Ruimin Zhang
March 13th, 2020

Neural Network

Abstract:

This is the homework 5 of AMATH 482 in the University of Washington. This assignment focuses on neural network.

Sec. I. Introduction and Overview

In this assignment, we will discuss fully connected neural network (Part I) and convolutional neural network (Part II) through MATLAB. Given some pictures of 10 different kind of clothes, we are able to train our network in order to let the program help us classify.

Sec. II. Theoretical Background

2.1 Neural network

For linear and logistic regression, there were three ingredients: the data, the model, and the loss function. Recall that the model was just a function that gives the relationship between the independent variable (x) and the dependent variable (y), which could be scalars or vectors. For linear regression, the model was a linear function. For logistic regression, it was a linear function that was then plugged into a nonlinear function (the logistic or softmax function). A neural network is just a different model. It is a different (and more complicated) function that we can use to relate all our x and y variable. However, we are going to see that sometimes the function is so complicated that we don't write down the formula. Instead, we draw pictures.

Neural networks have an interesting history and the history helps explain a lot of the terminology involved with neural networks and why we use these complicated functions as a model for regression and classification. I will give just a brief overview of the history (with more to come in later lectures). First, we need to talk about neurons. These are the cells that make up our brains. Neurons have a cell body with a long filament called an axon. Coming off the axon are connections called synapses. Neurons send out electrical signals down the axon and through the synapses to other neurons. If a neuron receives enough electrical signals from other neurons, it will fire its own electrical impulse.

Although a single neuron behaves in a relatively simple way, our brains have about 86 billion neurons. On average, each is connected to 7,000 other neurons. They are often arranged in consecutive layers. The collective behavior of all of the neurons can lead to very complex thought and behavior.

Motivated by the biological nueron, Warren McCulloch and Walter Pitts (1943) create a simple model of networks of artificial neurons that can do simple computations. As an example, consider the two networks below. The two neurons on the left can send electrical signals to the one on the right through the connections. We will assume that the output neuron is active (i.e. 1outputs and electrical signal) if at least two of its inputs are active.

While the model is nice, it isn't really set up to do the kinds of calculations that we need to classification and regression. The framework for that came in 1957 by Frank Rosenblatt when he invented the perceptron. Instead of the building blocks we saw above, 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. The layer of neurons on the left are the input

layer and they just output the same number they input. We will call these numbers x1, x2, and x3. We can denote the weights of the connections by w1, w2, and w3

We first calculate a weighted sum of the inputs:

$$z = w_1x_1 + w_2x_2 + w_3x_3 \tag{1}$$

We've been doing enough linear algebra in this course to know that we can write this sum using vectors (it is a dot product):

$$x = \begin{pmatrix} x1 \\ x2 \\ x3 \end{pmatrix}, \qquad w = \begin{pmatrix} w1 \\ w2 \\ w3 \end{pmatrix}$$

$$\mathrm{z} = w^T x = w_1x_1 + w_2x_2 + w_3x_3 \tag{2}$$

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. To do this mathematically, we can just plug the weighted sum into a step function.

$$y = \sigma(z) = \begin{cases} 0, & z < threshold \\ 1, & z \geq threshold \end{cases} \tag{3}$$

We are taking some input vector x and projecting it onto the vector w. If this exceeds some threshold, we get one output (y = 1) and if it is below the threshold, we get another output (y = 0). This is exactly what we did for linear discriminant analysis for dogs and cats! For linear discriminant analysis, we had a procedure for finding an optimal w vector in order to best separate our data into two classes. With this linear threshold unit, the calculation is exactly the same so these can be used for binary classification. The problem is that Rosenblatt didn't have some optimal way to determine the weights w. One thing to note is that typically the threshold is 0. So the step activation function is

$$\sigma(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases} \tag{4}$$

To account for shifting the threshold, we can add an extra neuron to the input layer called a bias neuron. It always outputs a 1.

In terms of the formula this amounts to adding a constant term b.

$$y = \sigma(w1x1 + w2x2 + w3x3 + b) \tag{5}$$

This is a linear threshold unit. I want you to notice the similarity to logistic regression. For logistic regression, you take a linear function and then plug it into the logistic function. The formula above is exactly the same except that it uses the step function instead of the logistic function. So if you were to use a logistic function instead of a step function, this is logistic regression. (We will get to this soon, but eventually people did start using the logistic function instead of a step function).

A perceptron is composed of a layer of linear threshold units. Each one is connected to the input layer and has a different set of weights.

If you define the vectors and matrix

$$A = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix}, \qquad b = \begin{pmatrix} b1 \\ b2 \\ b3 \end{pmatrix}, \qquad y = \begin{pmatrix} y1 \\ y2 \\ y3 \end{pmatrix}$$

then the calculation of the outputs y1, y2 and y3 can be written as

$$y = \sigma(\mathbf{A}x + \mathbf{b}) \tag{6}$$

In other words, you do the calculation Ax + b (which is the formula for multivariate linear regression) and then you plug each component of that vector into the (step) activation function. Rosenblatt invented the perceptron in 1957, but it quickly fell out of favor. There are a few reasons for this. First, a perceptron is pretty limited in what it can calculate. So it was never going to be able to solve really difficult classification or regression problems. However, it was discovered that you can solve much more complicated problems by stringing together multiple perceptrons. This gives what is called a multilayer perceptron (MLP).

The leftmost layer is the input layer and the rightmost layer is the output layer. Any layers in between are called hidden layers. Here we have pictured just one hidden layer, but you can have as many hidden layers as you want. More hidden layers allows you to solve more complicated problems.

The logistic activation function was the most popular for a long time, but eventually other activation functions began to be used for different reasons. One that has been used a lot is the hyperbolic tangent function σ(x) = tanh(x). It has the same shape as the logistic function but goes from −1 to 1. You could refer to both of these functions as sigmoid functions. The activation function that is probably the most popular today is the rectified linear unit (ReLU) which is σ(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.

The multilayer perceptron that we referred to earlier, 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.

Sec. III. Algorithm Implementation and Development

Part I: fully connected neural network

Based on the sample code, I increased the number of layers, the width of layers and learning rate, decreased regularization parameters, and changed the activation functions and the maximum number of training iterations. After these process, I was able to get a really good result.

Part II: convolutional neural network

I want to employ the LeNet5 architecture, so I first processed my training data into the proper form for LeNet5 neural network. Then I constructed the layers through Deep Network Designer on MATLAB and changed some variables in order to get higher accuracy.

Based on the sample code, I changed all the padding number in the convolutional layers and received a considerably good result for my final model. I also tried adding layers, changing the pool size, stride number, and filter size, but did not get very good results.

Sec. IV. Computational Results

Part 1: Fully connected neural network

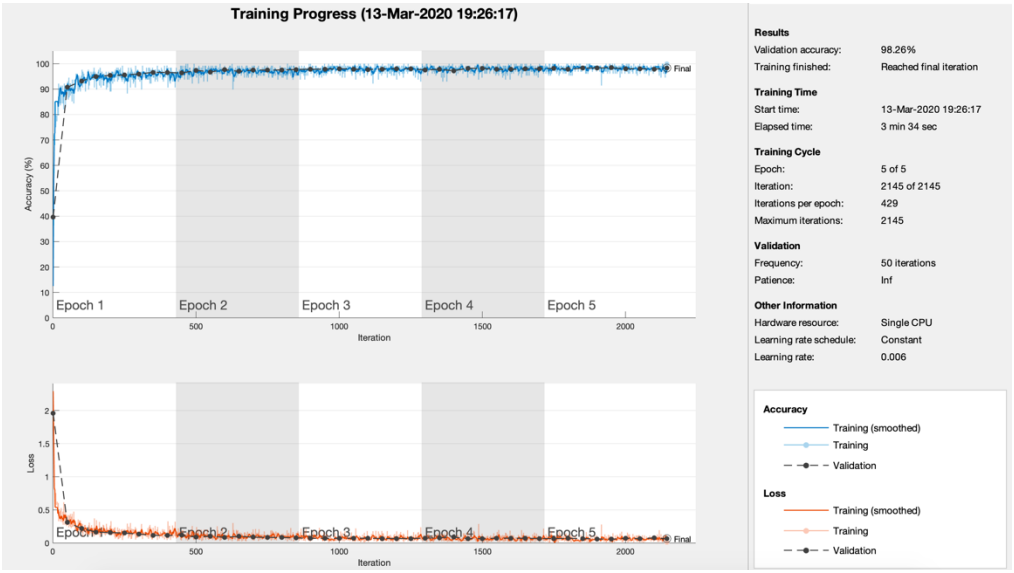

Figure1. This plot shows the training progress of fully connected neural network. The accuracy for this final model is 98.26%.

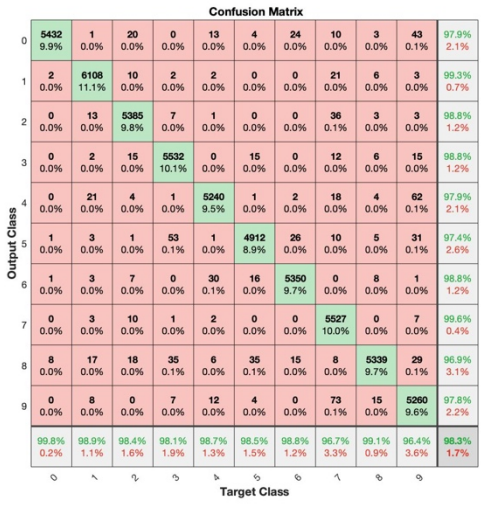

Figure2. Confusion matrix for training data.

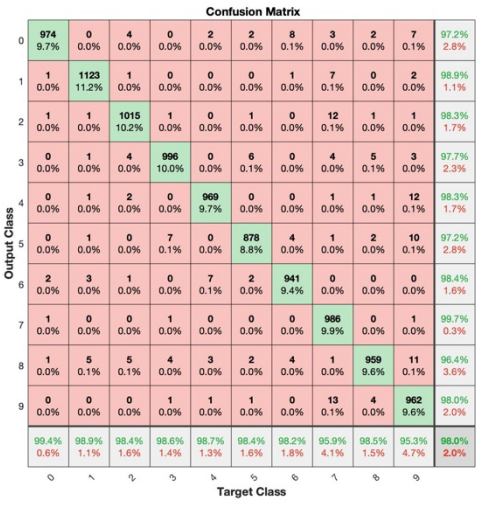

Figure3. Confusion matrix for testing data.

Part 2: Convolutional neural network

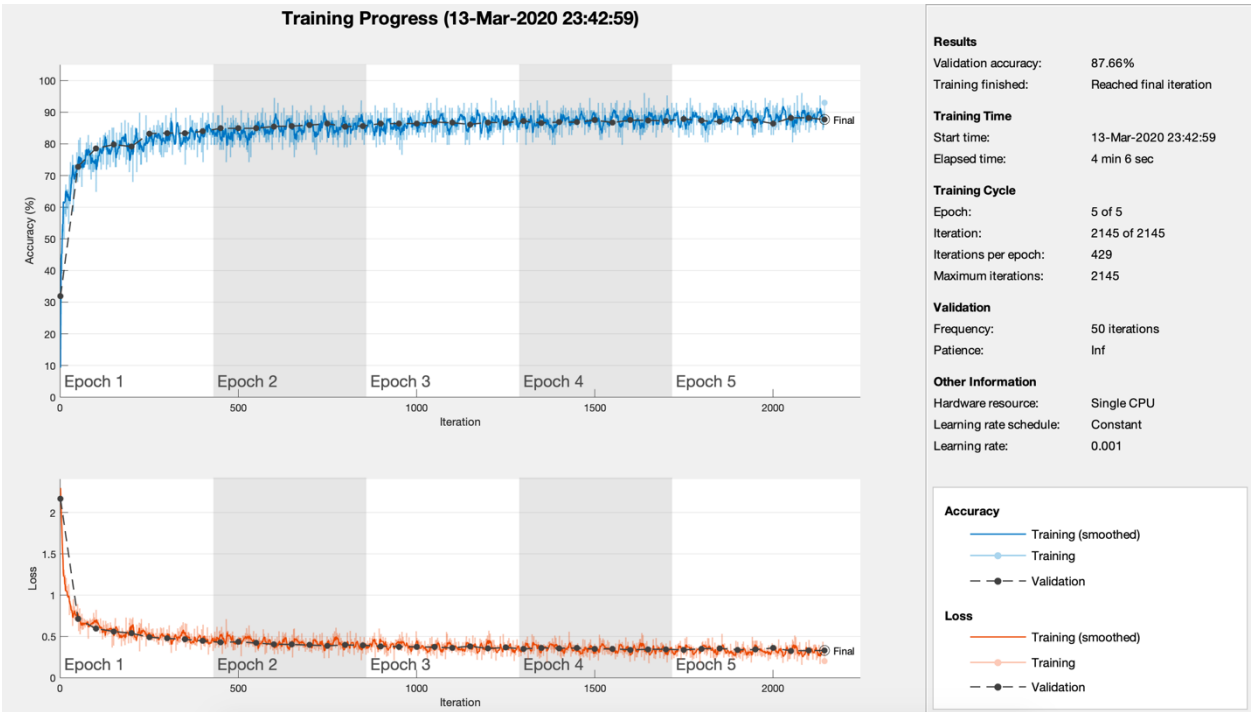

Figure4. This plot shows the training progress of convolutional neural network. The accuracy for this final model is 87.66%.

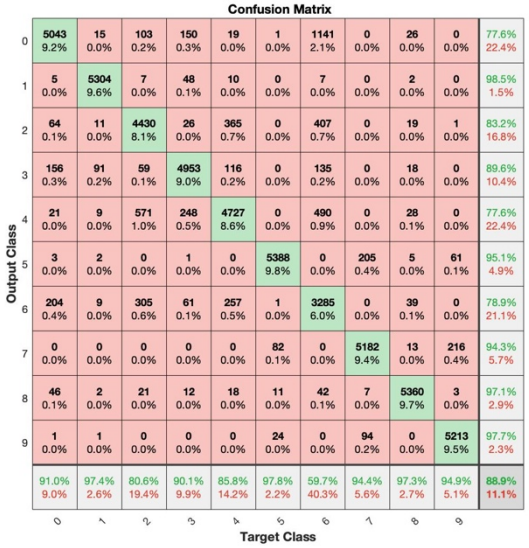

Figure5. Confusion matrix for training data.

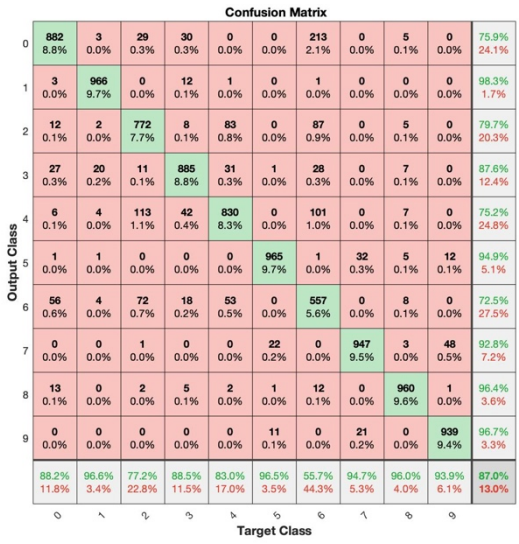

Figure6. Confusion matrix for testing data.

Observation

I had a better result in Part I. I believe this is because I have more control over the network, so I am able to understand how the variable works. However, for convolutional network, I can only build the structure but do not understand why some works better and some don't. This is a challenge for me. Additionally, each process takes a really long time if I change the stride to [1, 1] or add more layers (for approximately 10 minutes when I had 30 layers). If I have more time, I would like to try more neural network architecture and see if I can pass 90%.

Sec. V. Summary and Conclusions

To sum up, given a set of data, we are able to train the computer with our dataset, allowing the computer to find the features of the data and help us solve some difficult problems.

For fully connected neural networks, I will have more control over the number of layers and the width of the layers. However, since there are many variables in convolutional neural network, we need some existing architecture (such as LeNet5, GoogleLeNet, AlexNet, etc.) to help build the training process.

Appendix A. MATLAB functions used and brief implementation explanation

1. `B = permute(A,dimorder)` rearranges the dimensions of an array in the order specified by the vector `dimorder`. For example, `permute(A,[2 1])` switches the row and column dimensions of a matrix A.

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

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

Appendix B.MATLAB codes

Part I:

```
%% MNIST Classifier with deep neural network
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)';
```

```
%%
layers = [imageInputLayer([28 28 1])
        fullyConnectedLayer(500)
        reluLayer
        fullyConnectedLayer(900)
        reluLayer
        fullyConnectedLayer(500)
        tanhLayer
        fullyConnectedLayer(400)
        tanhLayer
        fullyConnectedLayer(100)
        tanhLayer
        fullyConnectedLayer(300)
        tanhLayer
        fullyConnectedLayer(100)
        tanhLayer
        fullyConnectedLayer(10)
        softmaxLayer
        classificationLayer];

options = trainingOptions('adam', ...
    'MaxEpochs',4,...
    'InitialLearnRate',2e-3, ...
    'L2Regularization',1e-4, ...
    'ValidationData',{X_valid,y_valid}, ...
    'Verbose',false, ...
    'Plots','training-progress');

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

%% Confusion for training
figure(2)
y_pred = classify(net,X_train);
plotconfusion(y_train,y_pred)

%% Test classifier
figure(3)
y_pred = classify(net,X_test);
plotconfusion(y_test,y_pred)
```

Part II:

```
%% MNIST Classifier with convolutional neural network
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)';


%%
layers = [
    imageInputLayer([28 28 1],"Name","imageinput")
    convolution2dLayer([5 5],6,"Name","conv_1","Padding",[1 1 1 1])
    tanhLayer("Name","tanh_1")
    averagePooling2dLayer([2 2],"Name","avgpool2d_1","Padding",[1 1 1
1],"Stride",[2 2])
    convolution2dLayer([5 5],16,"Name","conv_2")
    tanhLayer("Name","tanh_3")
    averagePooling2dLayer([2 2],"Name","avgpool2d_2","Padding",[1 1 1
1],"Stride",[2 2])
    convolution2dLayer([5 5],120,"Name","conv_3")
    tanhLayer("Name","tanh_2")
    fullyConnectedLayer(84,"Name","fc_1")
    tanhLayer("Name","tanh_4")
    fullyConnectedLayer(10,"Name","fc_2")
    softmaxLayer("Name","softmax")
    classificationLayer("Name","classoutput")];

options = trainingOptions('adam', ...
    'MaxEpochs',5,...
    'InitialLearnRate',1e-3, ...
    'L2Regularization',1e-4, ...
    'ValidationData',{X_valid,y_valid}, ...
    'Verbose',false, ...
    'Plots','training-progress');

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

%% Confusion for training
figure(1)
y_pred = classify(net,X_train);
plotconfusion(y_train,y_pred)

%% Test classifier
figure(2)
y_pred = classify(net,X_test);
plotconfusion(y_test,y_pred)
```

Reference

Kutz, Nathan. *Data Driven Modeling & Scientific Computing*