Deep Learning

MNIST Digits Classification with CNN

Author: Yantian Luo

Institute: Electronic Engineering

ID: 2018310742

Update: April 17, 2019

Contents

Chapter 1 Introduction

MNIST digits dataset is a widely used database for image classification in machine learning field. It contains 60,000 training samples and 10,000 testing samples. Each sample is a 784×1 column vector, which is transformed from an original 28×28 pixels grayscale image.

In this homework, we will continue working on MNIST digits classification problem by utilizing convolutional neural network (CNN). The main challenge is to implement the forward and backpropagation functions of convolutional layer and pooling layer from scratch! And at last, we compare the the difference of results obtained when working with MLP.

Chapter 2 Algorithm Design

In this homework, we have basis about the Linear layer, Relu layer, Sigmoid layer, EuclideanLoss and SoftmaxCrossEntropyLoss from homework1, thus we don't need to complete their forward and backward functions another time. Therefore, the most important challenge for us is to implement the forward and backpropagation functions of convolutional layer and pooling layer.

2.1 *im2col* operation

This section, we introduce a very important operation for pooling and convolution, that is *im2col*. *im2col* operation rearranges sliding image neighborhoods of kernel size into columns with padding and stride, and returns the concatenated columns in another tensor. Figure 2.1 shows an *im2col* operation on a 2-dimension matrix with kernel-size = (3×3) and padding = 0, stride = 2. From the figure we can find, the column of the output matrix is rearranging sliding image neighborhoods of kernel size.

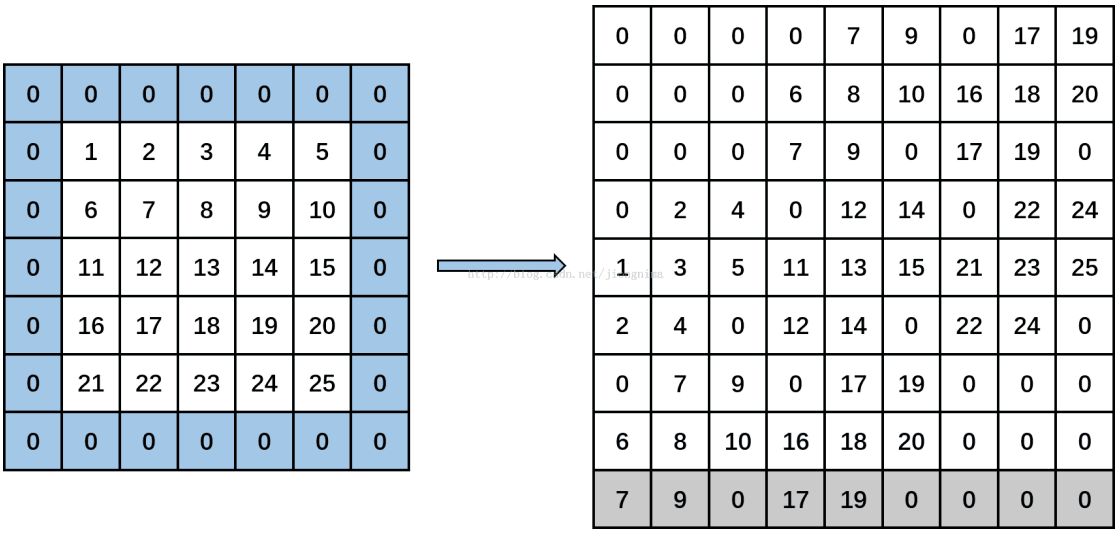

Figure 2.1: 2-dimension *im2col* operation: left is the input matrix, right is the output matrix with kernel-size = (3×3) and padding = 0, stride = 2

2.2 convolution layer

Using $im2col$ operation, we can easily convert convolution operation to tensor multiplication.

2.2.1 Forward

Assuming the input tensor is x with the shape $n \times c_{in} \times h_{in} \times w_{in}$, the weight matrix is W with the shape $c_{out} \times c_{in} \times k \times k$, the bias is b with the shape c_{out}, the kernel_size is k, the pad is p and the output tensor is y with the shape$n \times c_{out} \times h_{out} \times w_{out}$. Using $im2col$ operation we can gey the Forward algorithm as shown in Algorithm 1.

Algorithm 1 Convolution Forward Algorithm

Input: x, W, b, k, p
Output: y
1: $h_{out} = h_{in} + 2 \times p - k + 1$
2: $w_{out} = w_{in} + 2 \times p - k + 1$
3: $x_t = im2col(x)$ and reshape x_t^T into x_m with shape $h_{out} \times w_{out} \times n \times (c_{in} \times k \times k)$
4: Reshape W with shape $c_{out} \times (c_{in} \times k \times k)$
5: Compute $y = x_m \cdot W^T$
6: At last, transpose y into right shape which is $n \times c_{out} \times h_{out} \times w_{out}$

2.2.2 Backward

From the ppt of the class, we can get the gradient and local sensitivity as follow:

$$\begin{aligned} \frac{\partial E^{(n)}}{\partial W_q^{(l)}} &= Y^{(l-1)} *_{valid} rot180(\delta_q^{(l)}) \\ \frac{\partial E^{(n)}}{\partial b^{(l)}} &= \sum_{i,j} \delta_{ij}^{(l)} \\ \delta_p^{(l-1)} &= \triangle_q^{(l)} *_{full} flip_0(\tilde{W}_p^{(l)}) \end{aligned} \tag{2.1}$$

From the equation, we can find that the the gradient of W and local sensitivity is also convolution operation, which can be used Algorithm 1 to solve it. The radient of b is simply add operation which is easy to implement.

2.3 average pooling layer

2.3.1 Forward

For average pooling layer, we can regard it as convolution layer with W is in equation (2.2) and $b = 0, pad, stride = k$, then we can also use Algorithm 1 to implement Average pooling layer forward operation.

$$W = \left[\frac{1}{k \times k}\right]_{c_{in} \times c_{in} \times k \times k} \tag{2.2}$$

2.3.2 Backward

As stated in the course ppt, the backward of average pooling layer satifies equation (2.3).

$$\delta^{(l-1)} = \frac{1}{poolingsize} upsample(\delta^{(l)}) \tag{2.3}$$

The $upsample$ operation can be defined as equation (2.4)

$$upsample(\mathbf{a}) \triangleq \left[\begin{array}{cc|c|cc} a_{11} & a_{11} & \cdots & a_{1m} & a_{1m} \\ a_{11} & a_{11} & \cdots & a_{1m} & a_{1m} \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots \\ \hline a_{n1} & a_{n1} & \cdots & a_{nm} & a_{nm} \\ a_{n1} & a_{n1} & \cdots & a_{nm} & a_{nm} \end{array}\right] \tag{2.4}$$

Chapter 3 Experients and Results

In this chapter, we use CNN to work on MNIST digits classification problem and get the accuracy and loss, and we compare the difference of results obtained when working with MLP in homework1. And at last, we visualize the first convolution layer's output after ReLU for 0-9 digit images.

3.1 Experiments1 (two conv layers + one linear layer)

The network structure in this section is designed as Figure 3.1 and training arguments is as follow:

```
config = {
  'learning_rate': 0.01,
  'weight_decay': 0.0001,
  'momentum': 0.9,
  'batch_size': 100,
  'max_epoch': 100,
  'disp_freq': 50,
}
```

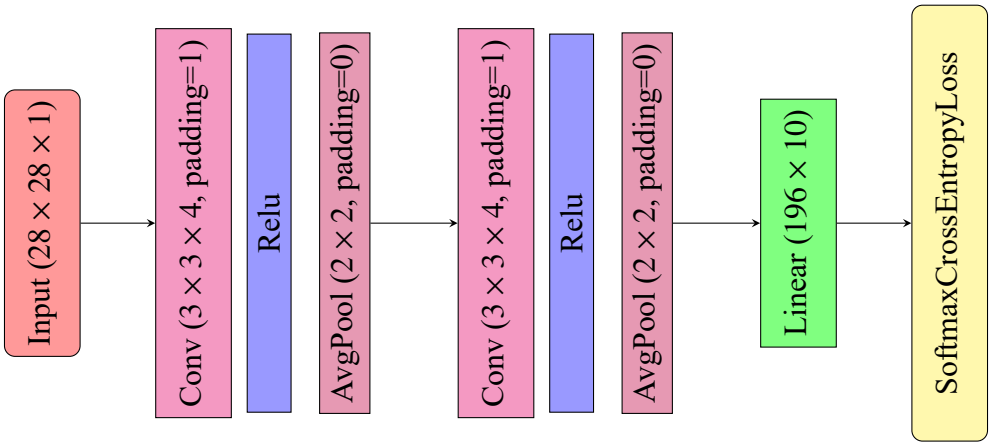

Figure 3.1: Experiments1 Network Structure

In this experiments, the accuracy on test set is 0.983 and the loss on test set is 0.052. We draw the train accuracy curve and train loss curve as shown in Figure 3.2 and we draw

the test accuracy curve and test loss curve with respect to epoch as shown in Figure 3.3

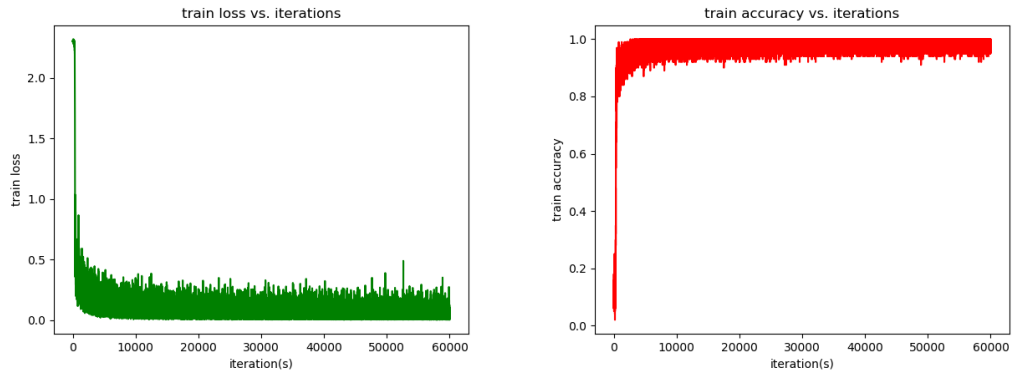

Figure 3.2: train loss curve and train accuracy curve in Experiments1

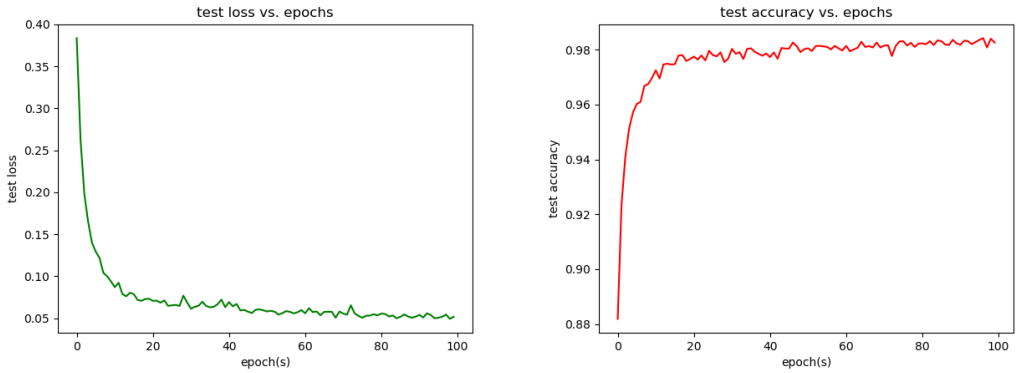

Figure 3.3: test loss curve and test accuracy curve in Experiments1

3.2 Experiments2 (two conv layers + two linear layers)

The network structure in this section is designed as Table**??** and training arguments is as follow:

```
config = {
    'learning_rate': 0.1,
    'weight_decay': 0.0001,
    'momentum': 0.9,
    'batch_size': 100,
    'max_epoch': 100,
    'disp_freq': 50,
}
```

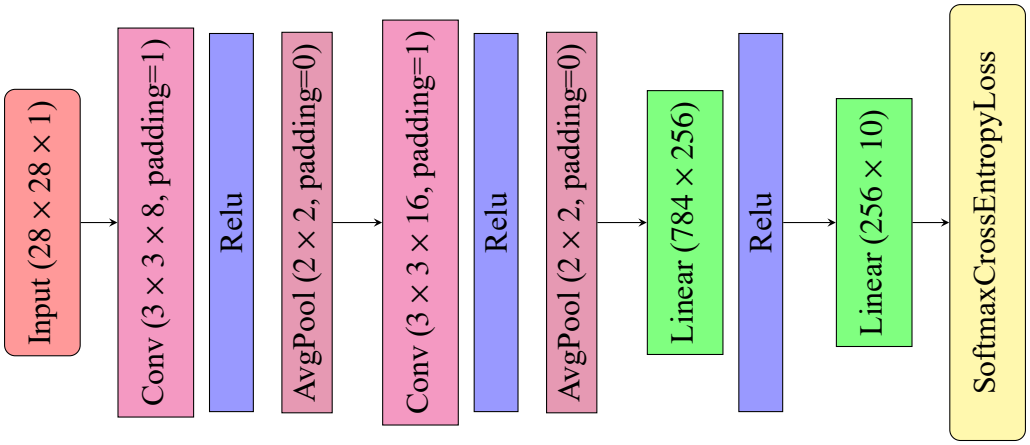

Figure 3.4: Experiments2 Network Structure

In this experiments, the accuracy on test set is up to 0.987 and the loss on test set is 0.048 after only 30 training epochs. Then we draw the train accuracy curve and train loss curve as shown in Figure 3.5 and we draw the test accuracy curve and test loss curve with respect to epoch as shown in Figure 3.6

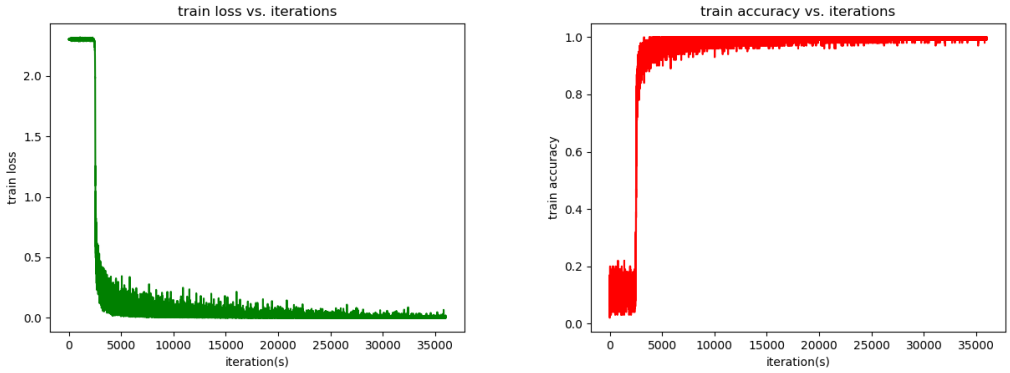

Figure 3.5: train loss curve and train accuracy curve in Experiments2

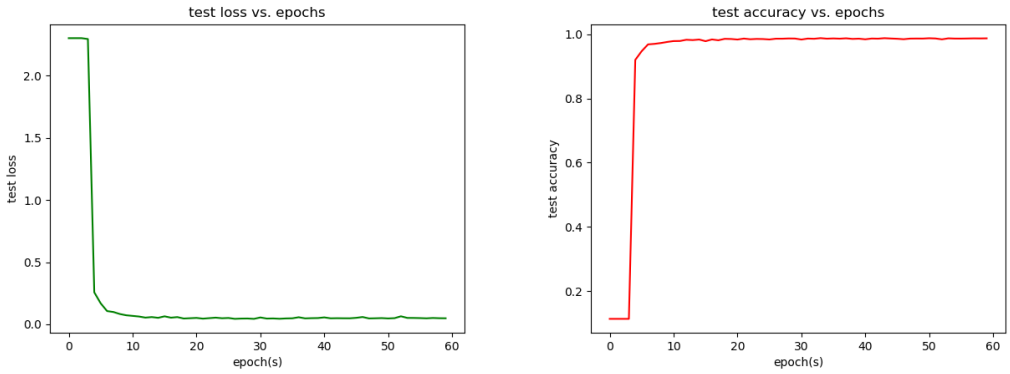

Figure 3.6: test loss curve and test accuracy curve in Experiments2

3.3 Comparisions

We summarize all test accuracys and test losses of every experiments and compare to the best experiment in the last homework as shown in Table 3.1. From the results we can get the following conclusions:

- CNN has larger test accuracy than MLP, and from the experiments, we find that CNN has faster convergence speed than MLP. But CNN also has larger test loss than MLP. The reason may be that local interconnectivity of CNN results in losing some global information, but the information is not important for classification.
- Compare Experiment1 with Best experiment in HW1 we can find that CNN has less parameters than MLP with equal accuracy. And less parameters is not easy to result in overfitting. Therefore, CNN is much better than MLP in this task.
- For training time, MLP has less time than CNN in one epoch. The reason may be convolution operation has more time than Linear operation.

Table 3.1: Results Summary

Cases	Test Accuracy	Test Loss	Parameters Number
Experiment1	0.983	0.052	2143
Experiment2	0.987	0.048	203,628
Best experiment in HW1	0.983	0.021	109,187

3.4 First convolution layer's output visualization

Here we use the network in Experiment2 as shown in Figure 3.4 to visualize the first convolution layer's output after ReLU for 0-9 digit images. We draw the output in each channel as shown in Figure 3.7

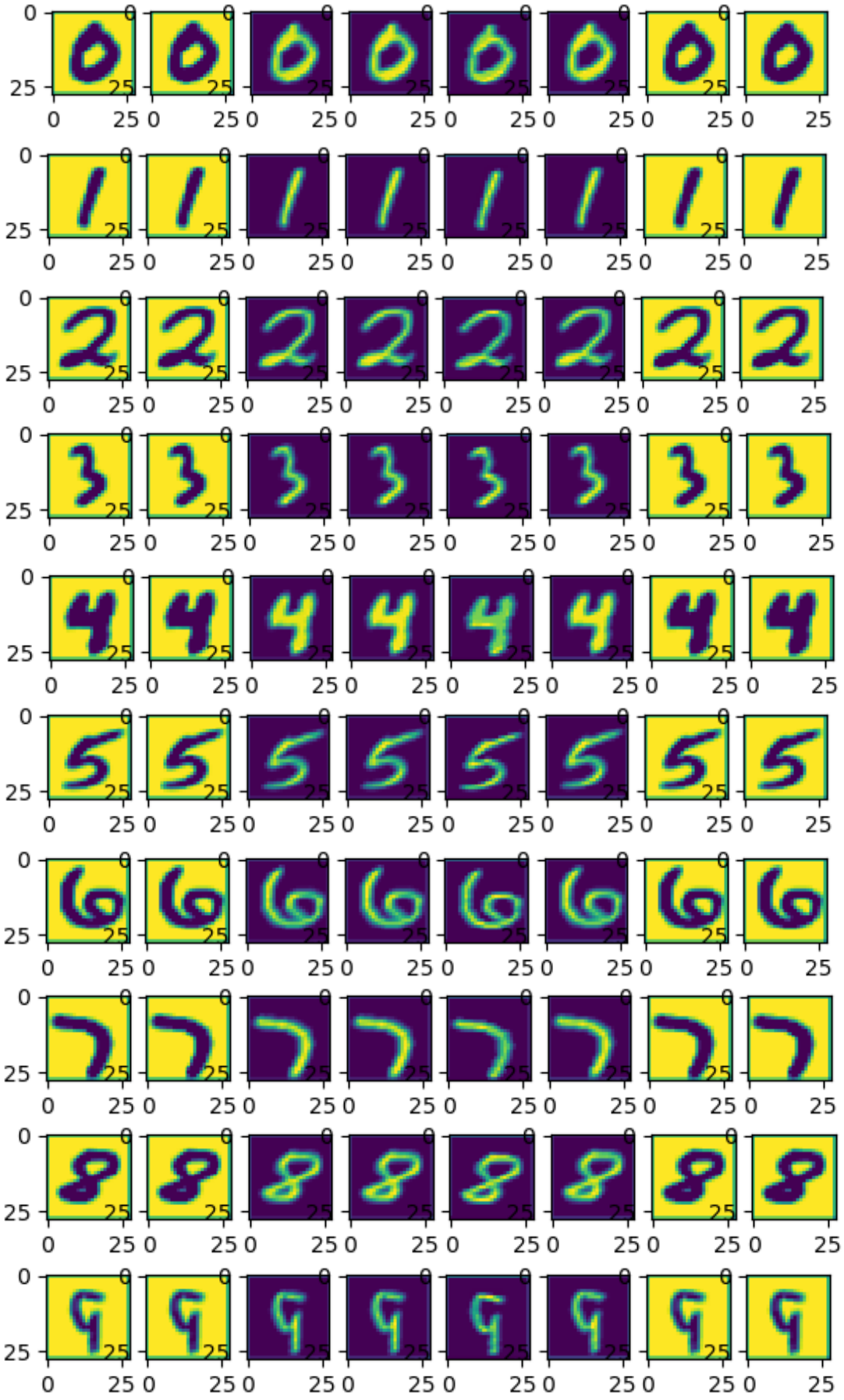

Figure 3.7: The visualization of the first convolution layer's output after ReLU for 0-9 digit images in Experiments2 (8 channels)