DLP HW1

310551053 Chieh-Ming Jiang

July 2021

I Introduction

In this assignment, I am going to implement the backpropagation from the scratch. Backpropagation is a simple method to train a neural network. In my network architecture, there are two inputs, two hidden layers and the output for prediction. I am using forward function to predict the output and using backward function to update the weights of the neural network.

II Experiment setups

1. Sigmoid functions

The sigmoid function is $\sigma(x) = \frac{1}{1+e^{-x}}$, and its first derivative is $\sigma'(x) = \frac{\sigma(x)}{1-\sigma(x)}$.

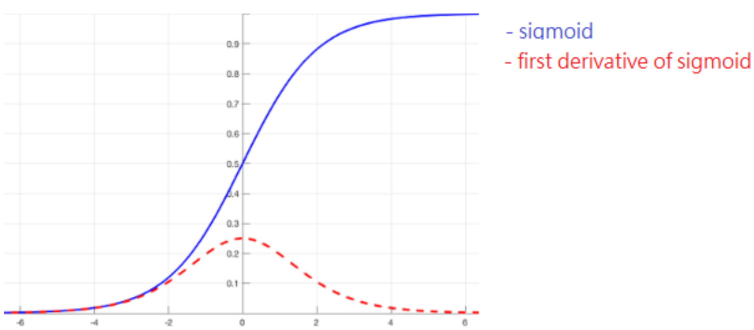

Figure 1: activation function

The sigmoid function is used as activation function in this assignment. The characteristic of sigmoid function is always limiting the value between 0 and 1. After calculating the linear combination of input value X of i-th neuron with weight W_i, we can get the output $z_i = XW_i$. Then pass this value to sigmoid function $\sigma(z_i)$ as the input value of the neurons in next layer.Repeat

the task until getting the output value. Because it's a binary classification problem, we can classify the data point depending on the output value is bigger than 0.5 or not.

2. **Neural network**
   For both of two neural networks, I use two hidden layers, and 8 neurons for each hidden layers. The network is fully connected and the structure is shown as below.

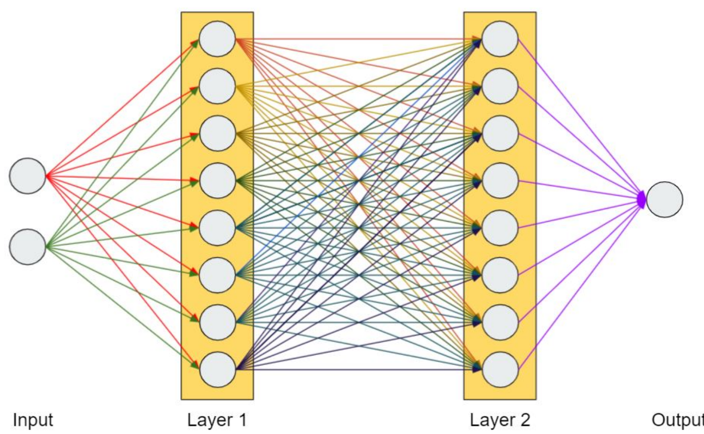

Figure 2: Network architecture

3. **Backpropagation**
   In order to update the parameters of the model, we need to compute the gradient $\frac{\partial C}{\partial W_i}$. Moreover, backpropagation is an efficient way to compute gradient. I will explain how it works with a simple example.

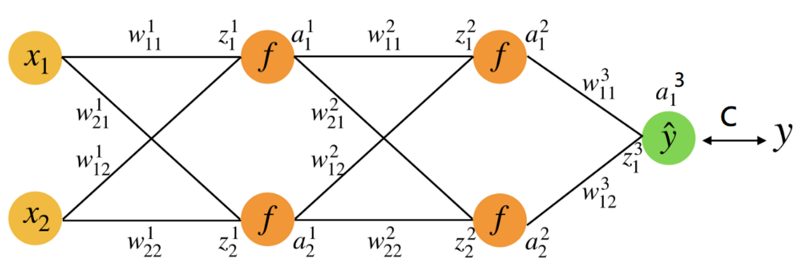

Figure 3: simplified network

In this assignment, I use MSE as error function $C = (\hat{y} - y)^2$. If

using gradient descent to update weights, the formula will be

$$\nabla C(\theta) = \begin{bmatrix} \partial C/\partial W^1_{11} \\ \partial C/\partial W^1_{12} \\ \vdots \\ \partial C/\partial W^3_{12} \end{bmatrix}$$

So let's calculate the terms.

$$z_1^1 = W_{11}^1 \cdot x_1 + W_{12}^1 \cdot x_2$$

$$\frac{\partial C}{\partial W_{11}^1} = \frac{\partial z_1^1}{\partial W_{11}^1} \frac{\partial C}{\partial z_1^1} = x_1 \cdot \left(\frac{\partial a_1^1}{\partial z_1^1} \cdot \frac{\partial C}{\partial a_1^1} \right)$$

$$= x_1 \times \sigma'(z_1^1) \left[\frac{\partial z_1^2}{\partial a_1^1} \frac{\partial C}{\partial z_1^2} + \frac{\partial z_2^2}{\partial a_1^1} \frac{\partial C}{\partial z_2^2} \right]$$

$$= x_1 \times \sigma'(z_1^1) \left[W_{11}^2 \frac{\partial C}{\partial z_1^2} + W_{21}^2 \frac{\partial C}{\partial z_2^2} \right]$$

input

$$z_1^2 = W_{11}^2 \cdot a_1^1 + W_{12}^2 \cdot a_2^1$$

$$\frac{\partial C}{\partial W_{11}^2} = \frac{\partial z_1^2}{\partial W_{11}^2} \frac{\partial C}{\partial z_1^2} = a_1^1 \cdot \left(\frac{\partial a_1^2}{\partial z_1^2} \cdot \frac{\partial C}{\partial a_1^2} \right)$$

$$= a_1^1 \times \sigma'(z_1^2) \left(\frac{\partial z_1^3}{\partial a_1^2} \frac{\partial C}{\partial z_1^3} \right)$$

$$= a_1^1 \times \sigma'(z_1^2) \left(W_{11}^3 \frac{\partial C}{\partial z_1^3} \right)$$

input

$$z_1^3 = W_{11}^3 \cdot a_1^2 + W_{12}^3 \cdot a_2^2, \quad C = (a_1^3 - y)^2$$

$$\frac{\partial C}{\partial W_{11}^3} = \frac{\partial z_1^3}{\partial W_{11}^3} \frac{\partial C}{\partial z_1^3} = a_1^2 \cdot \left(\frac{\partial a_1^3}{\partial z_1^3} \cdot \frac{\partial C}{\partial a_1^3} \right)$$

$$= a_1^2 \cdot \sigma'(z_1^3) \times 2(a_1^3 - y)$$

input

Figure 4: Backpropagation derivation

Observing from each equations above, we could find that $\partial C/\partial z^i$

is the linear combination of $\partial C/\partial z^{i+1}$ multiplies $\sigma'(z^i)$, which is shown as below.

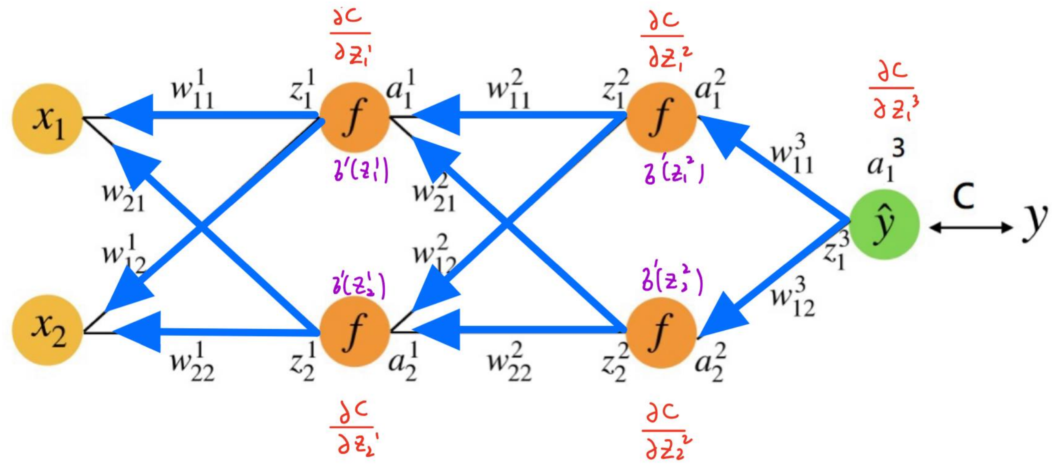

Figure 5: Partial derivative

For example,

$$\partial C/\partial z_1^1 = \sigma'(z_1^1) * (w_{11}^2 * \partial C/\partial z_1^2 + w_{21}^2 * \partial C/\partial z_2^2)$$

If we calculate the partial derivative backwardly, we could get all the partial derivative value efficiently, which is the most part of backpropagation. Moreover, observing from the Figure.4, we could find the gradient is the partial derivative multiplies the input of the neuron. Then we can update the model parameters. Take w_{11}^1 as example,

$$w_{11}^1 -= lr * x_1 * \partial C/\partial z_1^1$$

Last but not least, we could update all the weights in this way until it converges.

III Results of your testing

1. **Screenshot and comparison figure**

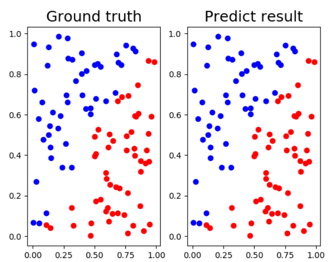

Figure 6: linear result

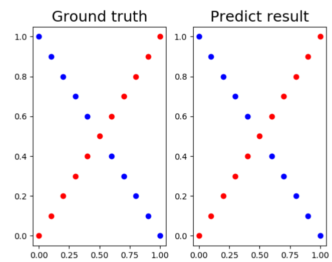

Figure 7: xor result

2. Show the accuracy of your prediction

```
0.010328886815399327
0.008531134998783175
0.896544106660533
0.9919445358163402
0.9915291101275308
0.008262446776484
0.009386763711470191
0.010687487645907396
0.9895579914604112
0.9917535963646571
0.9905576775011506
0.9725037877828612
0.01459163369714962
0.9912605767619276
0.009217159653804052
0.012176705477023342
0.990657545913458
0.9903236253002229
0.00991653424461069
0.9907999394789483
0.9853954824569804
0.012440079674953735
Accuracy:  100.0 %
```

Figure 8: linear accuracy

```
0.12469353468567448
0.9337104638528066
0.09291249809368547
0.9243353615151322
0.07054911283387014
0.9138597488434843
0.056351510068625016
0.901713565439122
0.048304557910537116
0.8557494263539991
0.04469878244258596
0.044378834854928724
0.9712028579931401
0.04662213397340344
0.9975132656248691
0.05097748759622642
0.9993360277191011
0.05717816577283638
0.9996138756736724
0.06511987106761799
0.9996550279463754
Accuracy:  100.0 %
```

Figure 9: xor accuracy

3. Learning curve (loss, epoch curve)

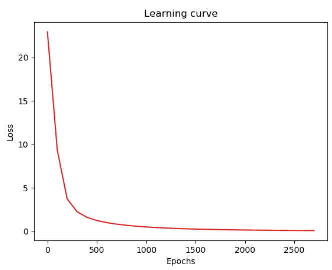

Figure 10: linear loss

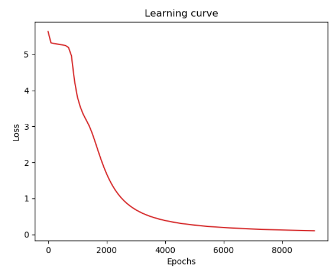

Figure 11: xor loss

IV Discussion

Because the linear data set is too simple, the accuracy is high even with few epochs training. So I use xor data set to show the difference in this section.

1. **Try different learning rates (4 neurons per layer is fixed)**

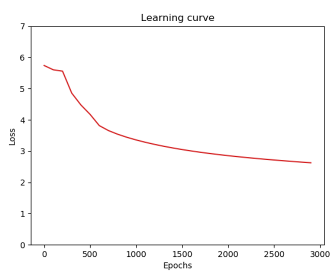

Figure 12: lr = 1

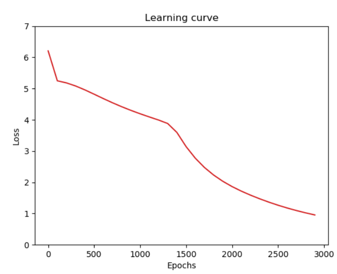

Figure 13: lr = 0.1

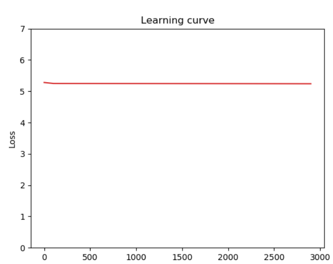

Figure 14: lr = 0.01

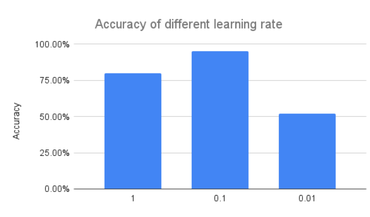

Figure 15: Accuracy comparison

Learning rate is the ratio how much to update the parameters. If the learning rate is too small(0.01), the loss decreases extremely slow, and the accuracy is low as well. However, if the learning

rate is too large(1), it is hard to find the global minimum with such large step in each iteration. In this case, 0.1 is an adequate learning rate which makes the loss decrease steadily and has nice performance.

2. Try different numbers of hidden units (learning rate of 0.1 is fixed)

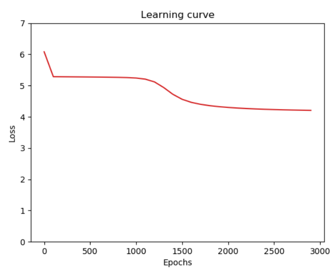

Figure 16: 2 neurons per layer

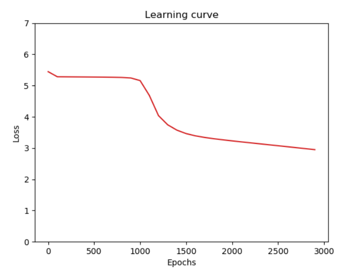

Figure 17: 3 neurons per layer

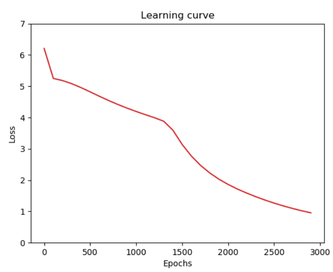

Figure 18: 4 neurons per layer

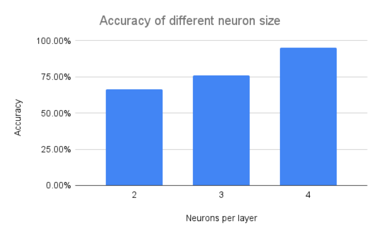

Figure 19: Accuracy comparison

Observing from the figures above, we could see with more hidden units, the loss decreases more quickly and the accuracy is higher as well. However, with more hidden units per layer, which means more computation is needed to update the weights. Meanwhile, it is time-consuming to train the network when the hidden units are large.

3. Try without activation functions

```
(base) ubuntu@ec037-048:~/DLP/HW1$ python3 test_back_prop.py
test_back_prop.py:84: RuntimeWarning: overflow encountered in matmul
  self.z = self.weight.T@input_data
test_back_prop.py:120: RuntimeWarning: invalid value encountered in add
  self.layers[layer+1].n[j].partial_c_z * self.layers[layer+1].n[j].weight[i]
Epoch  0  loss:  [nan]
Epoch  100  loss:  [nan]
```

Figure 20: Without activation function

Originally, the sigmoid function limits the output between 0 and 1. However, if no activation function is used, the output value could be extremely large. It will encounter overflow problem. Consequently, the training process is failed.

V Conclusion

Backpropagation is an efficient algorithm to train neural network. Moreover, it is also important to set the number of layers, hidden units per layer and the learning rate. With adequate parameters, it is able to train a neural network with fewer time and gets high accuracy. In this assignment, there are two layers and 8 hidden units per layer in my network. Moreover, both linear and xor data set gets 100% of accuracy.