confusion matrix at the end of 20 epochs

	0	1	2	3	4	5	6	7	8	9
0	808	0	10	21	8	1	146	0	6	0
1	2	977	2	12	4	0	1	0	2	0
2	14	0	843	10	60	0	72	0	1	0
3	8	3	6	929	34	0	17	0	3	0
4	0	0	33	12	930	0	25	0	0	0
5	0	0	0	0	0	992	0	6	0	2
6	69	0	38	24	81	0	779	0	9	0
7	0	0	0	0	0	14	0	957	0	29
8	3	1	0	2	3	2	5	0	983	1
9	0	0	0	0	0	7	0	18	0	975

Figure 5: part II - confusion matrix at the end of 20 epochs

From the two confusion matrices we can see that the models made more errors on predicting the category 6, and the category 6 is mainly confused with category 0 and category 2. Checking the labels, we find that the category 6, 0, 2 are respectively shirt, T-shirt/top, and pullover, which are three categories might be incorrectly classified.

Appendix A: Python methods used

1) tf.keras.models.Sequential: Linear stack of layers.
2) tf.keras.models.Sequential.compile: Configures the model for training.
3) tf.keras.models.Sequential.predict_classes: Generate class predictions for the input samples. The input samples are processed batch by batch.
4) tf.keras.layers.Dense: create a regular densely-connected NN layer.
5) tf.keras.layers.Conv2D: 2D convolution layer (e.g. spatial convolution over images). This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs.
6) tf.keras.layers.MaxPooling2D: Max pooling operation for spatial data.

Appendix B: Python code

FASHION_MNIST_Classifier

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


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

# 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()

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:]

# Part 1
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(100),
   my_dense_layer(10, activation="softmax")  # 10 classes to be classified
])
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=50, 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()

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

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 a 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')
```

FASHION_MNIST_CNN_Classifier

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

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]


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",
kernel_regularizer=tf.keras.regularizers.l2(0.0001))

model = tf.keras.models.Sequential([
   # do zero padding, so that the size of the feature map is the same as the size of the input
   my_conv_layer(256, 3, padding="same", input_shape=[28, 28, 1]),
   tf.keras.layers.MaxPooling2D(2),
   my_conv_layer(128, 3),
   tf.keras.layers.MaxPooling2D(2),
   tf.keras.layers.Flatten(),  # Always remember to flatten before processing into fully connected
layers
   my_dense_layer(128),
   my_dense_layer(10, activation="softmax")  # 10 classes to be classified
])
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=20, 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()

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

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 a 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')
```