Q1 and Q2 are based on the python code of CNN framework (in [appendix 1](#appendix1_)) with the structure [CONV-CONV-POOL-FC]. The LeNet style diagram (in [appendix 2](#appendix2_)) shows the number of feature map and the size of feature map after each convolution, max-pooling and full connection.

Q1. Description of convolution kernel and loss function

AlexNet style diagram (figure 1) describes more visual structure. The size of input image, downloaded from MNIST datasets, is , which means 1 channel and pixels. Next, the first convolution includes 8 @ filters with stride 1, outputting 8 feature maps, as the input data for the second convolution. Next, 16 @ filters with stride 1 participate in the second convolution, generating 16 feature maps. Then, the code executes the max pooling, which outputs 16 @ feature maps. After two full connections, we get prediction output, such as [0.11, 0.23, 0.12, 0.54, 0.12, 0.31, 0.21, 0.68, 0.21, 0.13], and the number stands for the score for the digit on that position.

图片包含 图示

描述已自动生成

Figure 1. The AlexNet style of CNN in code

**1.1 Description of Convolution**

Taking the image (figure 2) below as an example, the report analyzes the two convolutions and loss function used in the CNN. The figure (in [appendix 3](#appendix3_)) shows the pixels matrix of image above.

QR 代码

描述已自动生成

Figure 2. The digit image analyzed in the Q1

**Convolution 1**

Assumptions

Firstly, I assume the 1st convolutional kernel in the 1st convolution as:



Figure 3. The assumed convolutional kernel

The function in the convolution is shown below, but I set b=0 when firstly calculating.



After convolution, we get output with new size: , , and . ( is the width of input, is the height of input, is the extent of filter, is the number of zero padding, is the width of output, is the height of output, and is the dimension of data)

After each convolution, the output would go through ReLU, activation function in this experiment. The ReLU function will help capture the main feature from image.

(1.1)

The process of convolution

Before convolution, 0 as padding is padded into pixels to remain the size of output being and maximize the utility of edge pixels. This report just shows a little part of process.



In the 1st convolution, there are 8 different filters; therefore, after repeating the process above, we can get 8 @ feature maps. Finally, we get parameters.

**Convolution 2**

The execution in convolution 2 (in [appendix 4](#appendix4_)) is same as that in 1st convolution. However, there are differences. First, the input data for 2nd convolution is 8 @ feature maps undergoing previous ReLU. Second, 16 filters are used in this convolution. Therefore, after ReLU function, we get 16 @ feature maps, and parameters.

**1.2 Description of loss function**

**The calculation process on softmax\_with\_cross\_entropy**

The loss function (in [appendix 5](#appendix5_)) is softmax\_with\_cross\_entropy. The softmax function is shown below,

(1.2)

and the cross-entropy function is

(1.3)

In this case, I assume the output generated by CNN model is [0.11, 0.68, 0.12, 0.54, 0.12, 0.31, 0.21, 0.12, 0.21, 0.13].

After processed by softmax function, the new output is [0.08, 0.15, 0.08, 0.13, 0.08, 0.10, 0.09, 0.09, 0.09, 0.09].

The target is [0, 1, 0, 0, 0, 0, 0, 0, 0, 0], so in this case cross entropy loss is:

It is evident that the loss is so large that machine should optimize the parameters by to minimize the total cost. By following the backward, we can calculate the , and get the most optimized parameter (the result and proof process are in [appendix 6](#appendix6_)).

**Analysis of softmax\_with\_cross\_entropy**

In classification of handwritten digit, we use to stand for loss. According to the graph of loss function (in [appendix 7](#appendix7_)), the loss will be large if the output after softmax is small. Meanwhile, the output generated by softmax is also the confidence value for each prediction number.

According to the formula 1.2 and 1.3, softmax\_with\_cross\_entropy calculates probability for each prediction and has sensitivity to the output, so it is an outstanding loss function used to evaluate and optimize parameters.

Q2. Train and test on MNIST dataset

**2.1 Description of CNN model’s performance**

The train() (in [appendix 8](#appendix8_)) divides train dataset into 930 batches, and each batch includes 64 images and labels, totally input data. The experiment runs 14 epochs, in which test() (in [appendix 9](#appendix9_)) is called after train function. I printed average loss and accuracy of prediction in each epoch in [appendix 10](#appendix10_). It is exciting that the average loss after 14 epochs is minimized to 0.0363 and the accuracy of prediction reaches 98.82%, which represents the outstanding performance of the CNN model.

torch.save(model, **"./mnist\_cnn.pt"**)is added into test()rather than main(), to help save the most updated model into project.

**2.2 Well classified and Misclassified images**

The code (in [appendix 11](#appendix11_)) implements the function that predicts input images by calling trained CNN model and calculates the confidence values of each digit by using softmax loss as well as visualizes each image by calling matplotlib. According to results got, these images could be divided into three categories, namely well classified images, uncertainly classified images and misclassified images. And some examples are shown below.

Well classified images

The trained CNN model can accurately identify the handwritten digit, with providing very high confidence value.

图片包含 应用程序

描述已自动生成图片包含 应用程序

描述已自动生成QR 代码

描述已自动生成QR 代码

描述已自动生成图表

描述已自动生成

Uncertainly classified images

Though the model can accurately identify the handwritten digit, the confidence value on correct position is a little lower than previous category, and the machine also give a little high confidence value to other position.

QR 代码

描述已自动生成图片包含 QR 代码

描述已自动生成图片包含 QR 代码

描述已自动生成QR 代码

描述已自动生成

Misclassified images

Because of high accuracy of CNN, it is scarce to find misclassified images. However, when testing the model by using trainset from MNIST, I find two misclassified images (shown below).

QR 代码

低可信度描述已自动生成QR 代码

描述已自动生成

The digit on 1st image is 4, but the model predicts 6. However, by analyzing confidence value, it is obvious to discover that the model faces the trade-off between 4 and 6.

The digit on 2nd image is 2, but the model predicts 0. However, the confidence value of 0 is just 0.150 higher than that of 2.

Q3. Improving the classification performance

The LeNet-5 was firstly proposed in 1998 [1]. However, compared with the minimum mean squared error (MSE) used to calculate the loss of model, the softmax loss shows better performance [2]. In recent report [3], dropout was used to improve the model’s robustness. According to the latest theories, ReLU activation function, softmax\_with\_cross\_entropy loss and dropout are included in the new model to improve the performance.

**New CNN model and two changes**

The improved CNN model is also implemented by python code (shown in [appendix 12](#appendix12_)), and the structure of it is [CONV-POOL-CONV-POOL-FC] (in [appendix 13](#appendix13_)), the standard LeNet-5. To enhance the accuracy of CNN framework or reduce the model size, two attempts are included in the experiment.

First, a max pooling process is added. In the previous lab, the 2nd convolution is after the 1st convolution, which is a little different from classic framework; therefore, to seriously follow the LeNet framework, the new lab adds max pooling after the first convolution as the 1st max pooling. Second, the new lab does not pad 0 before executing convolution. And this change leads to changes of output’s width and height.

**Performance of new CNN model**

The new CNN model’s performance (in [appendix 14](#appendix14_)) is a little better than previous model’s. According to the [appendix 15](#appendix15_), new model’s lowest average loss is smaller, and accuracy is higher. And the new model reaches the most optimal status faster than previous model.

**Analysis of new CNN model**

According to [4], CNN-ReLU had the most number of correct predictions per class. Conversely, with its faster convergence, CNN-Softmax had the higher cumulative correct predictions per class. Therefore, this lab report once again proves the excellent performance of ReLU activation function and Softmax. The CNN-ReLU increase accuracy of model by 0.2% (in [appendix 16](#appendix16_)).

A new layer of max pooling helps over-fitting by providing an abstracted form of the representation. As well, it reduces the computational cost by reducing the number of parameters to learn and provides basic translation invariance to the internal representation.

In the new model, padding with 0 is removed because useful pixels are mainly concentrated in the central part of the image in MNIST. If we pad with 0 before convolution, the edge pixels will be overvalued, which will affect the extraction of main features.

The dropout layer is a mask that nullifies the contribution of some neurons towards the next layer and leaves unmodified all others. Dropout layers are important in training CNNs because they prevent overfitting on the training data. Two dropout layers are applied in the new model, after 2nd max pooling and 1st full connection. Though deleting two dropout layers increases accuracy of model by 0.13% (in [appendix 16](#appendix16_)), this action decreases the robustness of model, meaning the model would perform lower accuracy in other datasets than in MNIST dataset. Therefore, the lab concentrated on the improvement of robustness.

The purpose of the fully connected layer in a convolutional neural network is to detect certain features in an image. More specifically, each neuron in the fully connected layer corresponds to a specific feature that might be present in an image. The structure of two fully connected layers improves the accuracy of new model by 0.22% (in [appendix 17](#appendix17_)) because it more precisely identifies and selects the features from the outputs of 2nd max pooling.

The combination of layers above contributes to the improvement on the performance of new models. Global average pooling (GAP) may be an effective alternative to fully connected layer, so the accuracy of prediction would be improved by using GAP. However, this is just an assumption, and it needs more experiments to test the impacts of GAP in prediction process.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998.

[2] H. A. Alwzwazy, H. M. Albehadili, Y. S. Alwan, and N. E. Islam, "Handwritten digit recognition using convolutional neural networks," *International Journal of Innovative Research in Computer and Communication Engineering,* vol. 4, no. 2, pp. 1101-1106, 2016.

[3] Y. Wang *et al.*, "Improvement of MNIST Image Recognition Based on CNN," in *IOP Conference Series: Earth and Environmental Science*, 2020, vol. 428, no. 1: IOP Publishing, p. 012097.

[4] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375,* 2018.

Appendices

1. The code of CNN model in Q1 and Q2

文本

中度可信度描述已自动生成

2. The structure of CNN implemented in the code

图片包含 图表

描述已自动生成

3. The pixels matrix of image in Q1



4. The process of Convolution 2



5. The loss function executed in the code

文本

描述已自动生成

6. The proof process of derivation of softmax



7. The graph of softmax\_cross\_entropy loss



8. The code related to train function

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

9. The code related to test function

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

10. The performance report of CNN framework on MNIST



11. The code related to prediction function

文本

中度可信度描述已自动生成

12. The code of improved CNN model

图形用户界面, 文本

描述已自动生成

13. The LeNet style structure of new CNN model

图表

描述已自动生成

14. The performance report of new CNN model



15. Comparation of performance between new CNN model and previous CNN model

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | New CNN model | | Previous CNN model | |
|  | Average Loss | Accuracy | Average Loss | Accuracy |
| Epoch 1 | 0.0624 | 98.04% | 0.0682 | 97.88% |
| Epoch 2 | 0.0517 | 98.31% | 0.0488 | 98.45% |
| Epoch 3 | 0.0417 | 98.59% | 0.0527 | 98.31% |
| Epoch 4 | 0.0406 | 98.67% | 0.0449 | 98.73% |
| Epoch 5 | 0.0375 | 98.76% | 0.0392 | 98.75% |
| Epoch 6 | 0.0373 | 98.73% | 0.0387 | 98.76% |
| Epoch 7 | 0.0371 | 98.75% | 0.0385 | 98.76% |
| Epoch 8 | 0.0359 | 98.83% | 0.0382 | 98.79% |
| Epoch 9 | 0.0360 | 98.81% | 0.0377 | 98.80% |
| Epoch 10 | 0.0349 | 98.89% | 0.0371 | 98.79% |
| Epoch 11 | 0.0348 | 98.90% | 0.0373 | 98.76% |
| Epoch 12 | 0.0343 | 98.92% | 0.0379 | 98.81% |
| Epoch 13 | 0.0349 | 98.90% | 0.0372 | 98.81% |
| Epoch 14 | 0.0348 | 98.90% | 0.0372 | 98.83% |

16. The performance of new code without ReLU or without Dropout

 

17. The performance of new model without FC2

