Neural Networks for Classifying Fashion MNIST

Chenghao Chen

Amath 482 HW 5

Abstract

In the project, we are dealing with an analogous data set called Fashion-MNIST, which contains 10 different classes of fashion items. The whole project has two parts. The first part is to train a fully connected neural network to classify the images of the Fashion-MNIST. The second part is to use a convolutional neural network. We are required to test different parameters and adjust different optimizer in each part. Finally, we need to find a final model, which give us the highest accuracy in the end.

Introduction

In the homework, the analogous data set, Fashion-MNIST, contains 10 different classes of fashion items, which labels T-shirt/top as 0, Trouser as 1, Pullover as 2, dress as 3, coat as 4, sandal as 5, shirt as 6, sneaker as 7, bag as 8 and ankle boot as 9. It has a total of 60000 training images and 10000 test images. We also use first 5000 images as validation data. In the first part, we want to adjust the depth of the network, the width of the layers, the learning rate, the regularization parameters, the activation functions and the optimizer. In the second part, we want to test the number of filters, the filter sizes, the stride, the padding options and the pool sizes. Finally, we will compare each parameter and conclude the project.

Theoretical Background

Neural Network

Artificial neural networks are a machine learning tool used to perform tasks such as classification, regression, and dimensionality reduction. In ANN implementations, the "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

$$y = \sigma(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b) \quad (1)$$

Fully connected

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.

Convolutional

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product. The activation function is commonly a

RELU layer, and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution.

Activation function

In artificial neural networks, the activation function of a node defines the output of that node given an input or set of inputs. A standard integrated circuit can be seen as a digital network of activation functions that can be "ON" (1) or "OFF" (0), depending on input. This is similar to the behavior of the linear perceptron in neural networks. However, only nonlinear activation functions allow such networks to compute nontrivial problems using only a small number of nodes.

Algorithm Implementation

First, I load the data from Fashion-MNIST and convert the image to double precision using the command im2double. Then, I reshape and reorder the X_train and X_test so that they have dimensions 60000*28*28*1 and 10000*28*28*1. Finally, I will remove 5000 images from the training data to use as validation data, which leave us 55000 training examples and 5000 validation examples.

Second, I will use a fully connected neural network in part one. In the first layer, I use **imageInputLayer** with the size of the data. Then, I use four **FullyConnectedLayer** with the width of 784, 50, 10 and 10. In each layer, I will use the activation function **LeakyReluLayer** to connect the layers. At last, I use **softmaxLayer** and **classificationLayer** at the end. Then, I use **traningOptions** to change the options. I use optimizer **adam**, number of epochs as 8, initial learn rate as 1e-3, regularization parameter as 1e-5, validation data, verbose as false and plot the result. Then, I plot two confusion matrices, one for test data and one for traning data.

Third, I will use a convolutional neural network in part two. Still, I need to create layers using **convolution2dLayer** and **fullyConnectedLayer.** Also, I need to use activation function between each layer. Then, I still use option to test different options and plot the confusion matrices.

In my code, I change a lot of parameters to test impacts of different parameters. I will report those effects next.

Computational Result

Part 1

(a) The depth of the network

While I keep all other parameters same, I test 2 layers, 3 layers and 4 layers. As we can see from the result, the decreasing number of layers can improve the accuracy a little bit. Also, the operation time will decrease. Increasing number of layers will also cause over fitting problem.

Results	
Validation accuracy:	88.34%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 15:55:01
Elapsed time:	1 min 6 sec

Figure (1): 4 layers accuracy

Results	
Validation accuracy:	88.84%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 15:57:17
Elapsed time:	1 min 2 sec

Figure (2): 3 layers accuracy

Results	
Validation accuracy:	89.52%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 15:59:16
Elapsed time:	58 sec

Figure (3): 2 layers accuracy

(b) The width of layers

I adjust the second layers width from 50, 100 and 200. As we can see from the result, improving the width of the layers will decrease the accuracy and also slow down the operation. Also, large width of layers will cause overfitting problem. Therefore, the width with 50 is the best.

Results	
Validation accuracy:	88.84%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 15:57:17
Elapsed time:	1 min 2 sec

Figure (4): width 50

Results	
Validation accuracy:	88.50%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 16:09:49
Elapsed time:	1 min 6 sec

Figure (5): width 100

Results	
Validation accuracy:	88.02%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 16:11:27
Elapsed time:	1 min 7 sec

Figure (6): width 200

(c) Learning rate

I adjust learning rate from 1e-3, 1e-4, and 1e-2. As we can see, the learning rate cannot be too large or too small. If the learning rate is too big, overfitting problem will happen and decrease the accuracy. If the learning rate is too small, it is hard for epoch to converge and decrease the accuracy.

Results	
Validation accuracy:	88.84%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 15:57:17
Elapsed time:	1 min 2 sec

Figure (7): learning rate 1e-3

Results	
Validation accuracy:	88.42%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 16:18:00
Elapsed time:	1 min 2 sec

Figure (8): learning rate 1e-4

Results	
Validation accuracy:	87.86%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 16:19:36
Elapsed time:	1 min 1 sec

Figure (9): learning rate 1e-2

(d) Regularization parameters

I adjust the regularization from 1e-2, 1e-5, and 1e-10. As we can see, the regularization rate cannot be too large or small. If it is too small, the overfitting problem is serious. If it is too high, the accuracy is really low. So 1e-5 is the best.

Results	
Validation accuracy:	88.84%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 15:57:17
Elapsed time:	1 min 2 sec

Figure (10): regularization rate 1e-5

Results	
Validation accuracy:	84.22%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 16:24:20
Elapsed time:	1 min 3 sec

Figure (11): regularization rate 1e-2

Results	
Validation accuracy:	88.72%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 16:25:55
Elapsed time:	1 min 4 sec

Figure (12): regularization rate 1e-10

(e) Activation function

I choose two functions, ReLU and leakyReLu. The accuracy for relu is about 88.3% and the leakyRelu is 88.84%. So the leakyRelu is better.

(f) Optimizers

I choose ‘adam’ and ‘rmsprop’. So the accuracy does not have too much difference. Rmsprop works better in my computer.

Results	
Validation accuracy:	88.84%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 15:57:17
Elapsed time:	1 min 2 sec

Figure (13): adam

Results	
Validation accuracy:	88.94%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 16:36:37
Elapsed time:	58 sec

Figure (14): rmsprop

(g) Maxepochs

I choose maxepochs as 5,7, and 8. I found out that as I increase the number of epochs, the accuracy will increase, but the operation time will increase a lot.

Part two

(a) The number of filters

I choose 10 and 20. I don't see a much difference between the number of filters, but it takes a much longer time to operate. So, I suggest 10 filters.

Results	
Validation accuracy:	90.94%
Training finished:	Reached final iteration
Training Time	
Start time:	12-Mar-2020 21:55:53
Elapsed time:	4 min 54 sec

Figure (15): 10 filters

Results	
Validation accuracy:	90.82%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 16:48:56
Elapsed time:	7 min 7 sec

Figure (16): 20 filters

(b) Filter size

I choose [5 5] and [8 8]. There is not much a difference, but smaller filter size may result a more accurate number.

Results	
Validation accuracy:	90.94%
Training finished:	Reached final iteration
Training Time	
Start time:	12-Mar-2020 21:55:53
Elapsed time:	4 min 54 sec

Figure (17): [8 8]

Results	
Validation accuracy:	91.20%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 17:16:41
Elapsed time:	4 min 10 sec

Figure (18): [5 5]

(c) Stride

I choose 1 and 5. As we can see, decreasing stride number will increase the accuracy

Results	
Validation accuracy:	90.94%
Training finished:	Reached final iteration
Training Time	
Start time:	12-Mar-2020 21:55:53
Elapsed time:	4 min 54 sec

Figure (19): stride 1

Results	
Validation accuracy:	79.94%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 17:23:05
Elapsed time:	56 sec

Figure (20): stride 5

(d) Padding

I think using all the same padding will result the same.

(e) Pool size

I choose [5 5] and [10 10]. As we can see, smaller pool size have better accuracy.

Results	
Validation accuracy:	90.94%
Training finished:	Reached final iteration
Training Time	
Start time:	12-Mar-2020 21:55:53
Elapsed time:	4 min 54 sec

Figure (21): [5 5] pool size

Results	
Validation accuracy:	89.52%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Mar-2020 17:24:42
Elapsed time:	5 min 31 sec

Figure (22): [10 10] pool size

Best case for part 1

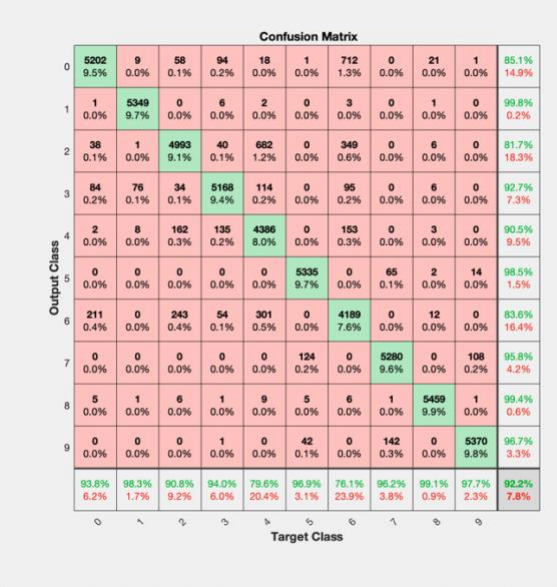

Figure (23): confusion matrix for validation

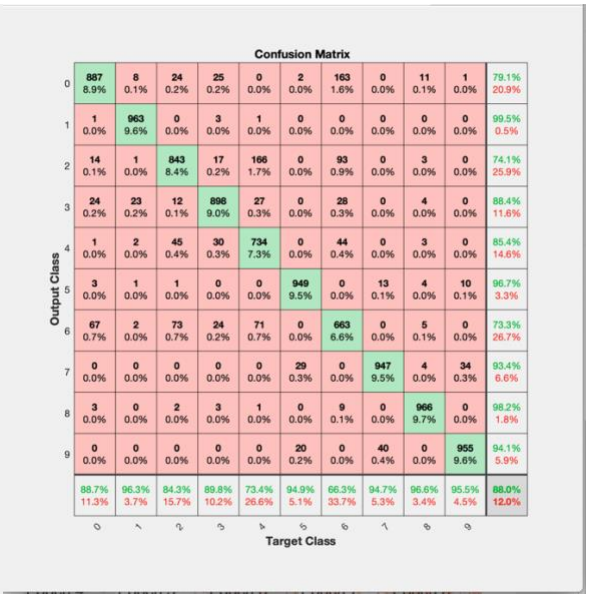

Figure (24): confusion matrix for test

My best case for part 1 is that the validation accuracy is 89.34% and my test accuracy is 88%. I have three fully connected layer with width 784, 50, and 10. The MaxEpochs is 8. Initial learn rate is 1e-3. The regularization is 1e-5.

Best case for part two

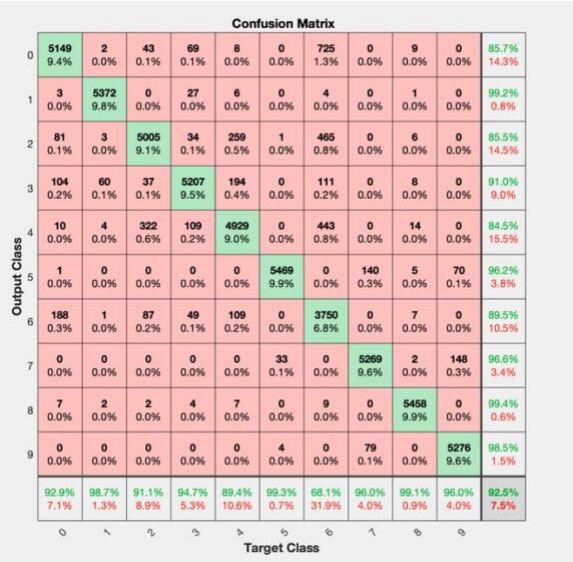

Figure (25): confusion matrix for validation

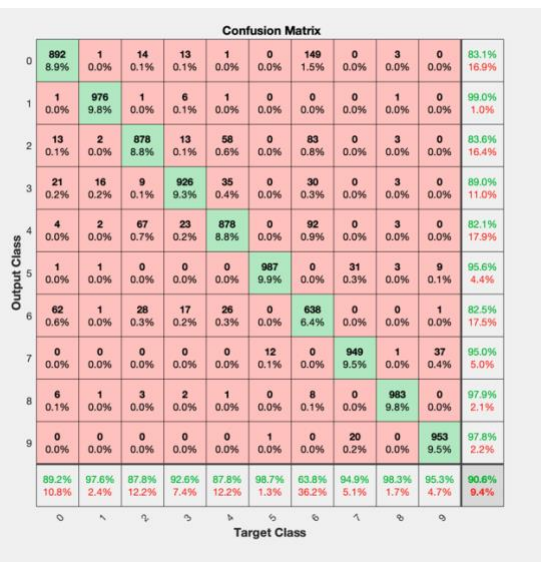

Figure (26): confusion matrix for test

My best case for part 2 is that the validation accuracy is 91.20% and my test accuracy is 90.6%. I have 2 convolution2dlayer as [5 5] ,10, stride 1 and [8 8] ,16, stride 1. I have two fully connected layer with width250 and 10. The MaxEpochs is 7. Initial learn rate is 1e-3. The regularization is 1e-5.

Conclusion

In this project, we learn the fully connected neural network and convolution neural network. As we compare two networks, the convolution is better than fully connected neural network. Also, we test a lot of different parameters in the project, which help us understand how the neural network function. In conclusion, the neural network is a really efficient tool for machine learning.

Citation

[1] “Makers of MATLAB and Simulink,” MathWorks. [Online]. Available: https://www.mathworks.com/. [Accessed: 25-Jan-2020].

Appendix A

reluLayer creates a ReLU layer.[1]
leakyReluLayer returns a leaky ReLU layer.[1]
tanhLayer creates a hyperbolic tangent layer.[1]
softmaxLayer creates a softmax layer.[1]
classificationLayer creates a classification layer.[1]
im2double(I) converts the image I to double precision. [1]
permute(A,dimorder) rearranges the dimensions of an array in the order specified by the vector dimorder.[1]
categorical(A) creates a categorical array from the array A. The categories of B are the sorted unique values from A.[1]
imageInputLayer(inputSize) returns an image input layer and specifies the InputSize property.[1]
fullyConnectedLayer(outputSize) returns a fully connected layer and specifies the OutputSize property.[1]
trainingOptions(solverName,Name,Value) returns training options with additional options specified by one or more name-value pair arguments.[1]
trainNetwork(X,Y,layers,options) trains a network for image classification and regression problems. [1]
plotconfusion(targets,outputs) plots a confusion matrix for the true labels targets and predicted labels outputs. [1]
classify(net,X) predicts class labels for the image data in X using the trained network, net.[1]
convolution2dLayer(filterSize,numFilters) creates a 2-D convolutional layer and sets the FilterSize and NumFiltersproperties.[1]
averagePooling2dLayer(poolSize) creates an average pooling layer and sets the PoolSize property. [1]

Appendix B

```
clear; close all; clc

load('fashion_mnist.mat')

X_train = im2double(X_train);
X_test = im2double(X_test);

X_train = reshape(X_train,[60000 28 28 1]);
X_train = permute(X_train,[2 3 4 1]);

X_test = reshape(X_test,[10000 28 28 1]);
X_test = permute(X_test,[2 3 4 1]);

X_valid = X_train(:,:,:,1:5000);
X_train = X_train(:,:,:,5001:end);

y_valid = categorical(y_train(1:5000))';
y_train = categorical(y_train(5001:end))';
y_test = categorical(y_test)';
%% fully
layers = [imageInputLayer([28 28 1])
        fullyConnectedLayer(784)
        leakyReluLayer
        fullyConnectedLayer(50)
        leakyReluLayer
        %fullyConnectedLayer(10)
        %leakyReluLayer
        fullyConnectedLayer(10)
        softmaxLayer
        classificationLayer];

options = trainingOptions('adam', ...
    'MaxEpochs',8,...
    'InitialLearnRate',1e-3, ...
    'L2Regularization',1e-5, ...
    'ValidationData',{X_valid,y_valid}, ...
    'Verbose',false, ...
    'Plots','training-progress');

net = trainNetwork(X_train,y_train,layers,options);

figure(2)
y_pred = classify(net,X_train);
plotconfusion(y_train,y_pred)

figure(3)
y_pred = classify(net,X_test);
plotconfusion(y_test,y_pred)

%% convolution
layers2 = [
    imageInputLayer([28 28 1])
```

```
    convolution2dLayer([5 5],10,"Padding","same","Stride",1)
    leakyReluLayer
    averagePooling2dLayer([5 5],"Padding","same","Stride",[2 2])
    convolution2dLayer([8 8],16,"Padding","same","Stride",1)
    leakyReluLayer
    fullyConnectedLayer(250)
    leakyReluLayer
    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer()];

options2 = trainingOptions('adam', ...
    'MaxEpochs',7,...
    'InitialLearnRate',1e-3, ...
    'L2Regularization',1e-5, ...
    'ValidationData',{X_valid,y_valid}, ...
    'Verbose',false, ...
    'Shuffle','every-epoch',...
    'Plots','training-progress');

net2 = trainNetwork(X_train,y_train,layers2,options2);

figure(1)
y_pred = classify(net2,X_test);
plotconfusion(y_test,y_pred)

figure(2)
y_pred = classify(net2,X_train);
plotconfusion(y_train,y_pred)
```