DLP Lab2 - Backpropagation Report

- Student name: 林浩君
- Student ID: 0816124

1. Introduction

- This lab require we make a hand-craft neuron network from scratch, and implement backpropagation algorithm when doing gradient descent.
- Since tasks in this lab are classification problems, i choose cross entropy as loss function.

$$L(\theta) = \sum \hat{y} \; log \, y_i \; + \; (1-\hat{y}) \; log(1-y_i)$$

2. Experiment setups

A. Sigmoid functions

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

B. Neural network

- 4 layers in total, including input and output layer and 2 hidden layers.
- input size: 2
- output size: 1

C. Backpropagation

- For the specific weight between two layers, denote
  1. w as the target weight
  2. z and z' as the value **before sigmoid function** in two sides of w
     - from z to z'
- the gradient of w can be written as below using chain rule

$$\frac{\partial L}{\partial w} = \frac{\partial z'}{\partial w}\frac{\partial L}{\partial z'}$$

- Then we called $\frac{\partial z'}{\partial w}$ as **forward pass**, $\frac{\partial L}{\partial z'}$ as **backward pass**

I. FORWARD PASS

- Let value of z' comes from $z_1, z_2, z_3 \ldots$ with weghts $w_1, w_2, w_3 \ldots$ in previous layer
- z' can be written as

$$\begin{aligned} z' &= \sigma(z_1)w_1 + \sigma(z_1)w_2 \ + \ \ldots \ + \sigma(z_n)w_n \\ &= \sum_{i=0}^{n} \sigma(z_i)w_j \end{aligned}$$

- Then calculate partial derivative using result above

$$\frac{\partial z'}{\partial w_i} = \sigma(z_i)$$

- Therefore, value of forward pass is equal to input value,we can just run throught the whole network to get values in forward pass, when implementation, i store z_i instead of $\sigma(z_i)$.
- Matrix form

$$F_0 = x, \ \ \text{where x is input}$$

$$F_{i+1} = \sigma(F_i)W$$

II. BACKWARD PASS

- case 1: z is in output layer
  - z after sigmoid function should be our model ouput y

$$\sigma(z) = y$$

  - Then conbime z and loss function L

$$L(\theta) = \sum \ \hat{y} \ log \, \sigma(z) \ + \ (1 - \hat{y}) \ log(1 - \sigma(z))$$

- Then calculate partial derivative using result above

$$\frac{\partial L}{\partial z} = \sigma'(z)\left[\frac{\hat{y}}{\sigma(z)} + \frac{(1-\hat{y})}{(1-\sigma(z))}\right]$$

- case 2: z is in input layer or hidden layer
  - Let value of z connect to $z'_1, z'_2, z'_3, \ \ldots$ with weight $w_1, w_2, w_3, \ \ldots$ in next layer
  - z contribute its value to $z'_1, z'_2, z'_3 \ldots$, so partial derivative can be written as

$$\begin{aligned}\frac{\partial L}{\partial z} &= \sigma'(z)\left[w_1\frac{\partial L}{\partial z'_1} + w_2\frac{\partial L}{\partial z'_2} + \ \ldots\right] \\ &= \sigma'(z)\sum_{i=1}^{n} w_i\frac{\partial L}{\partial z'_i}\end{aligned}$$

- Matrix form

$$B_4 = \sigma'(z)\left[\frac{\hat{y}}{\sigma(z)} + \frac{(1-\hat{y})}{(1-\sigma(z))}\right]$$

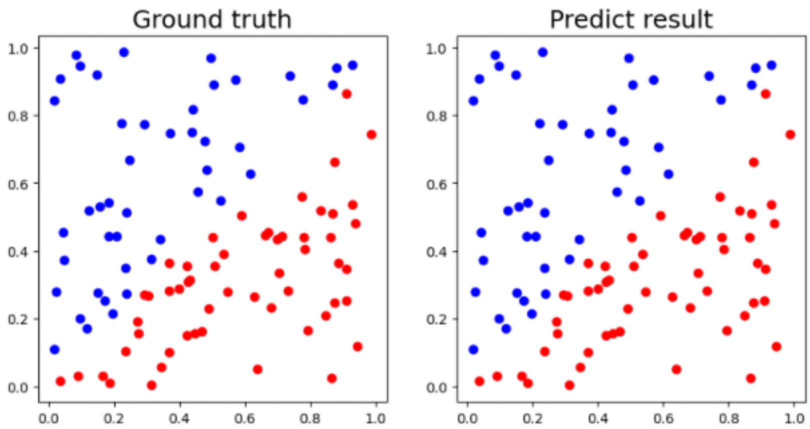
$$B_{i-1} = \sigma'(F_{i-1}) \cdot B_i W^T$$

  - where dot sign $\cdot$ means element wise multiplication

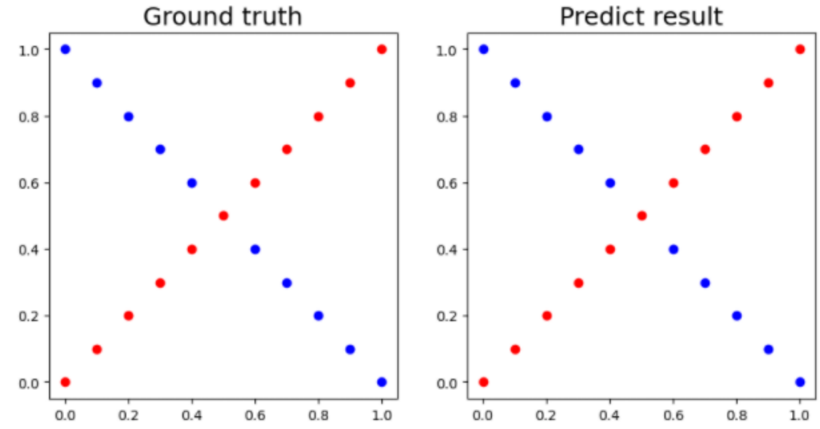
3. Results of your testing

A. Screenshot and comparison figure

- Linear case



- XOR case



B. Show the accuracy of your prediction

- Linear case

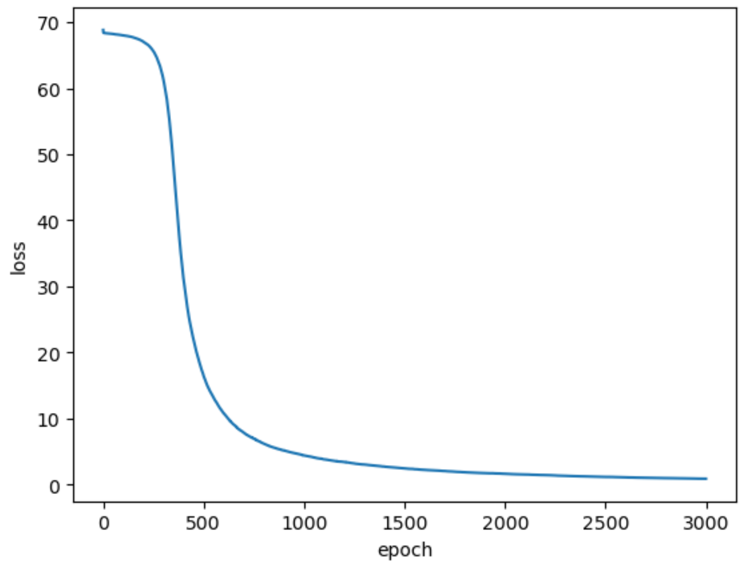
```
accuracy: 99.0%
```

- XOR case

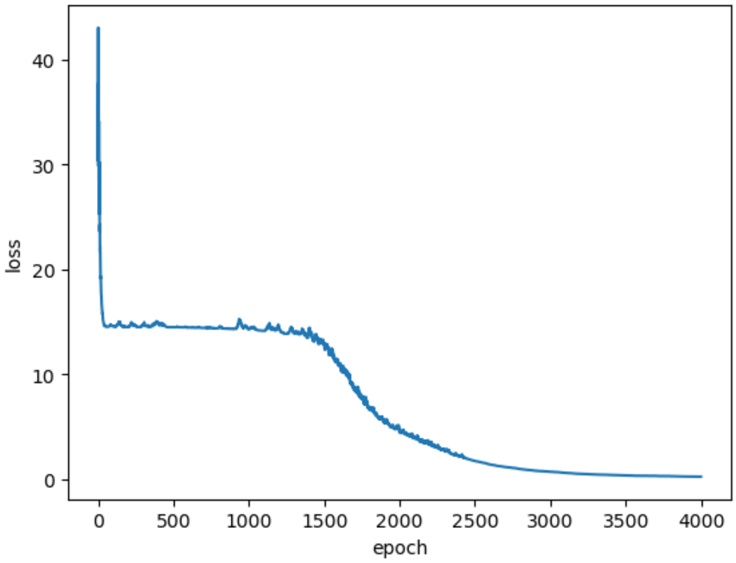
```
accuracy: 100.0%
```

C. Learning curve (loss, epoch curve)

- Linear case



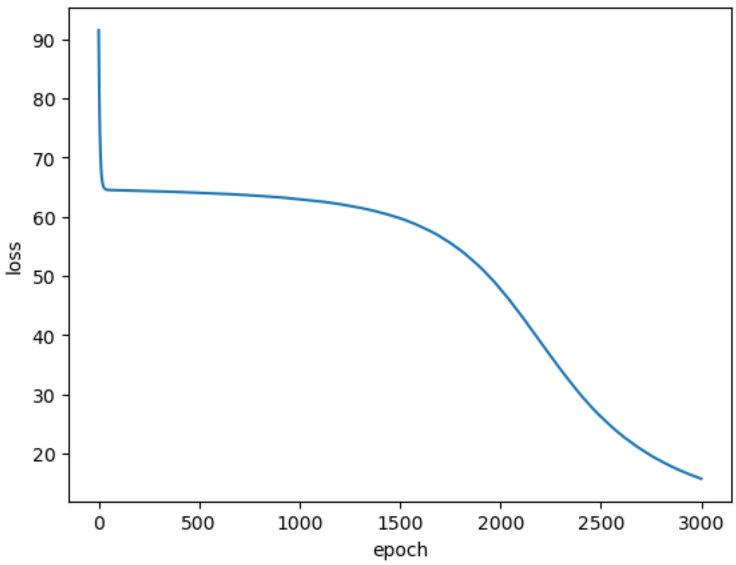
- XOR case



4. Discussion

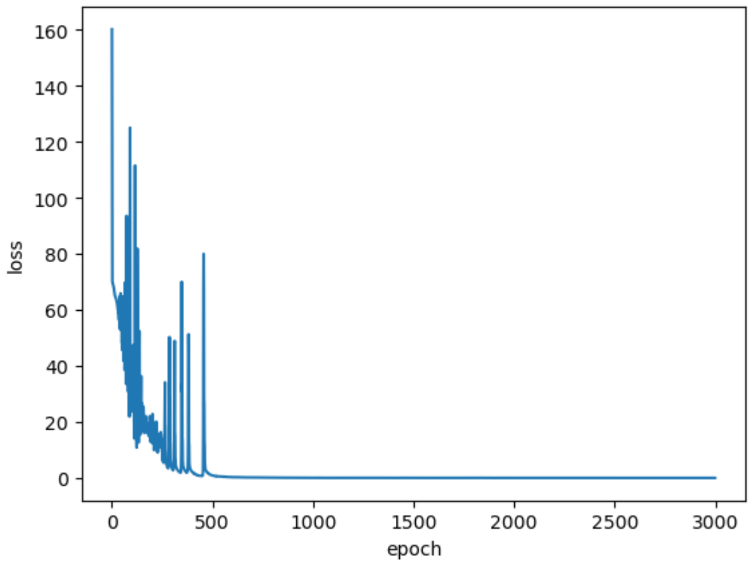
A. Try different learning rates

- I choose lr=1e-2 in linear case and lr=2e-2 in xor case.
- If choose smaller learning rate, loss will converge too slow.
  - Set lr=1e-3 in linear case, loss is about 20 after 3000 epoch while loss is getting 0 when we choose lr=1e-2 after same amount of epoch.



- If choose larger learning rate, loss will change rapidly and will be hard to converge, or even diverge to infinite.

- Set lr=1e-1 in linear case

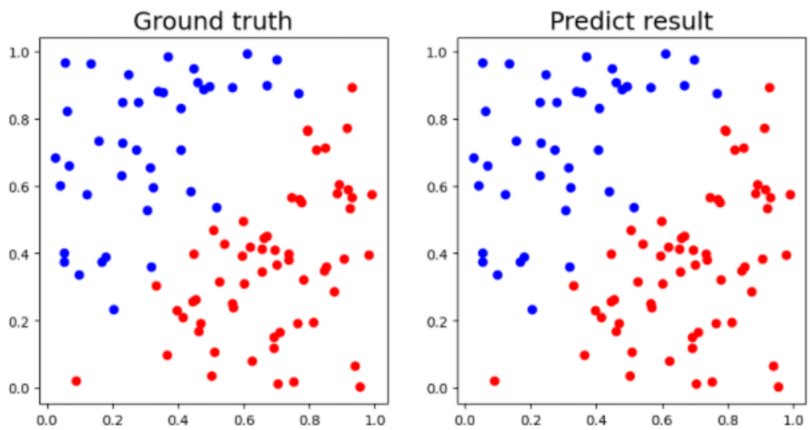


B. Try different numbers of hidden units

- I choose number of neurons as 10 in linear case and 100 in xor case.
- Linear dataset is easier to fit, a few number of neurons can lead to good performance.
- XOR dataset is harder to fit, need more number of neurons to get good performance, if there's too few neurons, loss will not converge or need more epochs.

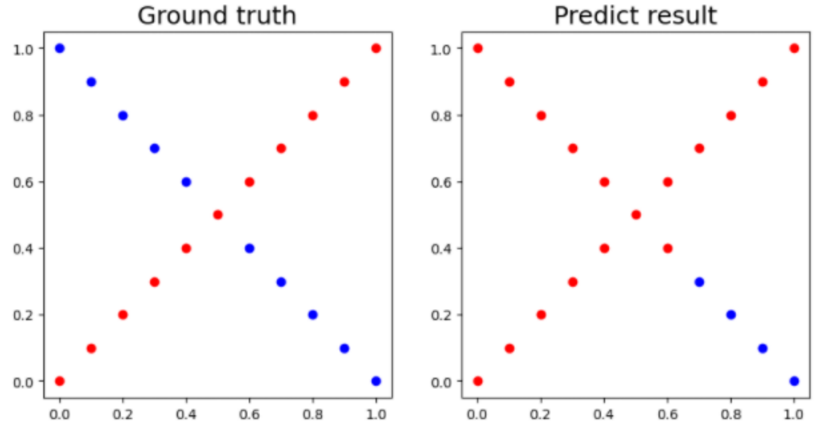
C. Try without activation functions

- Set a global boolean variable **wo_activation**
- In linear case, network still have good performance and get accuracy 100%




```
accuracy: 100.0%
```

- In xor case, netwrok can't fit well since linear model can't fit xor problem.




```
accuracy: 71.42857142857143%
```