Deep Learning 2020 **(Due: [18.11])**

Submission Assignment #[2]

Instructor: Jakub Tomczak *Name:* [Xingkai Wang], *Netid:* [xwg510]

The following section layout is a suggestion for the structure of the report. If it doesn't quite fit, you can use whatever structure does work.

1 Question answers

Question 1

$$\frac{\partial f}{\partial X} = \frac{1}{Y} \tag{1.1}$$

$$\frac{\partial f}{\partial Y} = -\frac{X}{Y^2} \tag{1.2}$$

Question 2

Assume h(X) is the output of F(X), when we do the backward of F, the chain rule is used. $\frac{\partial l}{\partial h}$ is the gradient of the loss with respect to the output, and since the function F(X) applies f element-wise, therefore $\frac{\partial h}{\partial F}$ is actually the gradient of f, called f'.

$$\frac{\partial l}{\partial F} = \frac{\partial l}{\partial h} \cdot \frac{\partial h}{\partial F} \tag{1.3}$$

Question 3

Assume $C = W \cdot X$, and y = f(C), f is the function produce the output. Therefore, we can calculate the gradient by apply the chain rule.

$$\frac{\partial y}{\partial A_{pq}} = \frac{\partial y}{\partial C_{ij}} \cdot \frac{\partial C_{ij}}{\partial A_{pq}} \tag{1.4}$$

By matrix multiplication, we know:

$$C_{ij} = A_{ih} \cdot B_{hj} \tag{1.5}$$

Therefore:

$$\frac{\partial C_{ij}}{\partial A_{pq}} = \begin{cases} B_{qj} & \text{i = p} \\ 0 & \text{i}\neq\text{p} \end{cases} \tag{1.6}$$

Now we insert this result to the equation above:

$$\frac{\partial y}{\partial A_{pq}} = \sum_{ij} \frac{\partial y}{\partial C_{ij}} \cdot \frac{\partial C_{ij}}{\partial A_{pq}} = \sum_{j} \frac{\partial y}{\partial C_{pj}} \cdot \frac{\partial C_{pj}}{\partial A_{pq}} = \sum_{j} \frac{\partial y}{\partial C_{pj}} \cdot B_{qj} = \frac{\partial y}{\partial C_{pj}} \cdot B_j{}_q^T \tag{1.7}$$

Finally, we can get the gradient:

$$\frac{\partial y}{\partial A_{pq}} = \frac{\partial y}{\partial C} \cdot B^T \tag{1.8}$$

$$\frac{\partial y}{\partial B_{pq}} = A^T \cdot \frac{\partial y}{\partial C} \tag{1.9}$$

Question 4

Due to the matrix Y consisting of 16 columns that are all equal to x, therefore in this case j is equal to 16, the gradient of f is :

$$\frac{\partial Y}{\partial x} = \sum_{j} \frac{\partial Y_j}{\partial x} = \sum_{j} \frac{\partial x}{\partial x} = j = 16 \tag{1.10}$$

Question 5

1. c.values contains the result of value of tensor node a plus value of tensor node b.
2. c.source refers to the property of c is a Opnode object.
3. c.source.input.value refers to the value of tensor node a
4. a.grad refers to the gradient of tensor a. The current value is 0.

Question 6

1. The class object defines the operation.
2. In the 280 line of code, the add class is the actual addition performed
3. Because the forward process should be executed and then the output value will be obtained. In the 212 line of code, the Opnode is connected to the output node.

Question 7
In the line 82 of code, the backward functions of the relevant Ops is being called.

Question 8
The Expand Op is not a element wise operation but a vector broadcasting operation. The forward process is the simply broadcasting for given dimension. We assume that a number of inputs along a given dimension is repeated N times, so that during backward, we need sum all gradients and times the gradient from previous layer.

$$\frac{\partial Y}{\partial x} = goutput \cdot \sum_N \frac{\partial Y_j}{\partial x} = goutput \cdot \sum_N 1 = \sum_N goutput \tag{1.11}$$

Question 10
Table 1 presents the accuracy on the test data with different combination of hyperparameters. After several experiments, 0.001 is the best value of learning rate. 0.0001 is too small even through we set the number of epochs to 10, it only can get 58% accuracy, and 0.01 is too high so it can only have 24% accuracy. Therefore the 0.001 of learning rate will be chosen. Then the value of momentum is tested, the results show that the neural network has the best performance with 0.9 momentum, this value shows that if the direction of gradient descent in current time is much similar with the direction in previous time, then the decline will increase. Finally, the number of epochs is tested, 10 epochs has the highest accuracy which is 61%, however it takes much longer time when training the neural network and only 1% higher than when we have number of epochs is 5.

$learning_rate$	momentum	epoch	accuracy
0.0001	0.9	2	27%
0.0001	0.9	10	58%
0.001	0.9	2	52%
0.01	0.9	2	24%
0.001	0.7	2	31%
0.001	0.5	0.5	29%
0.001	0.9	5	60%
0.001	0.9	10	61%

Table 1: test accuracy with different hyperparameters

2 Problem statement

Sigmoid active function(sigmoid) and ReLu active function(relu) are the most common active function being used in the neural network. The vanishing gradient problem is an important issue at training time on multilayer neural networks using the backpropagation algorithm. This problem is worse when sigmoid transfer functions are used, in a network with many hidden layers because the gradient of sigmoid function is close to 0(1). But

the relu function can avoid this problem happening. In this research, we will Compare the validation accuracy of the neural network with sigmoid function and with relu function.

$$sigmoid = 1/(1 + e^{-x}) \tag{2.1}$$

$$relu = max(0, x) \tag{2.2}$$

3 Methodology

The initial approach of this research was implementing a Op class for relu active function which contains the forward process and backward process. In forward process, the relu function make all inputs x that smaller than 0 equal to 0 and other inputs x larger than 0 equal to itself. In the backward process, the gradient of all x which are smaller than 0 equal to 0 and other x which are larger than 0 equal to 1. Figure 1 presents the code of implementation of Op class for ReLu function.

$$drelu = \begin{cases} 0 & \text{x} \leq 0 \\ 1 & \text{x} > 0 \end{cases} \tag{3.1}$$

```
class ReLu(Op):
    '''
    Op for ReLu function
    '''

    @staticmethod
    def forward(context,input):
        Relux = np.maximum(0,input)
        context['Relux'] = Relux
        return Relux

    @staticmethod
    def backward(context, goutput):
        Relux = context['Relux']
        Relux[Relux <= 0] = 0
        Relux[Relux > 0] = 1
        return goutput * Relux
```

Figure 1: implementation of ReLu class

4 Experiments

In this experiment, the synth data is used to train and validate the neural network. During the training of 20 epochs, the validation accuracy and total loss after each epoch training are recorded separately. The learning rate is being set to 0.001. In our expectation, the performance of neural network with relu active function should be better than the neural network with sigmoid function.

5 Results and discussion

Figure 2 and figure 3 illustrate the running loss and validation accuracy after each training epoch with sigmoid active function and relu active function. The first very significant result can be observed: the validation accuracy is higher after each training epoch when the active function is relu function. This result meets our expectations,

because the relu function can effectively avoid the vanishing gradient problem during the backpropagation process, which improves the efficiency of neural network learning. And the second result can be obtained is that when the sigmoid function is used, from training epoch 2 to epoch 9 the validation accuracy stays the same, it seems that neural network does not learn anything is this period of time, the reason why this happens is because when doing backward process, the sigmoid function may cause vanishing gradient problem, in this case, the parameters will update slowly or even stop updating. However, the gradient of relu function is equal to 1 when x is larger than 0, that will the backward process much more stable.

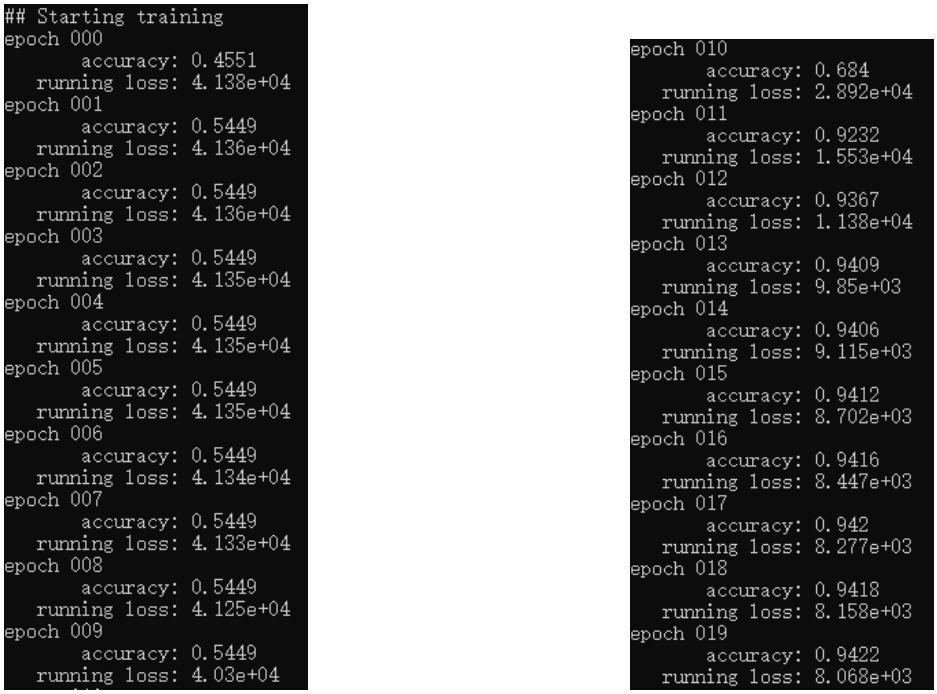

Figure 2: running loss and validation accuracy with sigmoid function

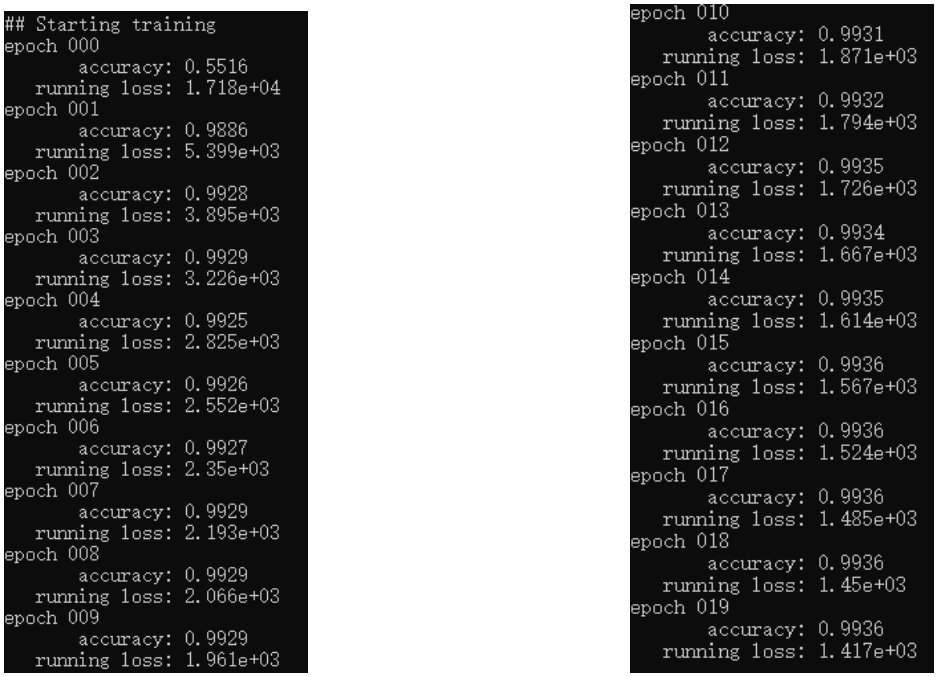

Figure 3: running loss and validation accuracy with relu function

References

Matfas Roodschild (2020) "A new approach for the vanishing gradient problem on sigmoid activation", Progress in Artificial Intelligence, 9, 351–360