COMP3314 Assignment2 Report

Name: Hu Zhenwei UID: 3035533719

1. Introduction:

This assignment implements a modified version of LeNet on the dataset MNIST. The implementation focuses on forward and backward propagations of convolutional layers, activation layers, fully connected layers and max-pooling layers. Some modifications are made to promise the actual performance of the model. There are also some implementation details and limitations to be discussed in later sections.

2. Model Structure and forward propagation:

The modified LeNet contains 3 convolutional layers C_1, C_3, C_5, 2 max-pooling layers S_2, S_4, 4 activation layers A_2, A_4, A_5, A_6 all using ReLu activation function, and 2 fully connected layers F_6, F_7. A loss layer L_8 using softmax cross-entropy loss function is appended to the end of the model.

The input has a dimension $n \times 1 \times 32 \times 32$, where n is the number of samples. Then the forward propagation goes through the following layers:

C_1: Input: $n \times 1 \times 32 \times 32$	Layer Dimension: $6 \times 3 \times 5 \times 5$	Output: $n \times 6 \times 28 \times 28$
S_2: Input: $n \times 6 \times 28 \times 28$	Layer Dimension: None	Output: $n \times 6 \times 14 \times 14$
A_2: Input: $n \times 6 \times 14 \times 14$	Layer Dimension: None	Output: $n \times 6 \times 14 \times 14$
C_3: Input: $n \times 6 \times 14 \times 14$	Layer Dimension: $16 \times 6 \times 5 \times 5$	Output: $n \times 16 \times 10 \times 10$
S_4: Input: $n \times 16 \times 10 \times 10$	Layer Dimension: None	Output: $n \times 16 \times 5 \times 5$
A_4: Input: $n \times 16 \times 5 \times 5$	Layer Dimension: None	Output: $n \times 16 \times 5 \times 5$
C_5: Input: $n \times 16 \times 5 \times 5$	Layer Dimension: $120 \times 16 \times 5 \times 5$	Output: $n \times 120 \times 1 \times 1$
A_5: Input: $n \times 120 \times 1 \times 1$	Layer Dimension: None	Output: $n \times 120 \times 1 \times 1$
F_6: Input: $n \times 120$	Layer Dimension: 120×84	Output: $n \times 84$
A_6: Input: $n \times 84$	Layer Dimension: None	Output: $n \times 84$
F_7: Input: $n \times 84$	Layer Dimension: 84×10	Output: $n \times 10$
L_8: Input: $n \times 10$	Layer Dimension: None	Output: 1

3. Forward propagation:

Most of the forward propagations are covered in the lectures. One thing to mention is the loss layer L_8. For the given $n \times 10$ matrix, It first transform it to corresponding probability matrix by:

$$p_{ij} = \frac{e^{a_{ij}}}{\sum_{k=1}^{10} e^{a_{ik}}} = \frac{C_i e^{a_{ij}}}{C_i \sum_{k=1}^{10} e^{a_{ik}}} = \frac{e^{a_{ij}+\log(C_i)}}{\sum_{k=1}^{10} e^{a_{ik}+\log(C_i)}}$$

where we take $\log(C_i) = \max\{e_{a_{ik}}\}$ to avoid the exponent overflow.
Then the cross-entropy is given by:

$$L = -\sum_{i=1}^{n} \log(p_{ij}), j = y_i$$

where y_i is the correct label of the i-th sample.

4. Backward propagation:

4.1. Loss layer:

From section 3 we have: $L = -\sum_{i=1}^{n} \log(p_{ij}), j = y_i$. Therefore, since $p_{i,y_i} = \frac{e^{a_{i,y_i}}}{\sum_{k=1}^{10} e^{a_{ik}}}$, we have:

$$\frac{\delta L}{\delta a_{ij}} = \frac{\delta - \log(p_{i,y_i})}{\delta a_{ij}} = \frac{\delta - \log(p_{i,y_i})}{\delta a_{ij}} = \frac{\delta - a_{i,y_i}}{\delta a_{ij}} + \frac{e^{a_{ij}}}{\sum_{k=1}^{10} e^{a_{ik}}} = -\mathbf{1}(y_i = j) + p_{ij}$$

4.2. Fully connected layer:

The forward propagation of fully connected layer is given by: $XW + \mathbf{b} = Y$
For example, in F_7, the dimensions of the parameters are:

$$X : n \times 84;\ W : 84 \times 10;\ \mathbf{b} : 1 \times 10;\ Y : n \times 10$$

Firstly, to derive the derivative with regard to X, we notice that:

$$\frac{\delta L}{\delta X_{ij}} = \sum_{k,l} \frac{\delta L}{\delta Y_{kl}} \frac{\delta Y_{kl}}{\delta X_{ij}} = \sum_{l} \frac{\delta L}{\delta Y_{il}} \frac{\delta Y_{il}}{\delta X_{ij}} = \sum_{l} \frac{\delta L}{\delta Y_{il}} W_{jl}$$

Which can be expressed as:

$$\frac{\delta L}{\delta X} = \frac{\delta L}{\delta Y} W^T$$

Secondly, to derive the derivative with regard to W, we notice that:

$$\frac{\delta L}{\delta W_{ij}} = \sum_{k,l} \frac{\delta L}{\delta Y_{kl}} \frac{\delta Y_{kl}}{\delta W_{ij}} = \sum_{k} \frac{\delta L}{\delta Y_{kj}} \frac{\delta Y_{kj}}{\delta X_{ij}} = \sum_{k} \frac{\delta L}{\delta Y_{kj}} X_{ki}$$

Which is equivalent to:

$$\frac{\delta L}{\delta W} = X^T \frac{\delta L}{\delta Y}$$

Lastly, to derive the derivative with regard to $\mathbf{b}$, we have:

$$\frac{\delta L}{\delta \mathbf{b}_k} = \sum_{i,j} \frac{\delta L}{\delta Y_{ij}} \frac{\delta Y_{ij}}{\delta \mathbf{b}_k} = \sum_{i} \frac{\delta L}{\delta Y_{ik}} \frac{\delta Y_{ik}}{\delta \mathbf{b}_k} = \sum_{i} \frac{\delta L}{\delta Y_{ik}}$$

4.3. Max-Pooling layer:

Since we are using a stride and pooling size = 2, we can express the derivative as:

$$\frac{\delta L}{\delta X_{kl}} = \sum_{i,j} \frac{\delta L}{\delta Y_{ij}} \frac{\delta Y_{ij}}{\delta X_{kl}} = \sum_{i,j} \frac{\delta L}{\delta Y_{ij}} \mathbf{1}(X_{kl} = \max\{Y_{i:i+2,j:j+2}\})$$

4.4. Convolutional layer:

Take the convolutional layer C_5 as an example, the dimensions of the parameters are:

$$X : n \times 16 \times 5 \times 5;\ W : 120 \times 16 \times 5 \times 5;\ \mathbf{b} : 1 \times 120 \times 1 \times 1;\ Y : n \times 120 \times 1 \times 1$$

4.4.1. Derivation of $\frac{\delta L}{\delta X}$

$$\frac{\delta L}{\delta X_{n_1,d_1,i_1,j_1}} = \sum_{n_2,d_2,i_2,j_2} \frac{\delta L}{\delta Y_{n_2,d_2,i_2,j_2}} \frac{\delta Y_{n_2,d_2,i_2,j_2}}{\delta X_{n_1,d_1,i_1,j_1}} = \sum_{d_2,i_2,j_2} \frac{\delta L}{\delta Y_{n_1,d_2,i_2,j_2}} \frac{\delta Y_{n_1,d_2,i_2,j_2}}{\delta X_{n_1,d_1,i_1,j_1}} = \sum_{d_2,i_2,j_2} \frac{\delta L}{\delta Y_{n_1,d_2,i_2,j_2}} W_{d_2,d_1,i_1-i_2*s,j_1-j_2*s}$$

if $i_1 \in [i_2 * s, i_2 * s + f\,], j_1 \in [\,j_2 * s, j_2 * s + f\,]$, where s is the stride and f is the size of the filter, or kernel.

To simplify the computation, within the scope of this assignment, we can assume that $s = 1$ and there is no padding.

Therefore, observing that the sum is in the opposite direction of the last 2 axes of W. We can take the flipped version of W, say W_f, where $W_f = W[:, :, ::-1, ::-1]$, and the above formula turns out to be a normal convolution operation where W_f is sliding through $\frac{\delta L}{\delta Y}$.

Notice that, to complete the convolution in practice, we need to pad $\frac{\delta L}{\delta Y}$ to make sure the size is appropriate.

To derive the padding size, assuming X, Y are all square matrices in last two axes, we denote the shape of the last two axes of X to be $i_1 \times i_1$, the shape of Y to be $i_2 \times i_2$.
We have: $i_1 - f + 1 = i_2$ since $s = 1$, $pad = 0$. After padding and convolution: $i_2 + 2p - f + 1 = i_1$.
Hence $p = f - 1$, we need to pad $f - 1$ to both side of $\frac{\delta L}{\delta Y}$.

4.4.2. Derivation of $\frac{\delta L}{\delta W}$

$$\frac{\delta L}{\delta W_{n_1,d_1,i_1,j_1}} = \sum_{n_2,d_2,i_2,j_2} \frac{\delta L}{\delta Y_{n_2,d_2,i_2,j_2}} \frac{\delta Y_{n_2,d_2,i_2,j_2}}{\delta W_{n_1,d_1,i_1,j_1}} = \sum_{n_2,i_2,j_2} \frac{\delta L}{\delta Y_{n_2,d_1,i_2,j_2}} \frac{\delta Y_{n_2,d_1,i_2,j_2}}{\delta W_{n_1,d_1,i_1,j_1}} = \sum_{n_2,i_2,j_2} \frac{\delta L}{\delta Y_{n_2,d_1,i_2,j_2}} X_{d_2,d_1,i_1+i_2*s,j_1+j_2*s}$$

if $i_1 \in [i_2 * s, i_2 * s + f], j_1 \in [j_2 * s, j_2 * s + f]$

Similarly, this is equivalent to a convolution where $\frac{\delta L}{\delta Y}$ is sliding through X.
There is no need to flip or pad any matrix this time as this is a normal convolution.

4.4.3. Derivation of $\frac{\delta L}{\delta \mathbf{b}}$

$$\frac{\delta L}{\delta \mathbf{b}_{n_1,d_1,i_1,j_1}} = \sum_{n_2,d_2,i_2,j_2} \frac{\delta L}{\delta Y_{n_2,d_2,i_2,j_2}} \frac{\delta Y_{n_2,d_2,i_2,j_2}}{\delta \mathbf{b}_{n_1,d_1,i_1,j_1}} = \sum_{n_2,i_2,j_2} \frac{\delta L}{\delta Y_{n_2,d_1,i_2,j_2}} \frac{\delta Y_{n_2,d_1,i_2,j_2}}{\delta \mathbf{b}_{n_1,d_1,i_1,j_1}} = \sum_{n_2,i_2,j_2} \frac{\delta L}{\delta Y_{n_2,d_1,i_2,j_2}}$$

4.5. Activation layer:

Since we are using only ReLu in this assignment. The derivative of ReLu can be easily given by

$$\frac{\delta L}{\delta X_{n,d,i,j}} = \frac{\delta L}{\delta Y_{n,d,i,j}} \mathbf{1}(X_{n,d,i,j} > 0)$$

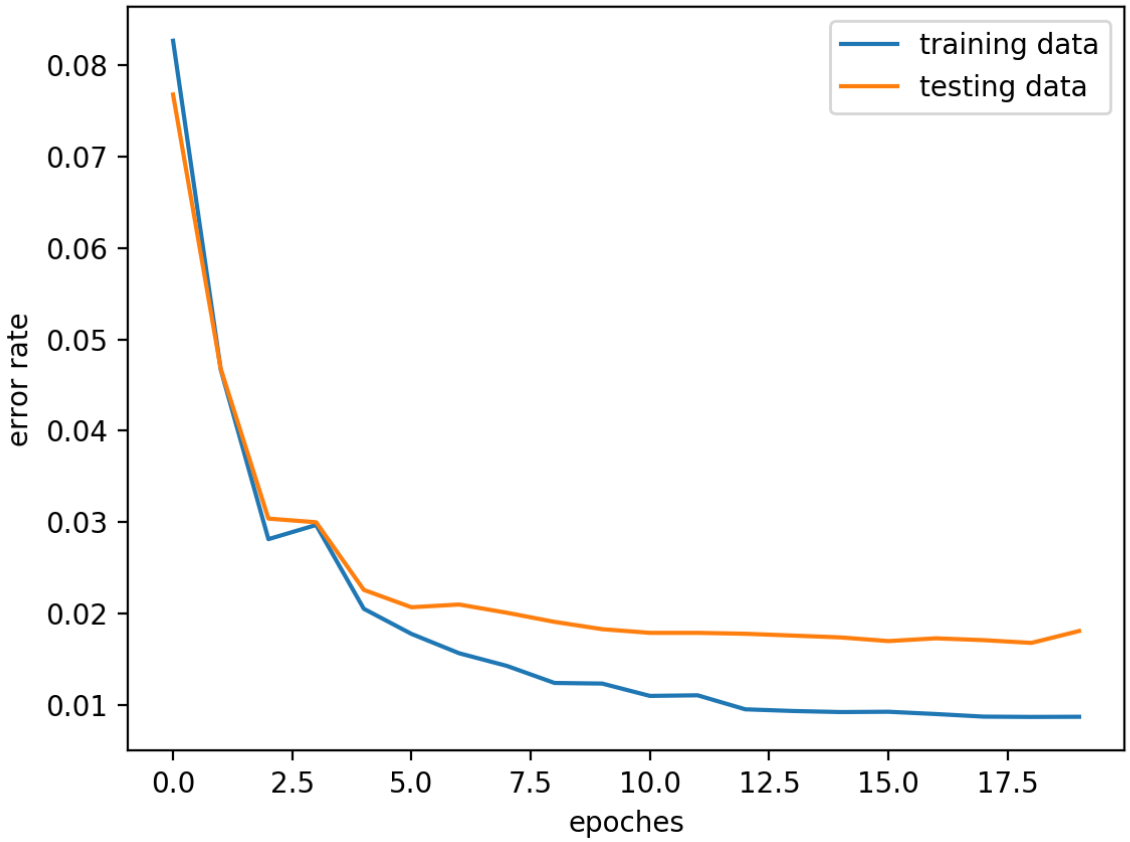
5. Hyper-parameters and training result:

In my implementation, I multiply all the global learning rates by 0.015. Otherwise, after 2-3 iterations, the weights rocketed to NaN, causing the program to crash. Besides from modifying the global learning rate, I applied a different scheme of weight initialization where

$$W = np.random.uniform(\frac{-2.4}{n_{in}}, \frac{2.4}{n_{in}})$$

Since I observed that there is a very high possibility (about 13/14) that the training would not converge, which is a problem caused by weight initialization. Therefore this more stable version of weight initialization is chosen, and the training result is converging if I run the program about 6-7 times. I further reduced the global learning rates by changing 0.015 to 0.008, and the converging rate is increasing dramatically to more than 50%, and the whole program is much more stable. A retrain mechanism is also applied, when the program observes that there is no sign of convergence after 60 iterations, it will automatically reinitialize the weights and restart the current epoch.

I choose one of the successful trained model (trained by global learning rate *= 0.015) and obtain the error rate curve during training, as follows:



From the above figure, it can be inferred that the trained model is getting a very high accuracy of the training data, and a relatively low accuracy for testing data. I also noticed that the accuracy for testing data nearly stopped to increase after 10 epoches or so. It is likely that this model suffers a bit from overfitting problem but shows a generally good performance, to solve this problem, we may consider decreasing the learning rate or training the model for only 10 epoches.

6. Limitations and further improvement:

One limitation lies in the implementation of convolutional layers' back propagation. I assume the stride = 1 and padding = 0, this largely simplifies the expression of the derivative. To make the expression more general, we can have extra padding between elements to consider the effects of stride. And it turns out the result is the same as one without extra stride or padding.

Another aspect to improve is that in the back propagation of the max-pooling layer, I assume there are only one maximum value inside every slice of the input matrix, which is not always true. However, in practice, the possibility of having more than 1 maximum values in a 2×2 slice of the input matrix is very small. Therefore, to speed up the implementation by avoiding naive for-loops, I distribute the derivatives to all the maximum values within each slice, even though there might be more than 1 maximum values.

There are also much space for improvement, for example, by tuning the learning rates, weight initialization schemes, we may get better results in terms of accuracy or have a larger possibility to have the model converge. Some other aspects can be considered, such as the momentum parameter in training, which has already been included in my implementation, or taking the weight regularization into consideration which might help solve the problem of weight explosion during training.

7. Summary:

In conclusion, this CNN model performs as expected, though using much smaller learning rates. It also has a short training time (about 1 hour only) since the implementation avoids for-loops. The convergence of the model is however not satisfactory, having a relatively small possibility of converging. Further improvements can be applied to increase the performance of this model, as well as its generality to accept different values of hyper-parameters such as stride and padding.

8. Reference:

https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d

http://machinelearningmechanic.com/deep_learning/2019/09/04/cross-entropy-loss-derivative.html

https://blog.csdn.net/tsyccnh/article/details/79163834

https://cs231n.github.io/neural-networks-3/#sgd

https://weizhixiaoyi.com/archives/227.html

https://medium.com/the-bioinformatics-press/only-numpy-understanding-back-propagation-for-max-pooling-layer-in-multi-layer-cnn-with-example-f7be891ee4b4

https://mc.ai/backpropagation-for-convolution-with-strides/