AMATH 482 Project 5 : Neural Networks for Classifying Fashion MNIST

Yixuan Liu

March 2020

1 Abstract

The artificial neural network is inspired by the biological neural networks. With this ANN, we can train our model to perform tasks rather than manually design some specific rules to do tasks. In other words, our model can learn from data by ANN. In this project, we will play around deep neural network and convolutional neural network.

2 Introduction and Overview

In this project, we will explore the power of the deep neural network and the convolutional neural network to classify 10 different class of fashion items. The main goal is to construct an appropriate neural network that can best predict new data and find hyperparameters that can best improve the efficiency of optimizer.

3 Theoretical Background

3.1 Artificial neural network

[2] An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron that receives a signal then processes it and can signal neurons connected to it.

[2] 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.

3.2 Some layers

There are few important layer that we use in this project:

- full connected layer can connect every node in the layer with the node in the previous layer.
- convolutional layer can slide small filters on the node in the previous layer and produce an activation map to represent features of data in previous layer.
- pooling layer can progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network.

3.3 Activation functions

The activation function can convert a input signal of a node to an output signal. The most famous function are ReLu and sigmoid function, which we are using in our project.

3.4 Gradient descent

[1] Gradient Descent is a process that occurs in the backpropagation phase where the goal is to continuously resample the gradient of the model's parameter in the opposite direction based on the weight w, updating consistently until we reach the global minimum of function J(w).

4 Algorithm Implementation and Development

4.1 Deep neural network

4.1.1 Loading data

Firstly, we load the fashion mnist data, which contains 60000 pictures of 10 categories of clothing for training and 10000 pictures of 10 categories of clothing for testing. Then, we convert those datasets into a double which is easier to handle. After that, we reshape those datasets into (28 28 1 size) matrices. Finally, we assign the first 5000 training data as validation data and the rest of the original training data as new training data. Also, making our label categorical is required by MATLAB. To implement a fully-connected neural network, we should create layers first. We first input the image by imageInputLayer, then use fullyConnectedLayer with 300 neurons with ReLU activation functions and use fullyConnectedLayer with 100 neurons with ReLU activation functions. Finally, use fullyConnectedLayer with ten neurons and SoftMax function to get the probability of each category. The last step before training is to set our hyperparameters of optimizer: we choice adam optimizers with initial learn rate 1e-3, L2 regularization 1e-3, validation data defined before, and max epochs 5. With all these setups, we get around 88% accuracy for validation data. To get above 90% accuracy, we play around hyperparameters and setting.

4.1.2 Adjusting max epochs

we try to adjust the max epochs because we say the trend in the training diagram is increasing when we stop it. We try 35,30,25 three numbers to see the effect of max epochs. Thirty-five rounds give us a separation of accuracy between training data and validation data. In the very end, we see that the accuracy of training data increases above 90%. Still, the accuracy of testing data stays around 88%, which means we do not need the last few rounds since the trend is not increasing anymore. Then, we try to 30 and 25, which both give us appropriate accuracies, so we choose 30.

4.1.3 Adjusting L2 Regularization

we try to adjust the L2 Regularization. Since this hyperparameter can avoid overfitting, we try 1e-2, 1e-7, 1e-3, and 1e-4 to see the differences of results. As the rate large, the loss of training data and validation data stay very close to each other. As we decrease the rate, their loss is separate, and the loss of validation data even increase at some point, which is the phenomenon of overfitting. So we just chose 1e-3.

4.1.4 Adjusting depth and width of hidden layer

We first set the threshold by looking at the diagram of projection data onto the w. Then, we manually choose the edge of each cluster and set the threshold to be the average of the two edges(see figure 2.4.6). After that, we do the same process to load data with different songs, then project those data onto the principal component space. Finally, we project these data onto the eigenspace and classify theses value with the threshold. In some cases, we compare two components of projection data with two thresholds, and sometimes we just compare one component with one threshold. All the decisions are made manually. There could be an improvement of accuracy if we choose two lines to separate the clusters.

4.1.5 Adjusting other parameters

we change the activation function in the hidden layer with tanh function and do not see much difference. In the very end, we change the learning rate because there is some relationship between the learning rate and the number of hidden layers, and the max epochs also affect the learning rate. We try several learning rates to see its effect:1e-2, 1e-3, 1e-4 and 1e-7. The obvious effect of decrement of learning rate is the increase of the slope of the decreasing loss. As we decrease the learning rate, the loss decreases faster, and the time of iteration becomes shorter. Also, the accuracy of validation data first increases and then decreases, while peaks at 1e-4.

4.2 part two

we start precisely the same settings except for different layers, which is in figure 1.

4.2.1 Adjusting sampling method

we change the sampling method to choose the maximum rather than average. We see the improvement of accuracy, which is a decent strategy. I think the maximum value can better represent features.

4.2.2 Adjusting the number of filters

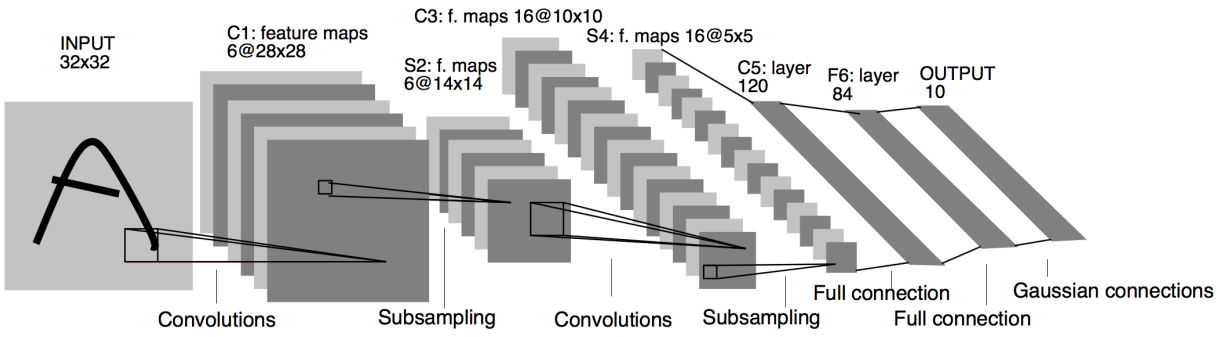
we change the number of filters for our convolutional layers. We increase the layer size to 20,50 and 120 for these three convolutional layers. As we increase the number of filters, the accuracy is enhanced by a tiny amount. However, the time of iteration becomes much more prolonged than a smaller size.

4.2.3 Adjusting the size of filters

we change the size of filters from 5x5 to 3x3. The accuracy is increased by a decent amount. We think the reason is that we have more excellent filters which can preserve more information.

4.2.4 Adjusting other parameters

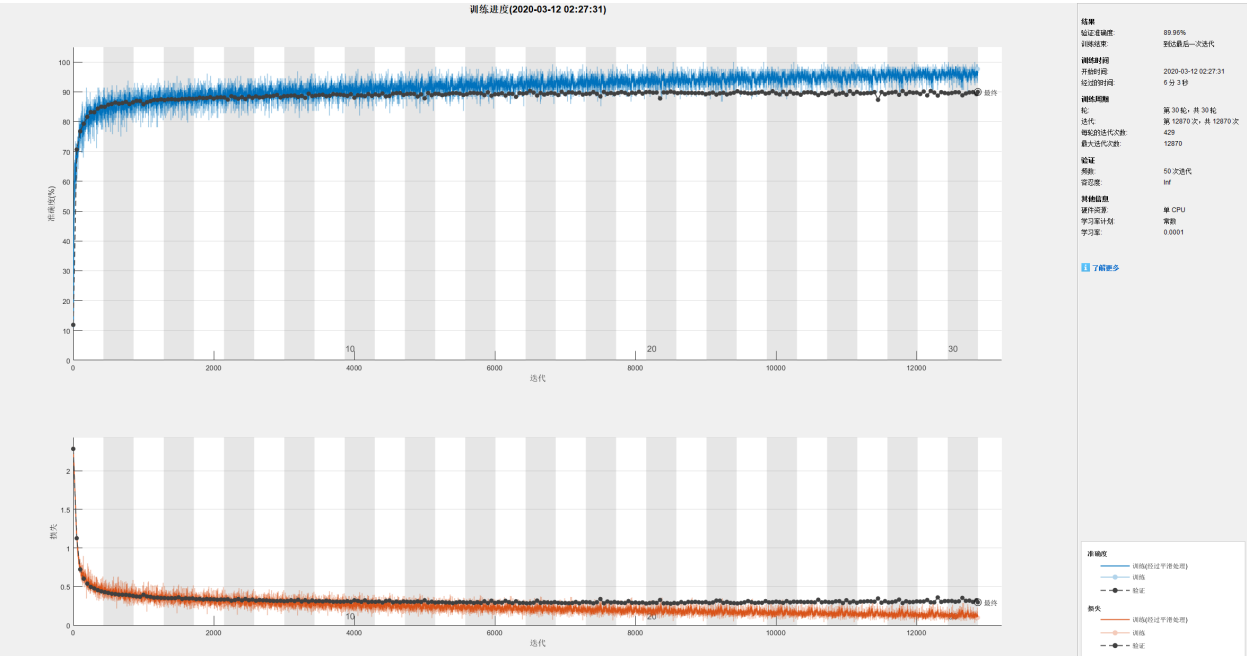
we decrease the strides to see some effect on accuracy. However, the speed of simulation is too slow to see the output. We do not do match on the strides. Lastly, we think the padding options are excellent with the 'same' options because we can capture all the information in the image.



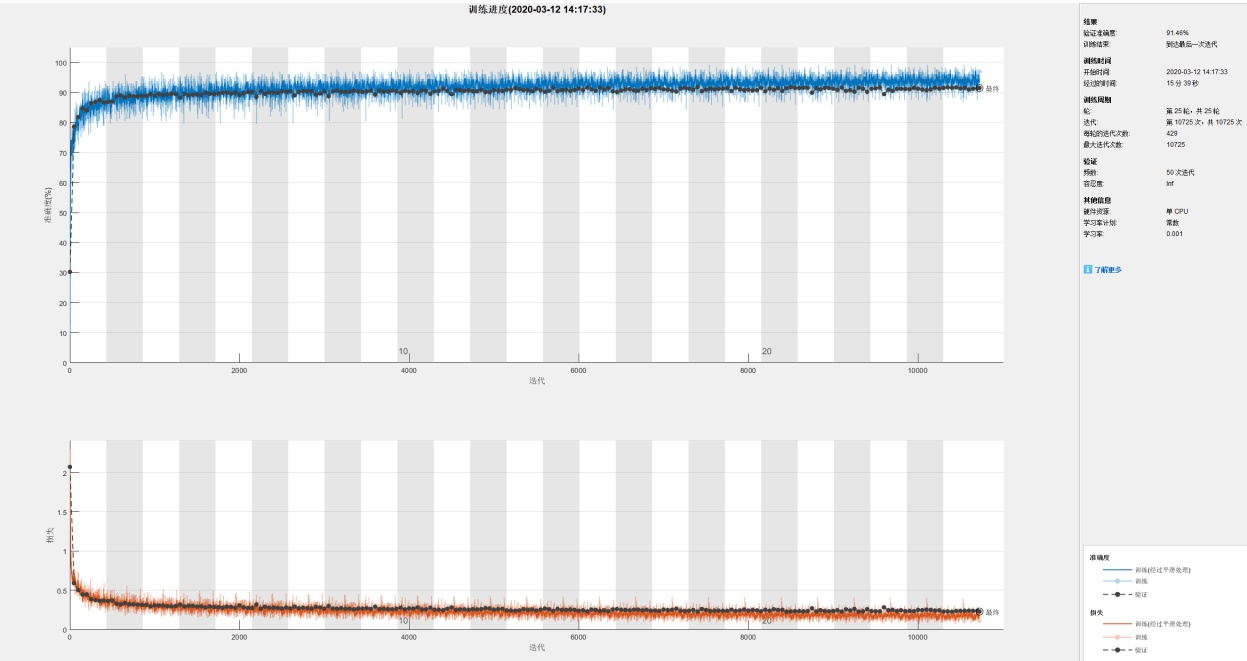
(a) Figure 1 : The layer for convolutional neural network

5 Computational Results and Conclusion

For part one, we achieve around 90% accuracy for validation data and around 89% for test data, which is a decent accuracy. For part two, we achieve 91.46% accuracy for validation data and 91% accuracy for test data easily. If we do more adjustment on the hyperparamters, we can achieve even higher accuracy. To put it into a nutshell, the convolutional neural network performed better than the deep neural network.



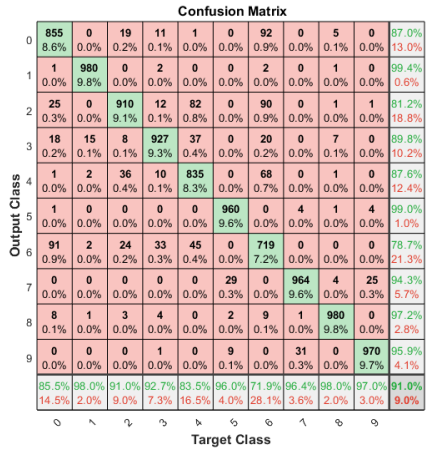
(a) Figure 2 : The simulation of deep neural network 89.86%



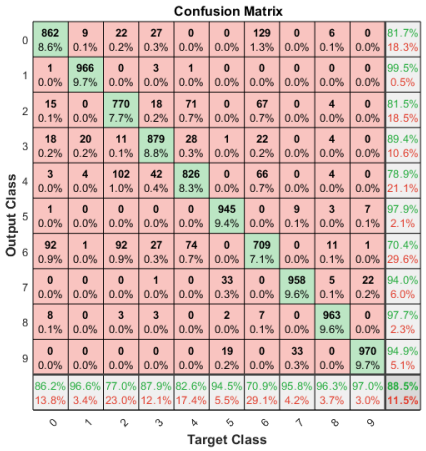
(a) Figure 3 : The simulation of CNN 91.46%

6 Appendix A: MATLAB functions used and brief implementation explanation

- im2double: convert uint8 to double
- permute(A,ORDER): rearranges the dimensions of A so that they are in the order



(a) Figure 4 : The confusion plot of deep neural network for test data



(b) Figure 5 : The confusion plot of CNN for test data

specified by the vector ORDER

- trainNetwork(imds, layers, options): trains and returns a network trainedNet for a classification problem
- classify(SAMPLE,TRAINING,GROUP): classifies each row of the data in SAMPLE into one of the groups in TRAINING
- plotconfusion(targets,outputs): plots a confusion matrix, using target (true) and output (predicted) labels

7 Appendix B: MATLAB codes

```
%% 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(300)
        reluLayer
        fullyConnectedLayer(100)
        reluLayer
        fullyConnectedLayer(100)
        reluLayer
        fullyConnectedLayer(50)
        tanhLayer
        fullyConnectedLayer(10)
        softmaxLayer
        classificationLayer];

options = trainingOptions('adam', ...
    'MaxEpochs',30,...
    'InitialLearnRate',1e-4, ...
    'L2Regularization',1e-3, ...
    '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_valid);
plotconfusion(y_valid,y_pred)

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

%% 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([3 3],20,"Name","conv_1","Padding","same")
    tanhLayer("Name","tanh_1")
    maxPooling2dLayer([2 2],"Name","avgpool2d_1","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],50,"Name","conv_2")
    tanhLayer("Name","tanh_3")
    maxPooling2dLayer([2 2],"Name","avgpool2d_2","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],120,"Name","convc_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',25,...
    'InitialLearnRate',1e-3, ...
    'L2Regularization',1e-3, ...
    '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)
```

References

[1] Hamza Mahmood. *Gradient Descent*. URL: https://towardsdatascience.com/gradient-descent-3a7db7520711.

[2] wikipedia. *Artificial neural network*. URL: https://en.wikipedia.org/wiki/Artificial_neural_network.