Classification with Neural Networks

Yuxin Jiang
AMATH 482 HW5

Abstract

In this report, we will examine an analogous data set that includes 10 different classes of images of fashion items using neural networks. In part one, we will build a fully-connected neural network and train it for classification. We will input different hyper parameters such as number of layers, width of layers, and regularization parameters, and compare the validation accuracy. In part two, we will build a convolutional neural network and also try different parameters for. In the end, we will produce two final networks that achieve the best validation.

I. Introduction and Overview

During classification, neural networks are one of the most useful tools for machine learning, which offers us an organized and accurate prediction on data. I was given a data set called Fashion-MNIST, which includes 10 classes of fashion items, including t-shirt labeled as 0, trouser labeled as 1, pullover labeled as 2, dress labeled as 3, coat labeled as 4, sandal labeled as 5, shirt labeled as 6, sneaker labeled as 7, bag labeled as 8, and ankle boot labeled as 9. There are in total 60000 training images and 10000 testing images, and we will use the first 5000 training images as our validation data. We will train a fully-connected neural network in part one and a convolutional neural network in part two. By changing different parameters such as learning rate and filter sizes, I will compare the validation accuracy for the analysis on the influence of these parameters. I will also compare two different neural networks and explore the usefulness.

II. Theoretical Background

A. Neural Network

Artificial Neural Networks are a machine learning tool used to perform tasks such as classification, regression, and dimension reduction. In 1957, Frank Rosenblatt invented a framework that introduces the concept 'perceptron'. A perceptron is made up of threshold logic units or linear threshold units. Each neuron inputs and outputs a number and each connection has a weight associated with it. Then we will say that the output neuron is active if this sum is greater than some threshold and inactive if the sum is less than some threshold. Plugging the weighted sum into an activation function, we will also add an extra neuron to the input layer called bias neuron which always outputs as 1 to account for shifting in threshold. The process can be expressed as following:

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

where σ is the activation function, y is the output, x is the input, w is the weight, and b is the bias neuron. Multilayer Perceptron (MLP) will offer us a better solution for complex problems. There are input layer and output layer, and any layers between them are called hidden layers.

A fully-connected neural network is a network which has all layers made up of neurons connected to every neuron in the previous layer. The number of neurons in each layer is called the width of layer and the number of layers is called depth of the network. A convolutional

neural network is a network that includes convolutional layers which use a small window that slides across the layer and transform the data through an activation function. Pooling layers are always recommended to be used together with convolutional layer for better efficiency. I will be using average pooling in my experiment which takes the average value of the data in a convolutional window.

B. Activation Function

There are various types of activation functions that can be used. Hyperbolic tangent function can be written as

$$\sigma(x) = \tanh(x) \tag{2}$$

One of the most popular activation functions is rectified linear unit (ReLU), which is

$$\sigma(x) = \max(0, x) \tag{3}$$

I also used leaky ReLU, which is almost the same as the ReLU except it uses a relatively less steep function for negative data instead of 0. [1]

III. Algorithm Implementation and Development

I will use the built-in function included in the deep learning toolbox in MATLAB for building the neural network. I first loaded the data and used the command **Im2double** to convert the images to double precision. The vector started with x is the image data, and the vector started with y is the labeling data. I then reshaped and reordered the data with **reshape** and **permute**. I also removed 5000 sets of data from the training sets to construct validation data. I also changed the labeling data y to categorical array. The preprocessing of the data is the same for part one and part two.

In part one, I built a fully-connected neural network. I first built the input layer with **imageInputLayer** corresponding to the size of our dataset. I then built a fully connected layer with the command **fullyConnectedLayer**, also adding activation function between them. I used different activation functions with different commands, including **reluLayer** for ReLU, **leakyReluLayer** for leaky ReLu, and **tanhLayer** for hyperbolic tangent function. I also changed the parameters in the command **fullyConnectedLayer** to compare the influence of different width of the layer. In addition, I tried different number of layers by adding activation function and fully-connected layers between the input and output layer. I then built a classification layer in the end and used **softmaxLayer** before the final output layer. Then I built the training option with the command **trainingOptions**. I used different optimizer such as **adam** and **rmsprop**, and different parameters required in every optimizer, including number of epochs, learning rate, regularization parameters and shuffle options. I also plugged in the validation data. Since I did not wish to print the training results to the screen since I will plot the confusion matrix later, I set the 'Verbose' option to false. I then started training using the command **trainNetwork**. I produced predicted results using **classify** command to classify the data using our trained network. I plotted two confusion matrices, one for test data and one for training data.

In part two, I built a convolutional neural network. I repeated the steps except for the layer. I built an input layer as in part one, and then started with a convolutional layer using **convolution2dLayer**. After the activation function, I built an average pooling layer together with the second convolutional layer. I also changed several parameters such as strides, filter sizes,

[1] Gin, Craig, *Neural Networks Lecture Notes*

number of filters, padding options and pooling sizes. The first value in the array after **convolution2dLayer** is the size of filter, that is the same for **averagePooling2dLayer**, the second value is the number of filters. I trained the network and produced confusion matrices in the end.

IV. Computational Results

I tried changing several parameters without changing other parameters to test the influence on validation accuracy.

A. Part One: Fully-Connected Neural Network

a. Number of Layers

I first tried three, four hidden layers. The validation accuracy is 88.18% and 88.28%. I also tried five hidden layers and result is 87.42%, but there are minor overfitting problems since the training accuracy is much higher than validation accuracy. We can see that increasing number of layers could sometime improve the accuracy by a little, but too many layers could also cause slow operations and overfitting problem, thus decreasing the validation accuracy.

b. Widths of Layers

I changed the number of neurons in the second layer to 100, 200 and 1000 and the results are 88.28%, 88.38% and 88.32% with some overfitting problems. The width of layers is similar to the depth of layers. Increasing number of neurons could improve accuracy sometimes but better results are produced at a proper number of neurons instead of at a really large number.

c. Learning Rates

I changed the learning rate to 10^{-3}, 10^{-5} and 0.8 and the results are 88.48%, 84.04% and 10.02%. The learning rate is also in a similar situation. If the learning rate is set too small, it will require too many epochs to converge and thus become trapped in local minima, decreasing the accuracy sharply. Oppositely, if the learning rate is set too large, overshooting problems will happen and the accuracy decreased even more sharply. Therefore, a proper learning rate should be selected as well.

d. Regularization Parameters

As I set the L2 Regularization rate in 'adam' optimizer as 10^{-3}, 0.8, 10^{-5}, 10^{-10}, the results are 88.48%, 9.8%, 88.52% and 89.3%. L2 regularization rate is set to avoid overfitting problem, and thus if it is set too high, the accuracy rate will be bad. My results get better as it gets smaller, but it also takes much more time to run. Therefore, I chose a fair rate that will both produce results faster and maintain good accuracy rate.

e. Activation Functions

I used ReLU, leaky ReLU and hyperbolic tangent function by replacing all activation function between each fully-connected layer. The corresponding results are 88.72%, 88.9% and 88.32%. Different activation functions do not seem to cause too many differences in my case. But in some cases, specific activation functions should be used for better results.

f. Optimizers

I tried the optimizer called 'adam' and 'rmsprop' in MATLAB and corresponding results are 89.1% and 88.96%. The optimizer also did not produce too many differences. But in my case, 'adam' works better.

g. Number of Epochs

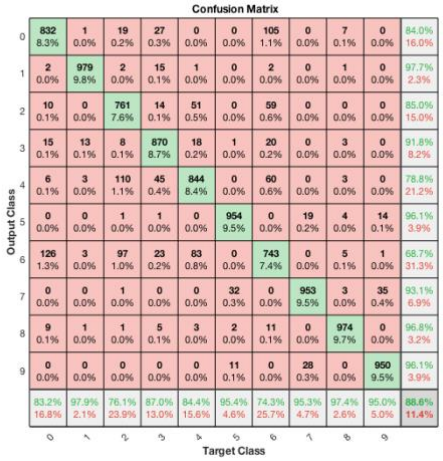

Figure (1): Confusion Matrix for Test Data for Fully-Connected Neural Network

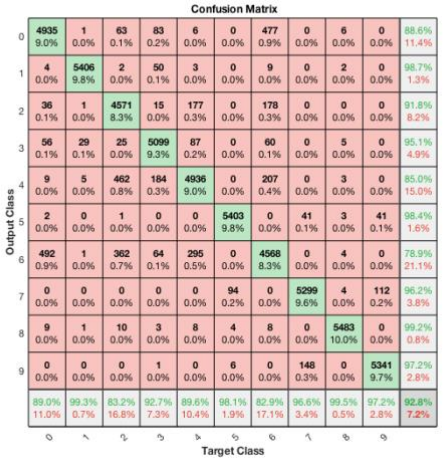

Figure (2): Confusion Matrix for Training Data for Fully-Connected Neural Network

I changed the **MaxEpochs** in the algorithm to 10, 7 and 5, and the results are 89.28%, 88.96% and 88.8%. As I increased the number of epochs, the results became better, but the time it took to run also increased.

h. Shuffle Options

I changed the shuffle options to once and every epoch, and the corresponding results are 88.44% and 89.1%. Therefore, when the training and validation data are shuffled in every epoch instead of only once in the beginning, the results also get better.

After adjusting the parameters according to my results, my best validation accuracy is 89.28% and test accuracy is 88.6%. My parameters are set as following: I set three hidden layers. The first fully-connected layer has 784 neurons, the second has 100 neurons and the last one has 10. I used leaky ReLU for all activation functions. I used 'adam' optimizer, with 7 epochs, learning rate as 10^{-3}, L2 regularization rate as 10^{-5} and shuffle in every epoch. The confusion matrix for test data is shown in figure (1) and training data in figure (2).

B. Part two

a. Number of Filters

I tried 32 and 6 filters, and the results are 89.2% and 87%. It is easy to see that increasing number of filters actually improve the accuracy but also take much more time to run.

b. Filter Sizes

I tried 5*5, 3*3, 1*1, and 10*10 for the filter sizes, and the results are 87%, 86.92%, 85.38% and 87.98%. The largest filter size seems to get best results but also takes a lot of time to run.

c. The Strides

I tried to set the step sizes (the strides) as 1, 2 and 5. The results are 89.2%, 85.68% and 79.8%. We can see that decreasing strides lead to increasing accuracy.

d. The Padding Options

I tried to set all of the padding options to 'same', [1 1 1 1] and [0 0 0 0]. The results are 87.66%, 86.98% and 86.62%. In my case, using all 'same' padding options works better.

e. Pool Sizes

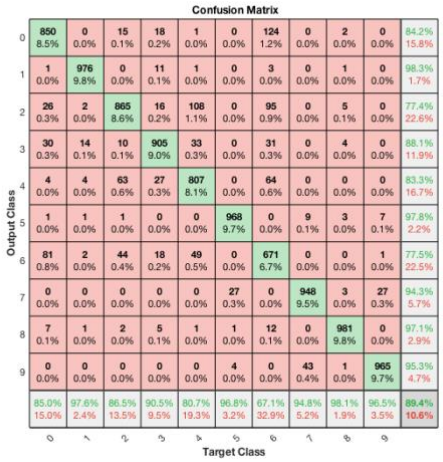

Figure (3): Confusion Matrix for Test Data for Convolutional Neural Network

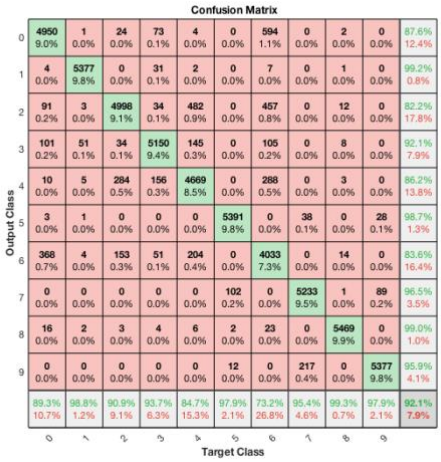

Figure (4): Confusion Matrix for Training Data for Convolutional Neural Network

I tried to set the pooling sizes to 2*2, 5*5 and 10*10 in the all of the pooling layers, and the results are 87.66%, 87.8% and 85.76%. Therefore, proper pooling sizes are needed for optimizing the results.

My best validation accuracy after adjusting the parameters is 90.52% and my test accuracy is 89.4%. My parameters are set as following: I used two convolutional layers, one with 8*8 filter size and 9 filters, the other with 7*7 filter size and 16 filters. The strides are all one and the padding options are all 'same'. I used leaky ReLU as all my activation functions and used two fully-connected layers, one with 300 neurons and the other with 10 neurons. I used one average pooling layer with 5*5 pooling size and 2 in both horizontal and vertical step sizes. The confusion matrices are shown above.

V. Summary and Conclusions

With this experiment, we could effectively see the influence of different parameters on the validation accuracy. As we could see from the results, the convolutional neural network works better than a pure fully-connected neural network. But the parameters also matter. In conclusion, convolutional neural network works better than fully-connected network, and a mixture of both is even better. Also, choosing proper values when we are setting the number of layers, width of layers, learning rate, padding options is preferable. Larger L2 Regularization rate, number of epochs, number of filters, and filter sizes, smaller strides and pooling sizes will lead to better results but also slower operation. Optimizers and activation functions should be chosen based on different situations. Adjusting all of the parameters above, you will get a good accuracy eventually.

Appendix A: MATLAB Functions Used and Brief Implementation Explanation

I2 = im2double(I) converts the image I to double precision. I can be a grayscale intensity image, a truecolor image, or a binary image. im2double rescales the output from integer data types to the range [0, 1].

B = permute(A,dimorder) rearranges the dimensions of an array in the order specified by the vector dimorder.

B = categorical(A) creates a categorical array from the array A. The categories of B are the sorted unique values from A.

layer = imageInputLayer(inputSize) returns an image input layer and specifies the InputSize property.

layer = fullyConnectedLayer(outputSize) returns a fully connected layer and specifies the OutputSize property.

layer = reluLayer creates a ReLU layer.

layer = leakyReluLayer returns a leaky ReLU layer.

layer = tanhLayer creates a hyperbolic tangent layer.

layer = softmaxLayer creates a softmax layer.

layer = classificationLayer creates a classification layer.

options = trainingOptions(solverName,Name,Value) returns training options with additional options specified by one or more name-value pair arguments.

net = trainNetwork(X,Y,layers,options) trains a network for image classification and regression problems.

plotconfusion(targets,outputs) plots a confusion matrix for the true labels targets and predicted labels outputs.

[YPred,scores] = classify(net,X) predicts class labels for the image data in X using the trained network, net.

layer = convolution2dLayer(filterSize,numFilters) creates a 2-D convolutional layer and sets the FilterSize and NumFiltersproperties.

layer = averagePooling2dLayer(poolSize) creates an average pooling layer and sets the PoolSize property.[2]

[2] "Makers of MATLAB and Simulink." MathWorks, www.mathworks.com/

Appendix B: MATLAB Code

```
clear all; 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_test = reshape(X_test,[10000 28 28 1]);
X_train = permute(X_train,[2 3 4 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)';

layers = [imageInputLayer([28 28 1])
        fullyConnectedLayer(784)
        %reluLayer
        leakyReluLayer
        fullyConnectedLayer(100)
        %reluLayer
        leakyReluLayer
        %fullyConnectedLayer(100)
        %reluLayer
        fullyConnectedLayer(10)
        softmaxLayer
        classificationLayer];

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

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

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

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

%% convolution

load fashion_mnist.mat
```

```
X_train = im2double(X_train);
X_test = im2double(X_test);
X_train = reshape(X_train,[60000 28 28 1]);
X_test = reshape(X_test,[10000 28 28 1]);
X_train = permute(X_train,[2 3 4 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)';

layers2 = [
    imageInputLayer([28 28 1])
    convolution2dLayer([8 8],9,"Padding","same","Stride",1)
    leakyReluLayer
    averagePooling2dLayer([5 5],"Padding","same","Stride",[2 2])
    convolution2dLayer([7 7],16,"Padding","same","Stride",1)
    leakyReluLayer
    fullyConnectedLayer(300)
    leakyReluLayer
    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer()];

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