CS492C – Introduction to Deep Learning – Programming Assignment #2

School of Computing 20183309 Giyeon Shin

1. Previous Experimental Setup in PA#1

The main issue of this programming assignment is to use convolutional neural networks for MNIST datasets. In programming assignment #1, we used only fully-connected layer to find the labels for MNIST datasets. I used this experimental setup in programming assignment #1 and I progressed this programming assignment #2.

In programming assignment #1, I used batch normalization, dropout and L2-regularization. Due to reduce the overfitting problem, I decided to use dropout, batch normalization and L2-regularization. As the layers become deeper, higher level features can be found. On the other hand, the decrease of accuracy in deeper layers means that the model tends to be overfitting.

When the rate of dropout is 0.30 and L2-reg scale is 0.005, the model showed the best performance in PA#1. So, I used these fully-connected layer options for output layer or dense layer.

In addition, I used data augmentation methods in programming assignment #1. The main task of this assignment and previous assignment is same to classify the image dataset to numeric digits (0-9). Unlike the numeric digit dataset used in MNIST in general, we should deal with more complex dataset to work on. As I looked at the dataset, there were some noises in the images, and there were many cases where the numeric digits were not in the vertical direction correctly. Since the numeric digits were also inverted upside down or reversed left and right, the features seemed to be insufficient to just learn it as a fully-connected layer or convolutional layer. This institution comes from the fact that there are only 10,000 train dataset, while the test dataset has 50,000 image datasets. I decided to proceed with the data augmentation of the training dataset based on the judgement that the amount of training dataset is insufficient to learn the architecture by deep learning method.

The condition for the data augmentation shows the best performance to create 15 new datas for each data and concatenate them with original data. In this case, the rotate degress was determined rom –PI to PI by normal distribution, and the existence of vertical flip was determined by Bernoulli distribution. It shows better performance than the data generated in the uniform distribution.

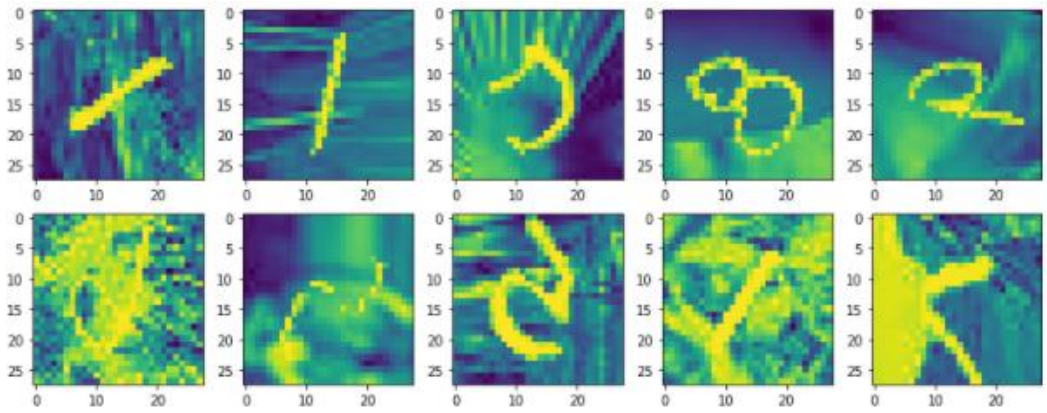

Figure 1. Plots of the training dataset

Finally, the each convolutional neural network layer consists convolutional layer and pooling layer. Each setup for options of convolutional layer and pooling layer is referred in Tensorflow MNIST site. All experiment setups have the convolutional neural network layer, one fully-connected layer for combination of filters in convolutional neural network layer and the last output layer for classification.

2. Experimental Setup

I worked the experiment with the change of layer depths, pool stride size and convolutional filter sizes. The common process of these experiment is that there are the data augmentation process to extend the input data. So, the each data would be extended to sixteen times of its size with itself. Then, the data would go through convolutional neural network.

Each convolutional neural network is consist of convolutional layer, pooling layer and dropout layer. With change of the options in each layer, I examined the performance in various experiments. Then, the output of convolutional neural network is processed in dense layer with l2 regularization and batch normalization and dropout. All convolutional neural network padding option is 'SAME'. Finally, It is processed in output layer.

3. Result of Programming Assignment #2

Conv1 Filter	Pool1 Stride	Conv2 Filter	Pool2 Stride	Conv3 Filter	Pool3 Stride	Conv4 Filter	Pool4 Stride	Conv5 Filter	Pool5 Stride	Fully units	Output units	Acc.
32	2									1024	10	0.817
32	2	64	2	128	2					1024	10	0.903
32	1	64	1	96	2	128	2	196	1	1024	10	0.941

Table 1. Final performance by the depth change (filter_size of conv. = [5,5], filter_size of pool. =[2,2])

As a result of integrating the above experiments, the model which is learned by the data extended through data augmentation processing has better performance than the model which is learned by only original data as programming assignment #1. The result of programming assignment #2 is the table above and the filter size of convolutional layer is [5,5] and filter size of pooling layer is [2,2] in all programming assignment #2 experiment option values.

The noticeable one in the final experiment of programming assignment #2 is that the depth of the network significantly affect the accuracy although it did not affect significantly affect the performance in programming assignment #1. This is due to the characteristics of the MNIST data, which means that the nature of the image data manes that the feature is better represented on convolutional neural networks and a higher level of feature can be found in deep convolutional neural networks.

Conv1 Filter	Pool1 Stride	Conv2 Filter	Pool2 Stride	Conv3 Filter	Pool3 Stride	Conv4 Filter	Pool4 Stride	Conv5 Filter	Pool5 Stride	Fully units	Output units	Acc.
32	2	64	1	96	2	128	1	196	1	1024	10	0.923
32	1	64	1	128	2	128	2	196	1	1024	10	0.941

Table 2. Final performance by pooling layer stride change (filter_size of conv. = [5,5], filter_size of pool. =[2,2])

Another noticeable point in this experiment is that the pooling stride size is very important to classify the image datasets. As you can see the table above, the difference of pooling layer size in the first and the second layer makes the difference of accuracy bigger. The important role of the pooling layer is to reduce the amount of computation by taking only those pixels that are the highest in the surrounding pixels, similar to the human eye, when judging image data. This role reduces the amount of computation which is beneficial in memory and time, but it also leads to the loss of features. In particular, the loss of features occurring at an early stage consecutively loses features at higher levels. In this sense, not using a pooling layer might be better, but it is better to delay the pooling steps in the architecture by understanding the amount of computation because of hardware limitations.

Conv1 FilterSize	Conv2 FilterSize	Conv3 FilterSize	Conv4 FilterSize	Conv5 FilterSize	Fully Units	Output Units	Acc.
[3,3]	[3,3]	[3,3]	[3,3]	[3,3]	1024	10	0.932
[5,5]	[5,5]	[5,5]	[5,5]	[5,5]	1024	10	0.941
[7,7]	[7,7]	[7,7]	[7,7]	[7,7]	1024	10	0.928

Table 3. Final performance by conv. layer filter size change (etc. option is same)

Table 3 is the result of final performance by convolutional layer filter size change when other option is same as Table 2. As you can see the table above, the bigger size of filter in convolutional layer or smaller size does not make the architecture efficiently. It tells us that we need to experiment with filter size to find the filter size with the best performance.

4. The total number of parameters in network

I calculated the total number of the parameters of 3, 5, 7 depth of layers in network that have best performance in each depth of layers of Table 1.

At first, in 3 depth layer, there are one convolutional neural network layer, one FC layer and one output layer. Input size is 784 (28*28*1), the convolutional kernel is [5,5] and the number of filters is (14*14*32). The parameter of convolutional layer is ((input_feature_maps) * (filter_size_height) * (filter_size_width) + 1) * (output_feature_maps), so the parameter of the convolutional layer in 3 depth layer is (5*5*1+ 1) * 32 = 832. The output size of convolutional layer is (32 * 14* 14). So the parameters of the dense layer is (32*14*14+1) * 1,024 = 6,423,552. Finally, the parameters of output layer is (1024 + 1) * 10 = 10,250. The total parameter of 3 depth network is 6,434,634.

#	Name	Size	Parameters
0	Input	28*28*1	0
1	Conv2d (1)	28*28*32 (Padding: SAME)	(5*5*1+1)*32 = 832
2	Maxpool (1)	14*14*32	0
3	Fully-Connected	1024	(14*14*32+1) * 1024 = 6,423,552
4	Output	10	(1024+1) * 10 = 10,250
	Total		6,434,634

Table 4. The number of parameters in 3-depth network.

In 5 depth layer, there are three convolutional neural network layer, one fully-connected layer and one output layer. The convolutional kernels are [5,5] and each feature maps are 32, 64 and 128. So if we considered the feature map size of each convolutional layers, we can solve the number of parameters easily such as 3-depth layer.

#	Name	Size	Parameters
0	Input	28*28*1	0
1	Conv2d (1)	28*28*32 (Padding: SAME)	(5*5*1+1) * 32 = 832
2	Maxpool (1)	14*14*32 (Stride : 2)	0
3	Conv2d (2)	14*14*64 (Padding: SAME)	(5*5*32+1) * 64 = 51,264
4	Maxpool (2)	7*7*64 (Stride: 2)	0
5	Conv2d (3)	7*7*128 (Padding: SAME)	(5*5*64+1) * 128 = 361,088
6	Maxpool (3)	3*3*128 (Stride: 2)	0
7	Fully-Connected	1024	(3*3*128+1) * 1024 = 1,180,672
8	Output	10	(1024+1) * 10 = 10,250
	Total		1,604,106

Table 5. The number of parameters in 5-depth network.

In 7-depth layer, there are five convolutional layer which have 32, 64, 96, 128, 196 feature maps.

#	Name	Size	Parameters
0	Input	28*28*1	0
1	Conv2d (1)	28*28*32 (Padding: SAME)	(5*5*1+1) * 32 = 832
2	Maxpool (1)	27*27*32 (Stride : 1)	0
3	Conv2d (2)	27*27*64 (Padding: SAME)	(5*5*32+1) * 64 = 51,264
4	Maxpool (2)	26*26*64 (Stride: 1)	0
5	Conv2d (3)	26*26*96 (Padding: SAME)	(5*5*64+1) * 96 = 153,696
6	Maxpool (3)	13*13*96 (Stride: 2)	0
7	Conv2d (4)	13*13*128 (Padding: SAME)	(5*5*96+1) * 128 = 307,328
8	Maxpool (4)	6*6*128 (Stride: 1)	0
9	Conv2d (5)	6*6*196 (Padding: SAME)	(5*5*128+1) * 196 = 627,396
10	Maxpool (5)	5*5*196 (Stride : 1)	0
11	Fully-Connected	1024	(5*5*196+1) * 1024 = 5,018,624
12	Output	10	(1024+1) * 10 = 10,250
	Total		6,169,390

Table 6. The number of parameters in 7-depth network.