DLP Lab1 Report

309554002 楊上萱

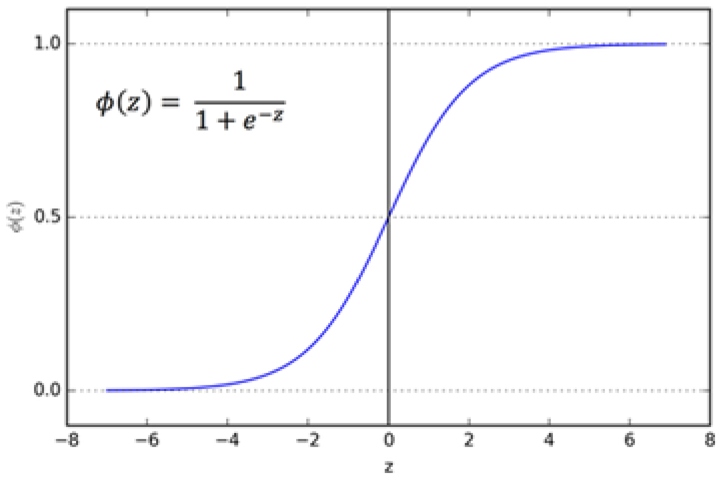
1. Introduction:

In this assignment, we are going to implement simple neural networks with forwarding pass and backpropagation using two hidden layers.

2. Experiment setups:

a. Sigmoid functions

Sigmoid function is chosen as the activation functions of the neural network. It's a mathematical function which has a characteristic "S"-shaped curve.



We use a common sigmoid function - **logistic function**, which is defined by:

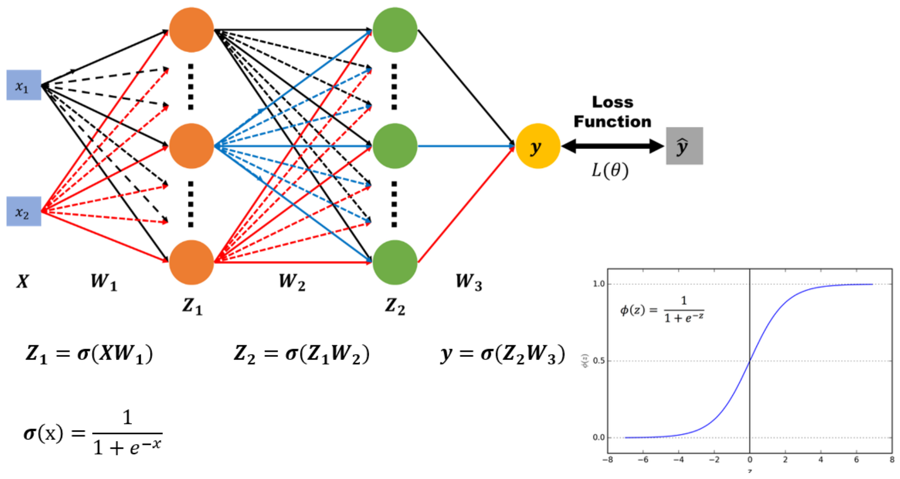
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

And its derivative is:

$$\sigma(x)(1-\sigma(x))$$

b. Neural network

Neural networks (NNs) are composed of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or neuron, connects to another and has an associated weight. The input is fed forward to those hidden layers and their corresponding activation functions, and get the output in the final output layer.



In our case, ***x*** is fed forward to those two layers, and get the output ***y***, and then compare to the ground true to calculate loss.

c. Backpropagation

In every training epoch, each layer calculates the value of every node and gets the outputs during **forward pass**.
To minimize the loss, we need to calculate the gradient with respect to the parameters. Here, **backpropagation** is used to make the calculation of gradient becomes easier.

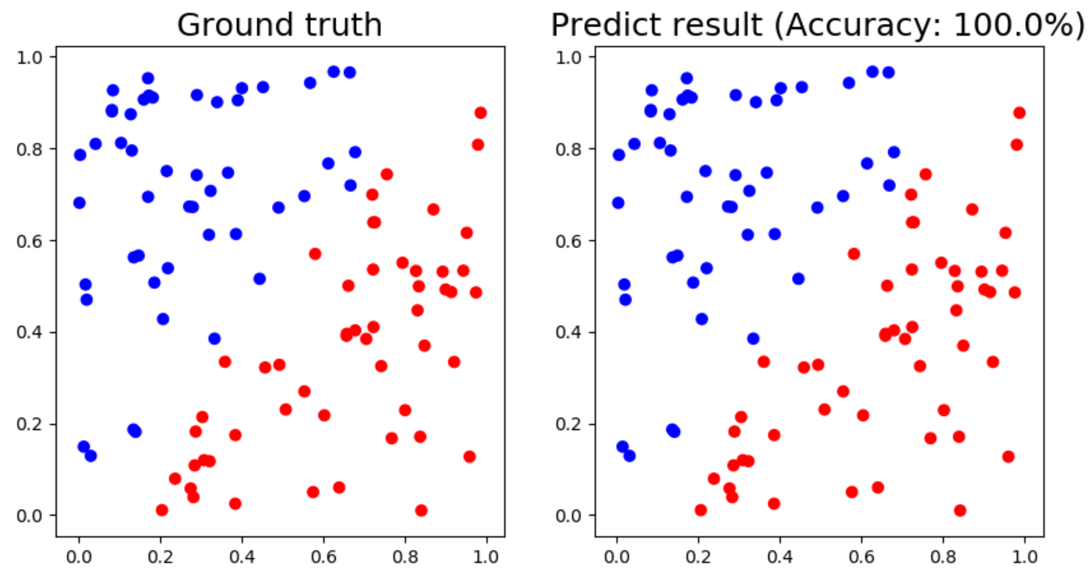
- We propagate the output activations back through the network using the training pattern target in order to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons.
- And then multiply its output delta and input activation to get the gradient of the weight.
- Finally use the gradient to update the weights:

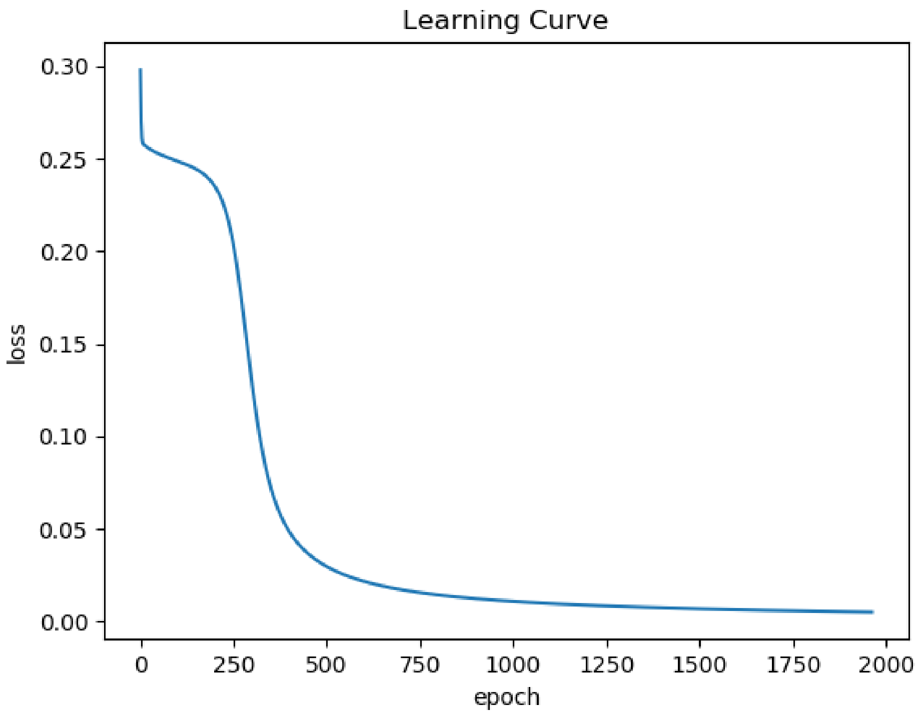
$$W = W - \alpha \cdot \frac{\partial L}{\partial W}$$

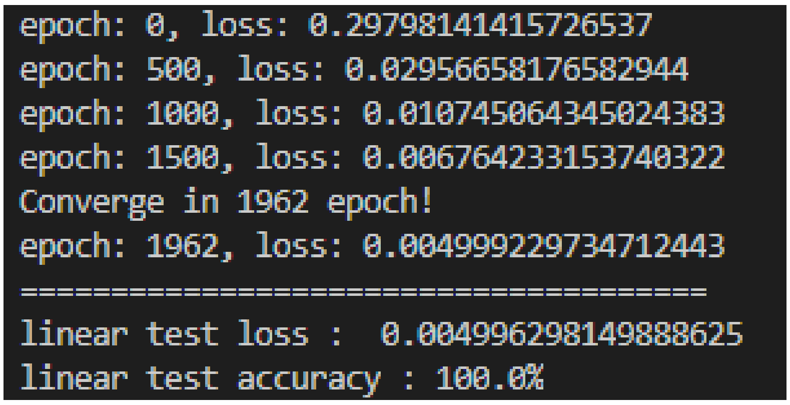
3. Results of testing:

a. Linear

- lr=1
- 2 hidden layers, each has 4 hidden nodes



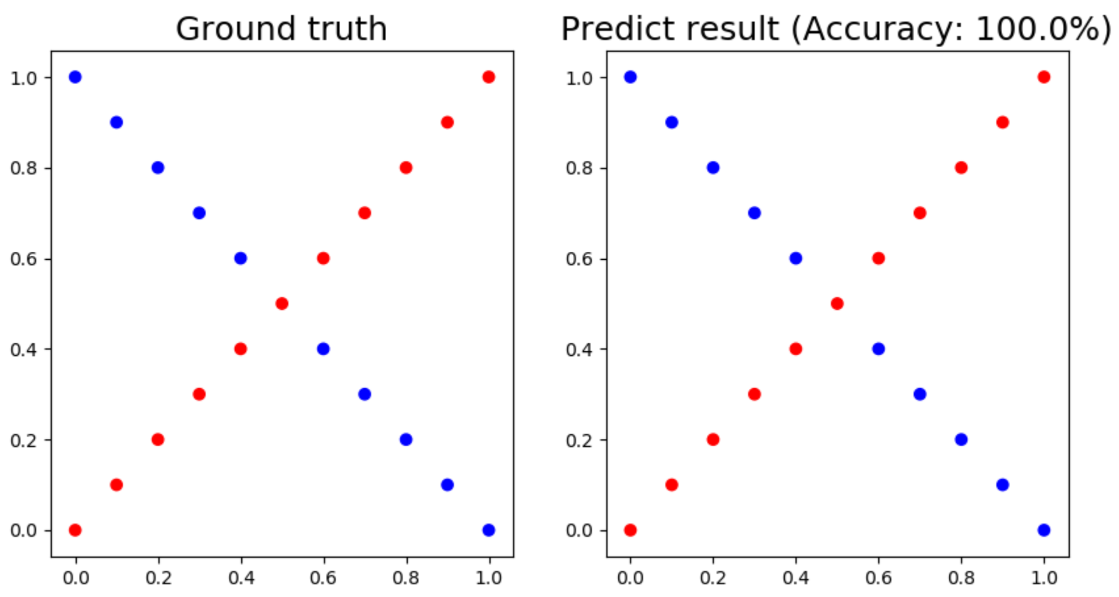


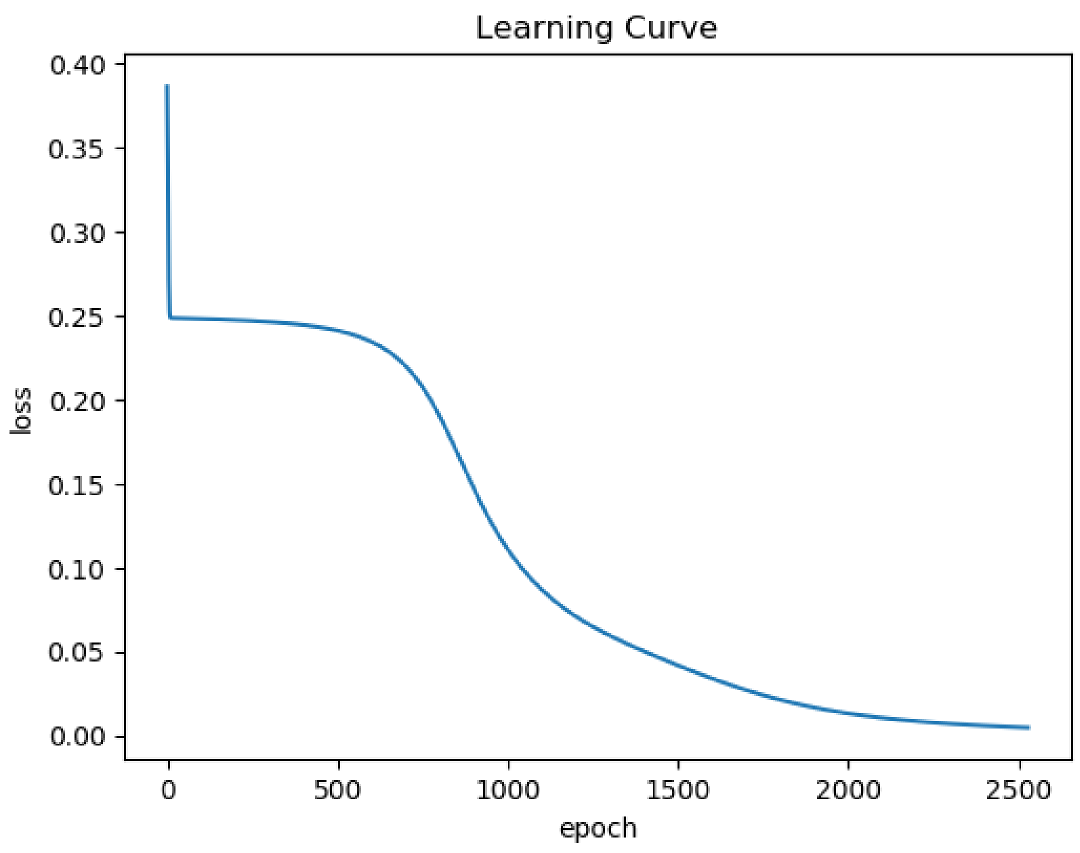


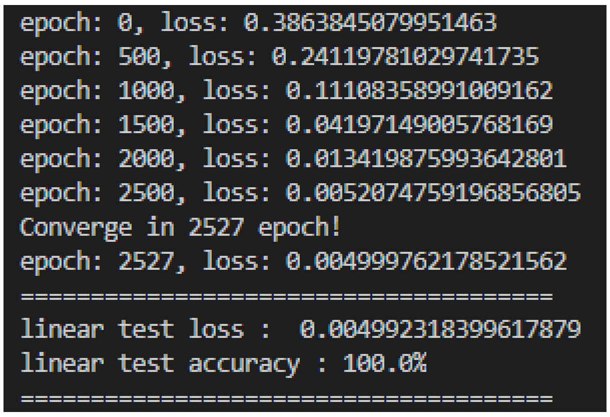
```
epoch: 0, loss: 0.29798141415726537
epoch: 500, loss: 0.02956658176582944
epoch: 1000, loss: 0.010745064345024383
epoch: 1500, loss: 0.006764233153740322
Converge in 1962 epoch!
epoch: 1962, loss: 0.004999229734712443
=======================================
linear test loss :  0.004996298149888625
linear test accuracy : 100.0%
```

b. XOR

- lr=1
- 2 hidden layers, each has 4 hidden nodes





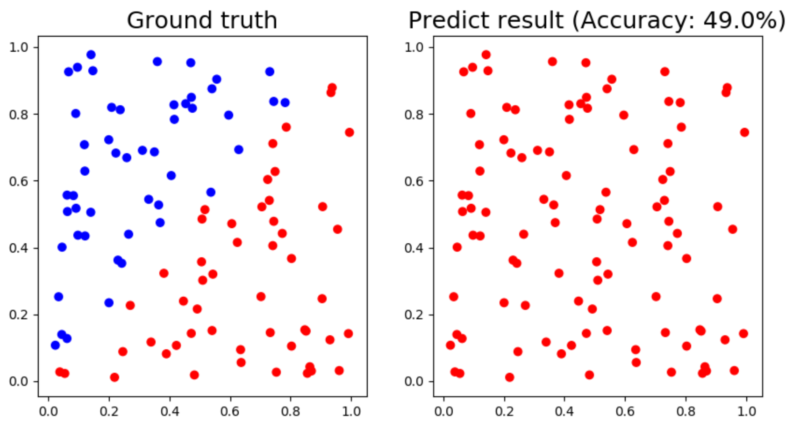


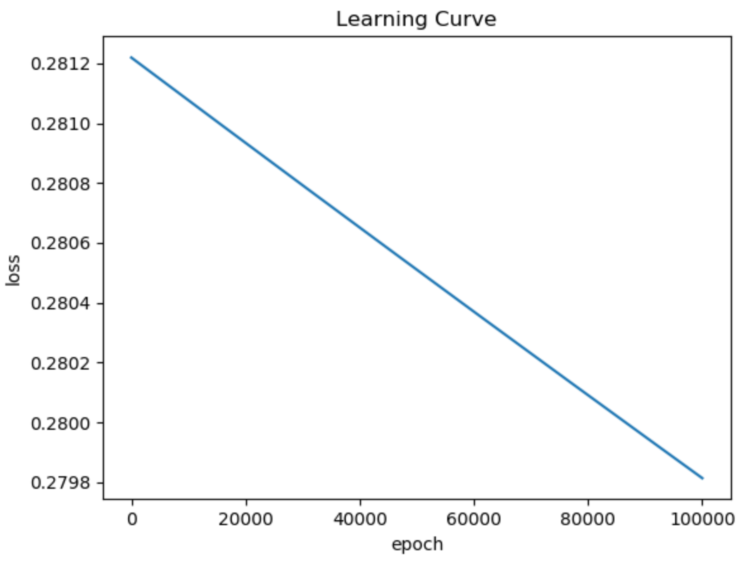
```
epoch: 0, loss: 0.3863845079951463
epoch: 500, loss: 0.24119781029741735
epoch: 1000, loss: 0.11108358991009162
epoch: 1500, loss: 0.04197149005768169
epoch: 2000, loss: 0.013419875993642801
epoch: 2500, loss: 0.005207475919685680 5
Converge in 2527 epoch!
epoch: 2527, loss: 0.004999762178521562
======================================
linear test loss :  0.004992318399617879
linear test accuracy : 100.0%
======================================
```

4. Discussion:

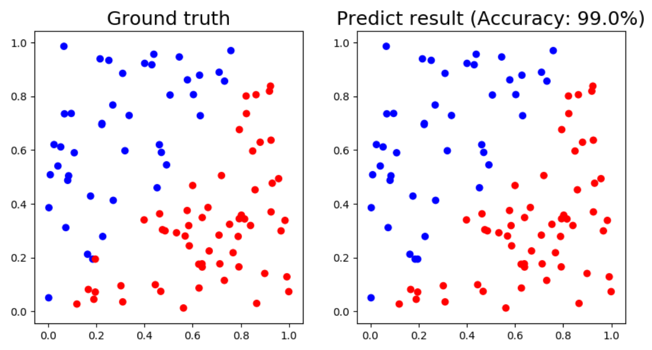
a. Try different learning rates

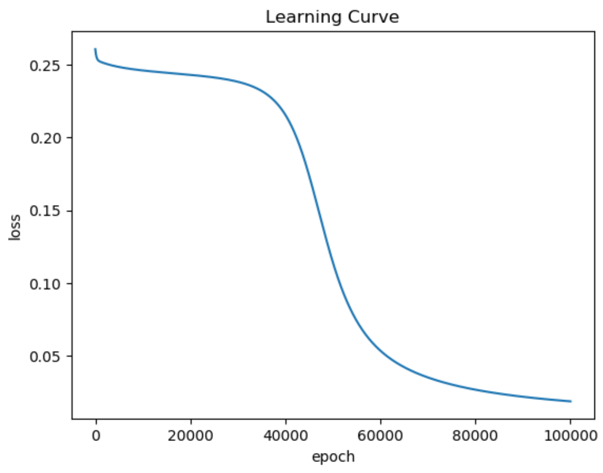
i. lr=1e-6



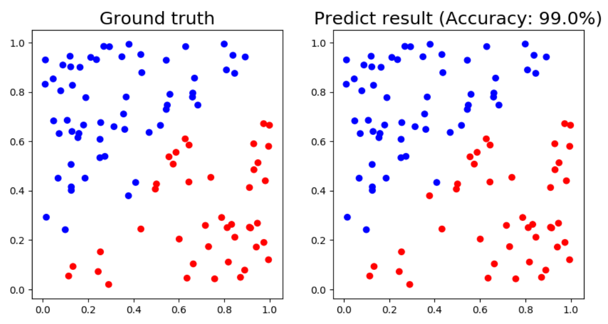


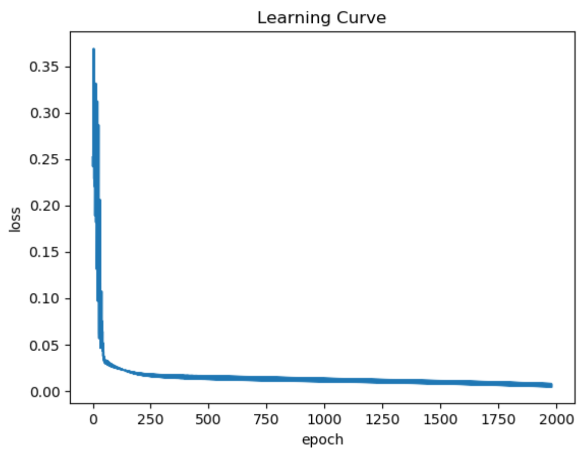
ii. lr=1e-2



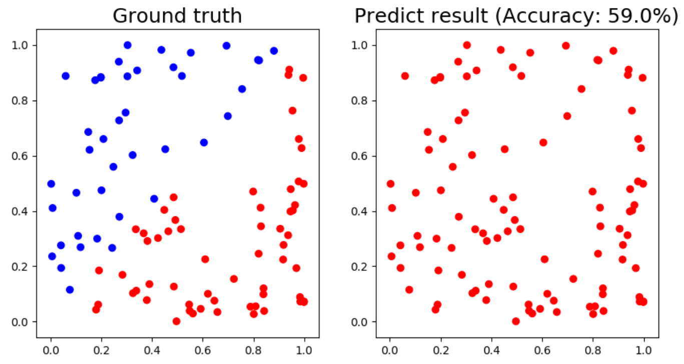


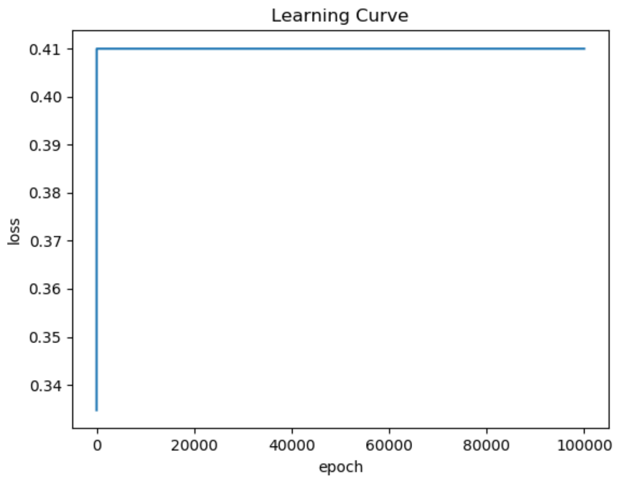
iii. lr=10



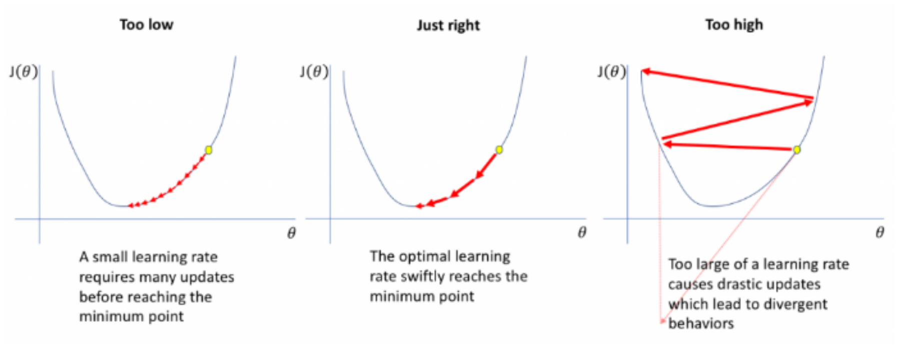


iv. lr=1e6





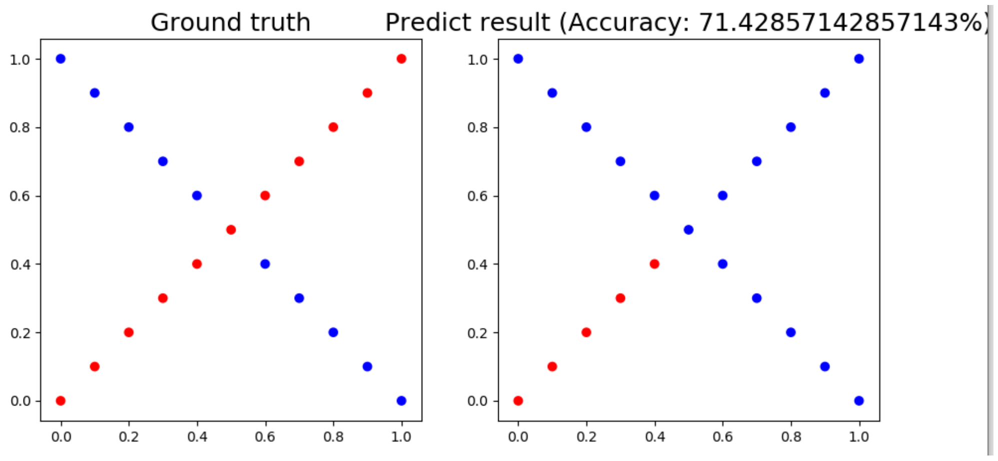
If we choose a proper learning rate, the larger it is, the more quickly it converges. However, if the learning rate is too large (ex: 1e6), it will fail to converge. Vice verse, if the learning rate is too small (ex: 1e-6), the learning progress will be too slow.

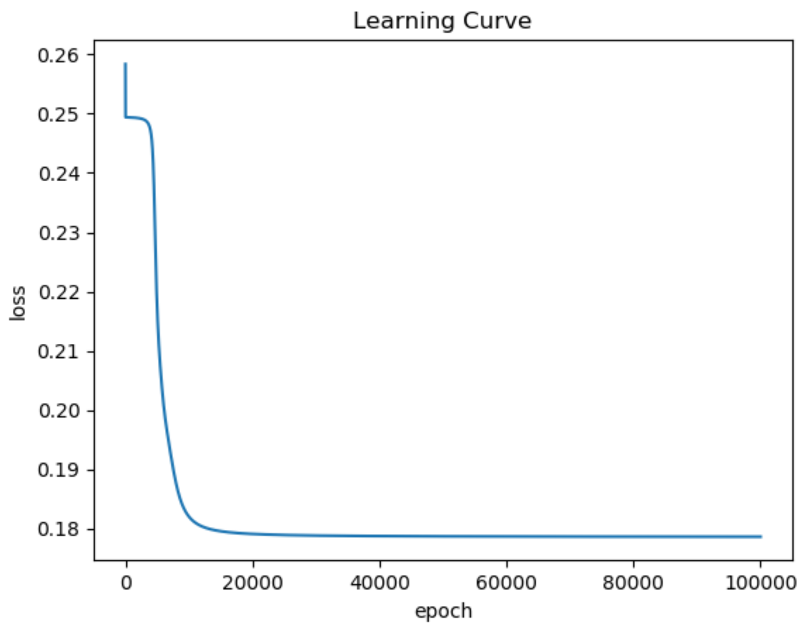

Too low
Just right
Too high
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

b. Try different numbers of hidden units

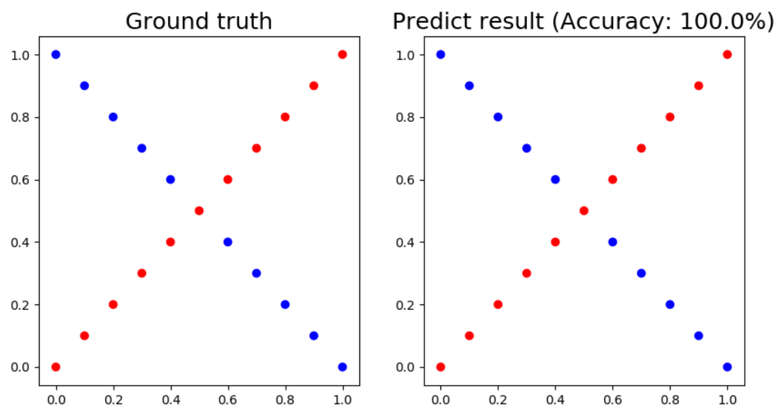
In this part, we chose the XOR task because it is harder to train, and different amount of hidden units will show more difference.

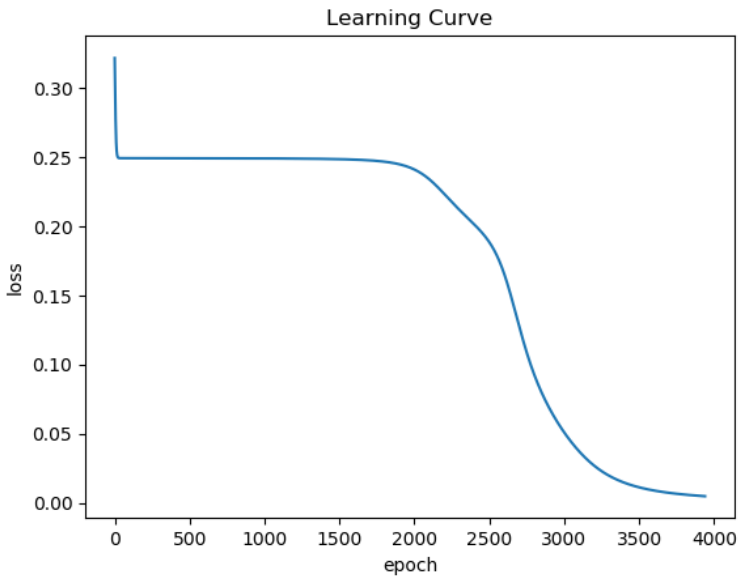
i. hidden units num of each layer: (1, 1)



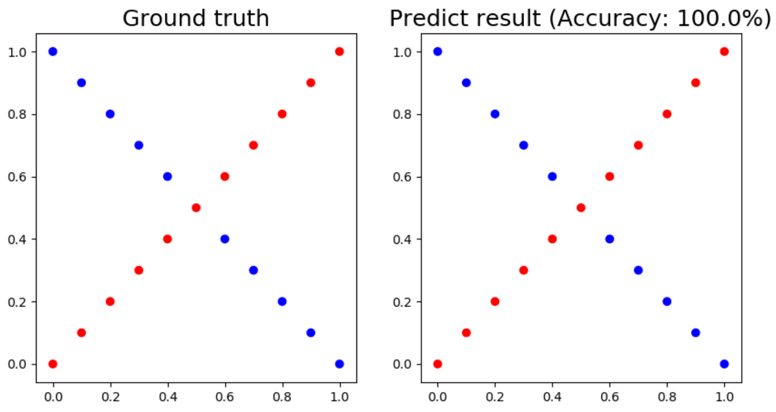


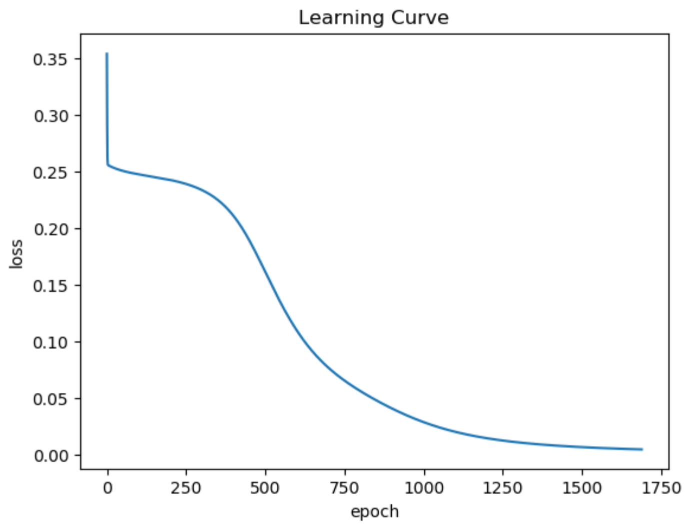
ii. hidden units num of each layer: (2, 2)



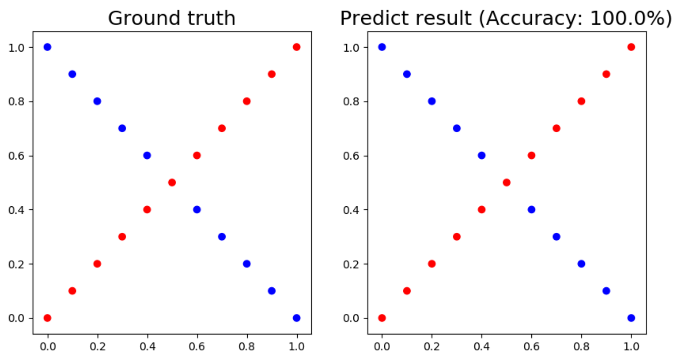


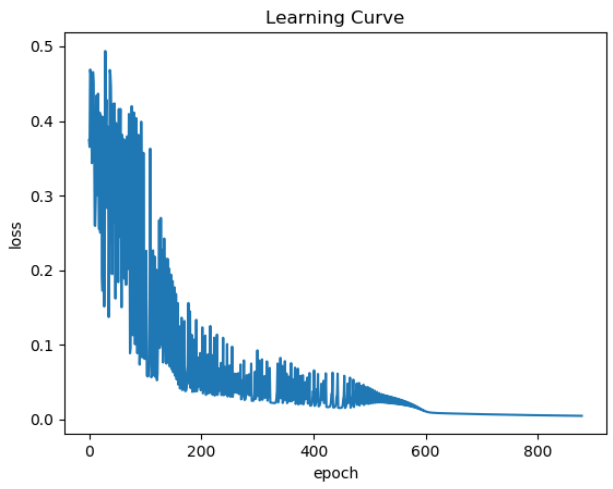
iii. hidden units num of each layer: (8, 8)



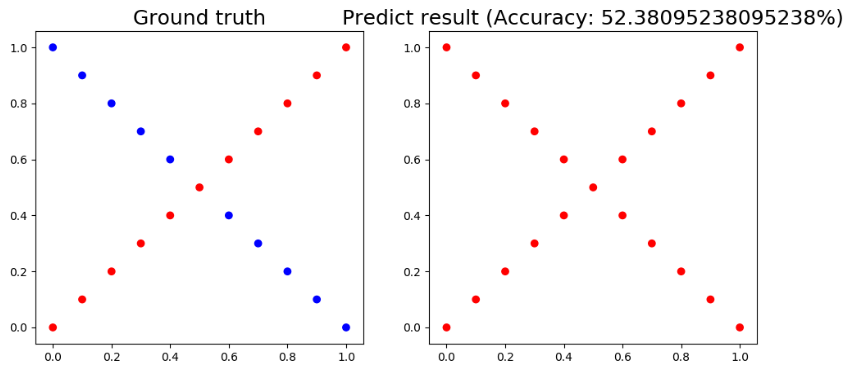


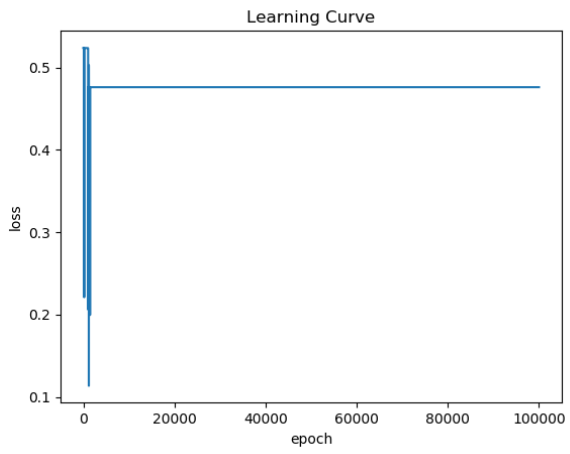
iv. hidden units num of each layer: (100, 100)





v. hidden units num of each layer: (400, 400)



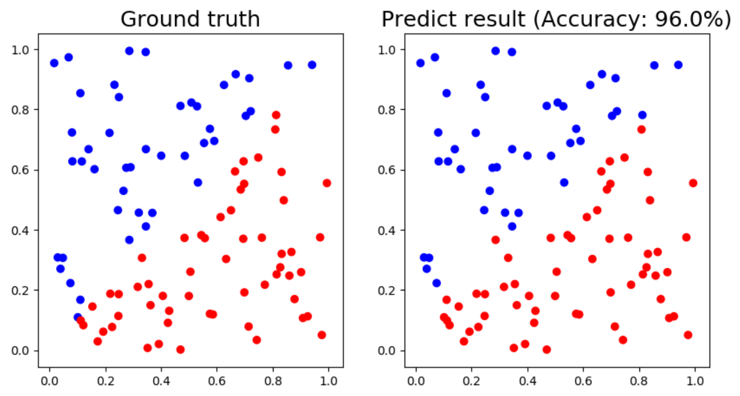


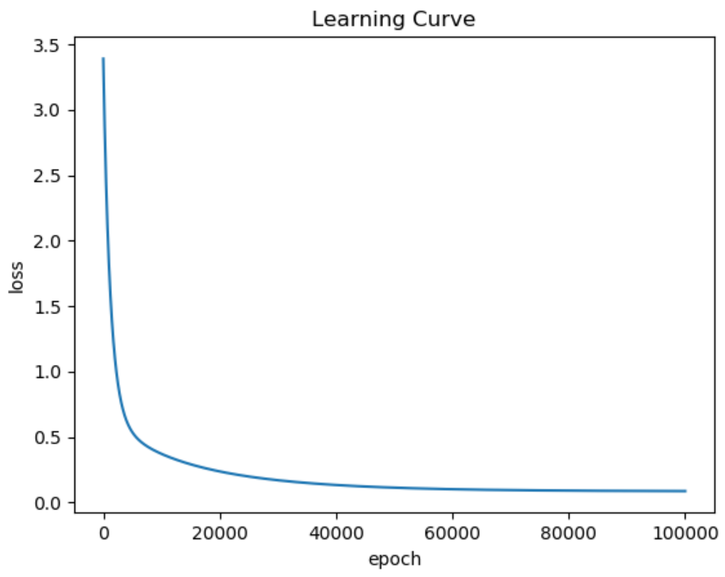
If the amount of hidden units is not enough, the XOR problem cannot be solved. It can be told in the case where the number of hidden units is (1, 1): the loss doesn't decrease at all in the late process.

However, if the number of units is too large (ex: 400, 400), the XOR task cannot be solved. Also, the computation time will significantly increase due to the more complicated network architecture.

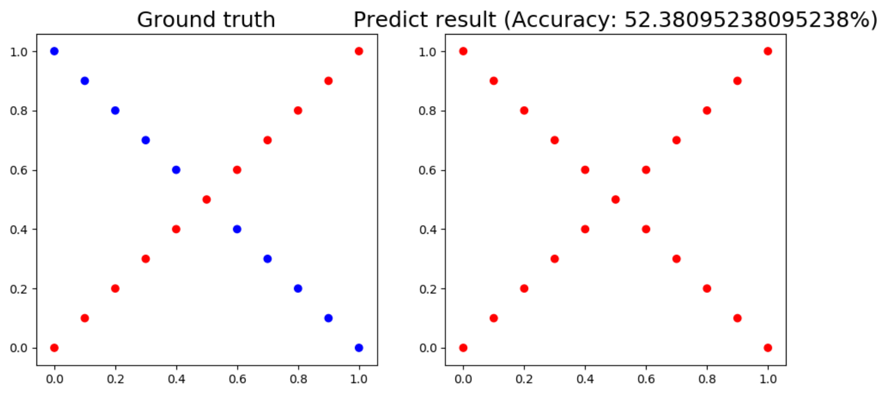
c. Try without activation functions

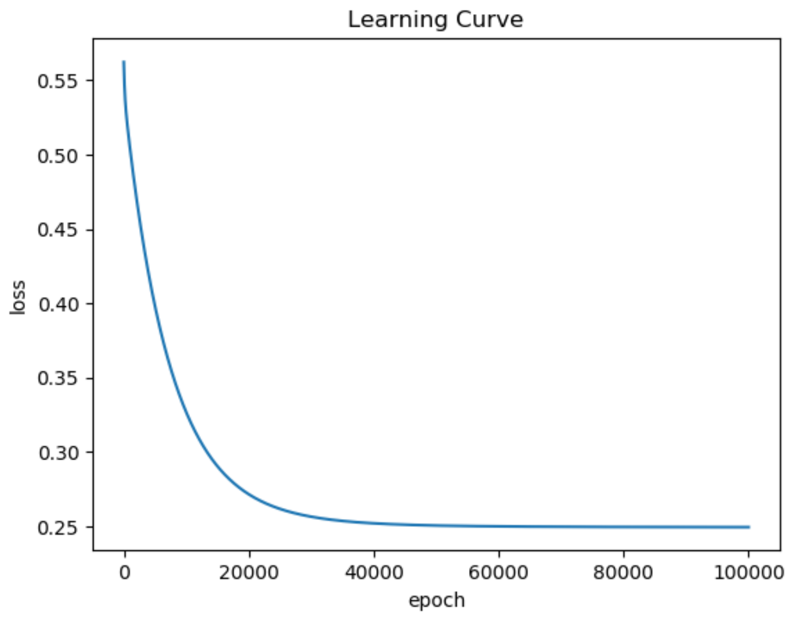
i. Linear (lr=1e-5, hidden layers with 4, 4 units for each layer)





ii. XOR





Without activation function, the gradient will possibly become infinite (or minus infinite). However, in the linear case, it can somehow be solved by decreasing the learning rate.
And as we can see, the XOR task cannot be solved without activation function.