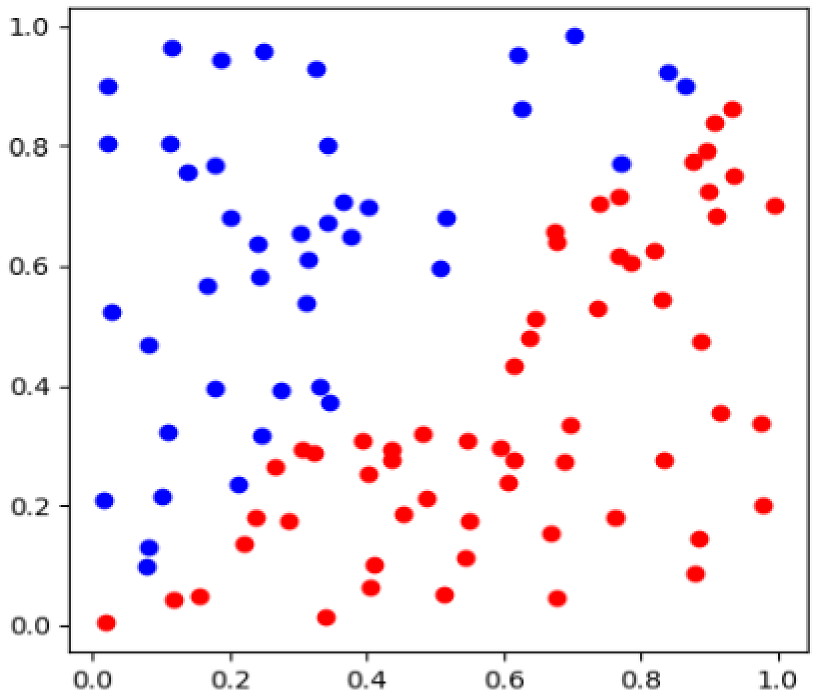
深度學習 HW1

機器人學程 李啟安 310605015

1. Introduction

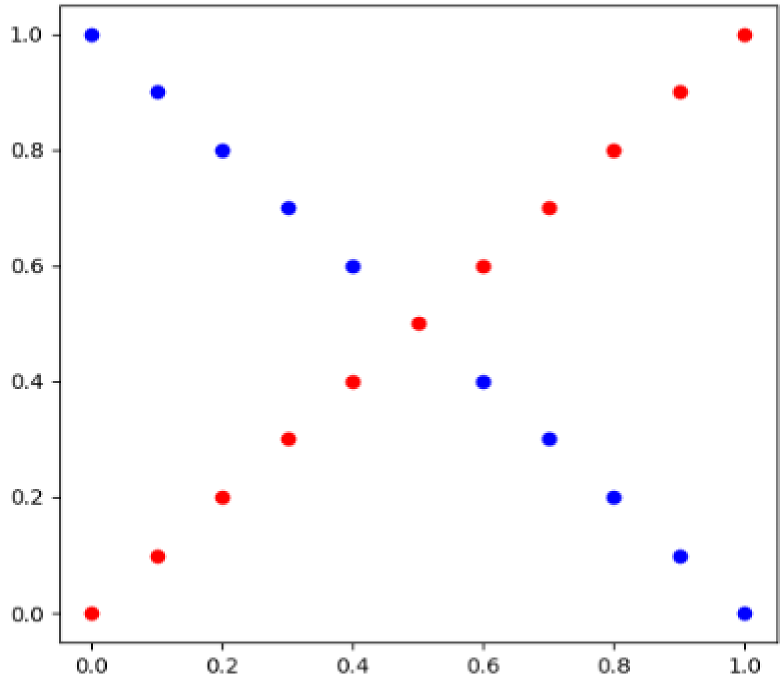
實做一個 2 layer 神經網路，沒有使用深度學習框架支援。雖然本次作業只要實做兩層，但是本次程式可以塞入任意形狀的網路，但是每一層的激勵函數都會是一樣的。

- Dataset
    - linear data (資料分佈簡單，比較簡單即可訓練起來)
        - 從助教範例程式來



    - XOR data （因為只有20筆資料，且型態較為複雜）

- 從助教範例程式來



- NN structure
  - linear data 比較好的網路形狀為 [2 8 6 1]
  - XOR data 比較好的網路形狀 [2 6 8 1]
- activation function
  - sigmoid
  - tanh
- optimizer
  - SGD
  - Momentum
- Cost function: cross entropy loss function
  - 因為本次作業唯一分類問題，非1則0
  - 實際推導蠻有趣的，但是因為有log，所以需要做一些變化
    - ex : tanh 需要投影到 0 - 1 之間

2. Experiment Setups

A. Sigmoid Functions

Sigmoid function is a nonlinear function, which can deal with the nonlinear problem like XOR!

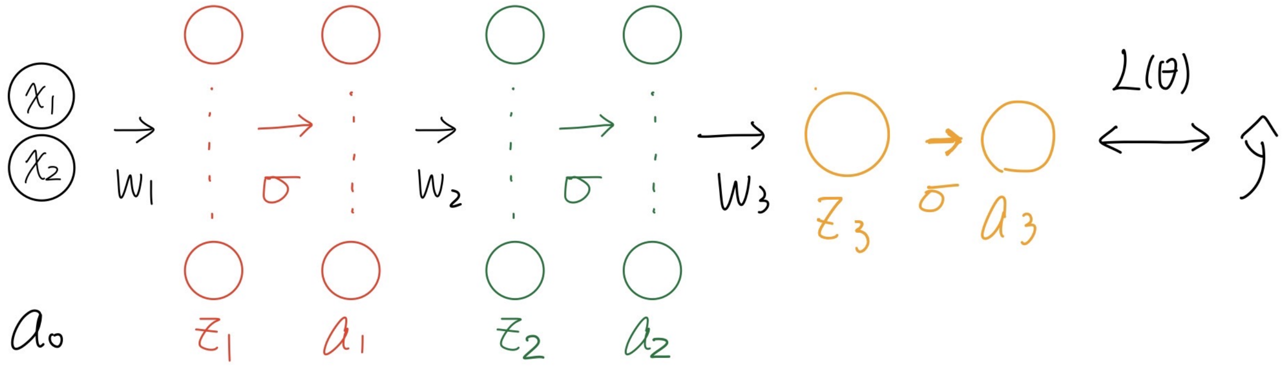
- sigmoid function

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

- derivative

$$\begin{aligned}\frac{d}{dx}\sigma(x) &= \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \frac{(1+e^{-x})-1}{(1+e^{-x})^2} \\ &= \sigma(x)(1-\sigma(x))\end{aligned}$$

B. Neural Network



- Architecture:
  - x_1, x_2 are input data (a_0)
  - $z_1 = W_1 a_0$
  - $a_1 = \sigma(z_1)$
  - $z_2 = W_2 a_1$
  - $a_2 = \sigma(z_2)$
  - $z_3 = W_3 a_0$
  - $a_3 = \sigma(z_3)$
- Loss Function
  - Cross Entropy Loss

C. Backward propagation

- cross entropy loss function

$$C = -y \log \hat{y} + (1-y)\log(1-\hat{y})$$

- Weight derivative can be see as
    - W_3

$$\frac{\partial C}{\partial W3} = \frac{\partial C}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial \omega_3}$$

    - W_2

$$\frac{\partial C}{\partial W2} = \frac{\partial C}{\partial a_3} . \frac{\partial a_3}{\partial z_3} . \frac{\partial z_3}{\partial a_2} . \frac{\partial a_2}{\partial z_2} . \frac{\partial z_2}{\partial W2}$$

    - W_1

$$\frac{\partial C}{\partial W_1} = \frac{\partial C}{\partial a_1} . \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial W1}$$

- if l is the last layer

$$\frac{\partial C}{\partial z_l} = \frac{\partial C}{\partial a_l} . \frac{\partial a_l}{\partial z_l} = \frac{\partial C}{\partial a_l} . \frac{d}{dz} \sigma\left(z_l\right)$$

$$\frac{\partial C}{\partial a_l} = \frac{\partial\left(-(y \log a_l + (1-y)) \log(1-a_l))\right)}{\partial a_l}$$

$$\frac{\partial C}{\partial a_l} = -\left(\frac{y}{a_l} - \frac{1-y}{1-a_l}\right)$$

$$\frac{\partial C}{\partial z_l} = \frac{\partial C}{\partial a_l} . \frac{\partial a_l}{\partial z_l}$$

- if l is not the last layer

$$\begin{aligned} \frac{\partial C}{\partial z_l} &= \frac{\partial C}{\partial z_{l+1}} \cdot \frac{\partial z_{l+1}}{\partial a_l}, \frac{\partial a_l}{\partial z_l} \\ &= W_{l+1}^T \cdot \frac{\partial C}{\partial z_{l+1}} . * \frac{d}{dz} \sigma\left(z_l\right) \end{aligned}$$

    - **notation $.*$ means element-wise dot product**
    - where $\frac{\partial C}{\partial z_{l+1}}$, you can get from the last propagation you do

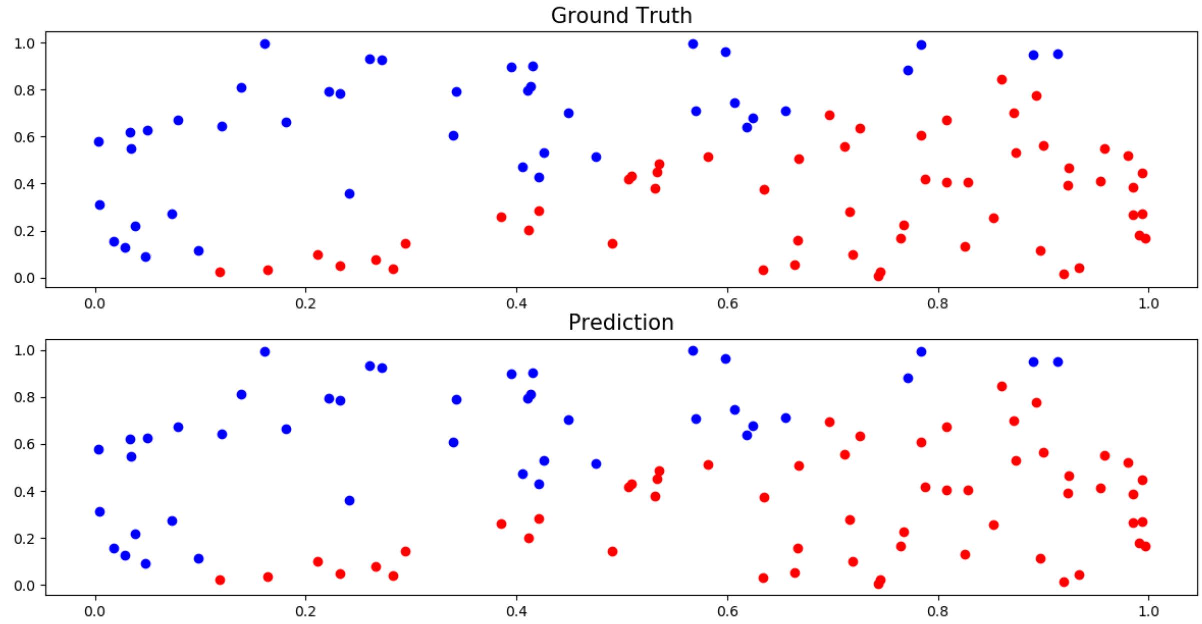
- you can easily get

$$z_l = W_l . a_{l-1}$$
$$\frac{\partial z_l}{\partial W_l} = a_{l-1}$$

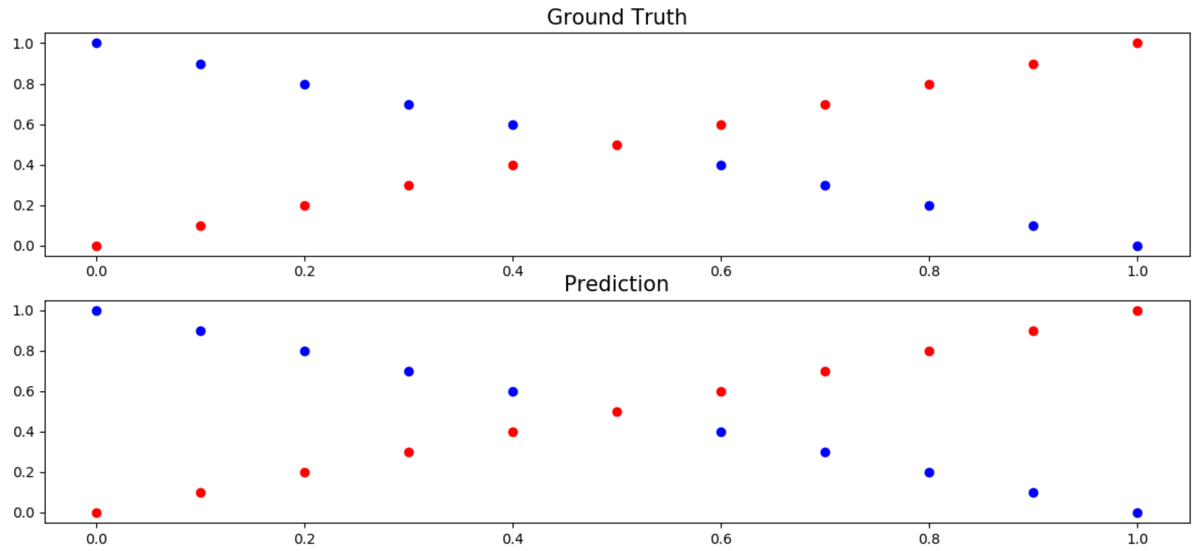
3. Result of your testing

A. Screenshot and comparison figure

- Linear data



- Nonlinear data



B.Show the accuracy of your prediction

- Linear data

```
loss of 1    epoch : 0.7188564088281241
loss of 501    epoch : 0.009771446495385253
loss of 1001    epoch : 0.005012320026073503
loss of 1501    epoch : 0.004153901415100101
loss of 2001    epoch : 0.0035899536562391554
loss of 2501    epoch : 0.003175815433816937
loss of 3001    epoch : 0.0028609860768514525
loss of 3501    epoch : 0.002613765706408853
loss of 4001    epoch : 0.002414194205319101
loss of 4501    epoch : 0.0022493598491474935
loss of 5001    epoch : 0.002110618233001952
loss of 5501    epoch : 0.0019919859880312466
loss of 6001    epoch : 0.0018891957279339928
loss of 6501    epoch : 0.0017991227480456556
loss of 7001    epoch : 0.0017194258049657199
loss of 7501    epoch : 0.001648315007328097
loss of 8001    epoch : 0.0015843975572557491
loss of 8501    epoch : 0.0015265725920669956
loss of 9001    epoch : 0.0014739578412913277
loss of 9501    epoch : 0.0014258374123529596
accuracy: 100.0 %
```

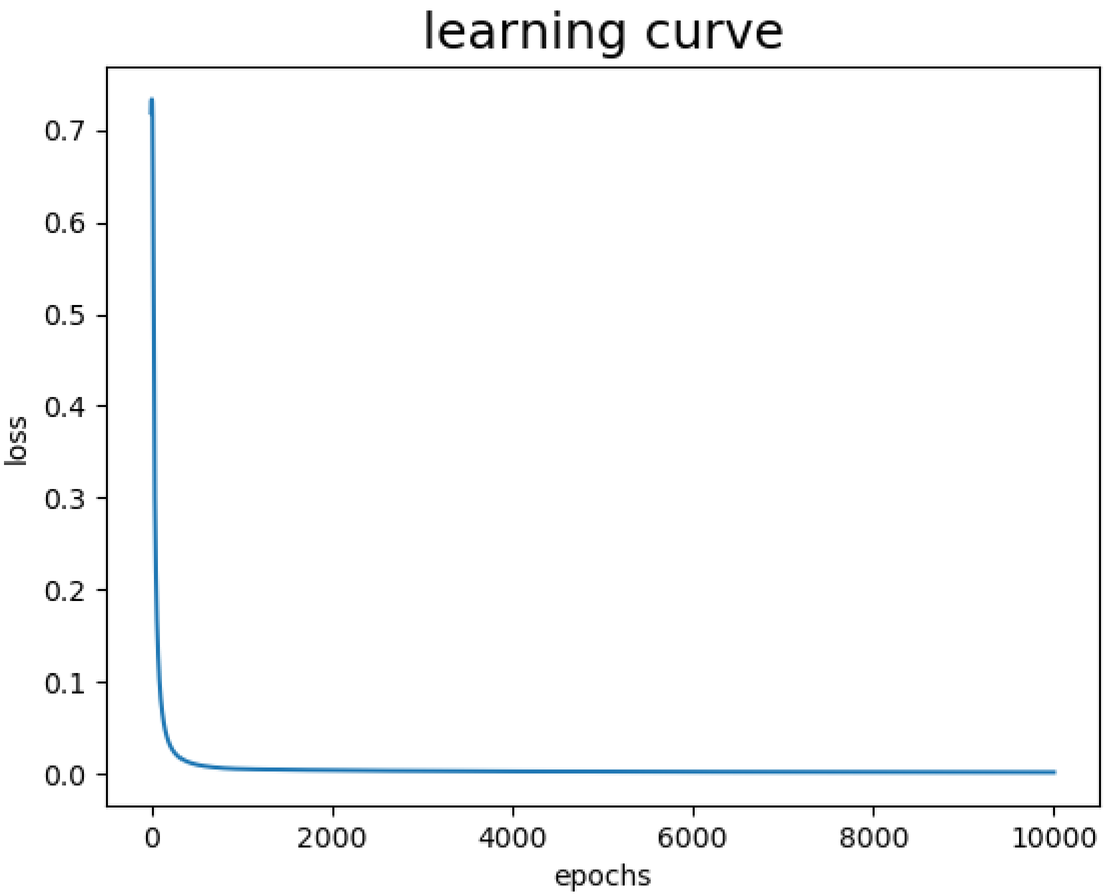
- Nonlinear data

```
loss of 1    epoch : 0.7787801849812069
loss of 501    epoch : 0.8791360619425911
loss of 1001    epoch : 0.23260147514561996
loss of 1501    epoch : 0.02006822009343498
loss of 2001    epoch : 0.008658323406679953
loss of 2501    epoch : 0.005861717449145865
loss of 3001    epoch : 0.004628957217899832
loss of 3501    epoch : 0.003930210020195398
loss of 4001    epoch : 0.0034758834964702993
loss of 4501    epoch : 0.0031540207178266167
loss of 5001    epoch : 0.002912290043872616
loss of 5501    epoch : 0.002722925060600207
loss of 6001    epoch : 0.0025697929506479076
loss of 6501    epoch : 0.002442857613781835
loss of 7001    epoch : 0.0023355341559826888
loss of 7501    epoch : 0.0022433136796798955
loss of 8001    epoch : 0.002162998914722047
loss of 8501    epoch : 0.002092255567945853
loss of 9001    epoch : 0.0020293367919042306
loss of 9501    epoch : 0.001972907382151349
loss of 10001    epoch : 0.0019219278795280937
loss of 10501    epoch : 0.0018755759728396336
loss of 11001    epoch : 0.0018331918632756802
loss of 11501    epoch : 0.0017942394488848043
loss of 12001    epoch : 0.0017582782095501494
loss of 12501    epoch : 0.00172494248727541
```

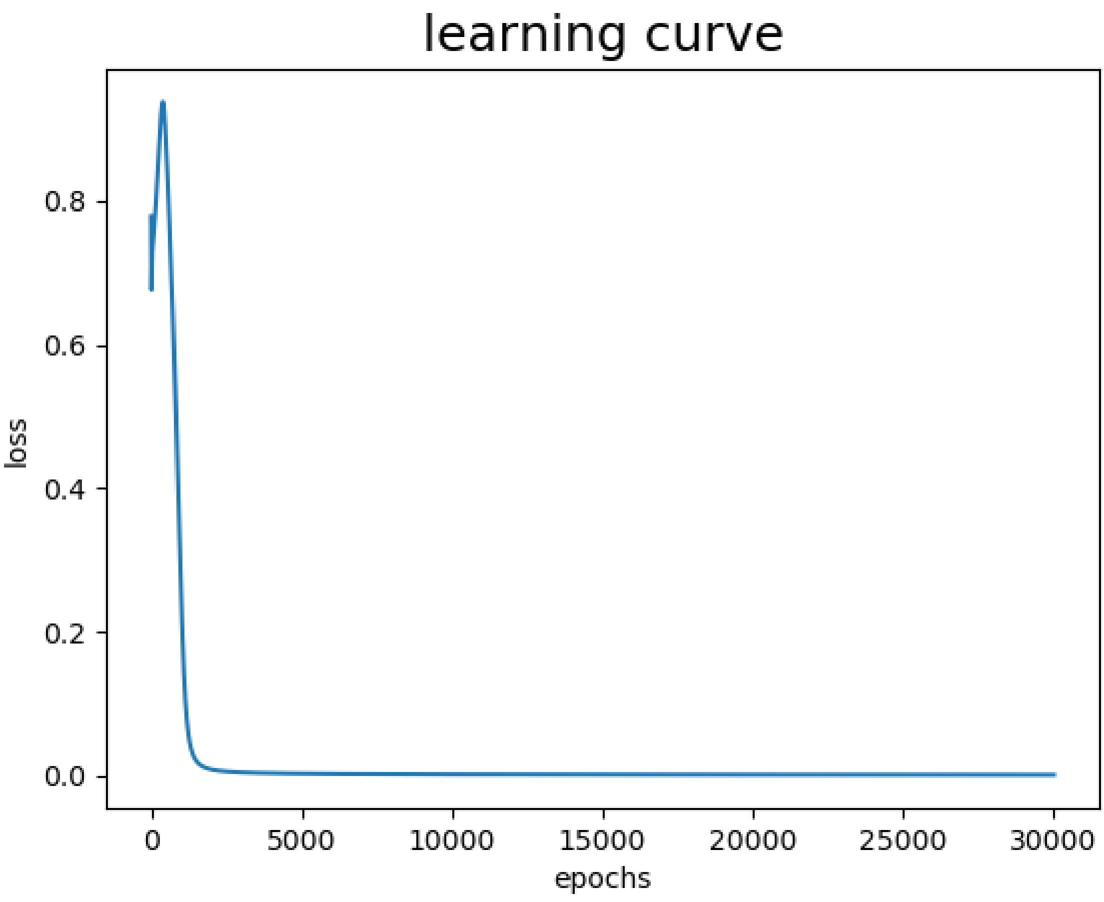
```
loss of 13501   epoch : 0.0016649699541065593
loss of 14001   epoch : 0.001637854220838964
loss of 14501   epoch : 0.0016123900660581672
loss of 15001   epoch : 0.0015884147280479884
loss of 15501   epoch : 0.0015657869986392514
loss of 16001   epoch : 0.0015443837009787751
loss of 16501   epoch : 0.0015240968436437308
loss of 17001   epoch : 0.0015048313032735836
loss of 17501   epoch : 0.0014865029238563248
loss of 18001   epoch : 0.0014690369471957285
loss of 18501   epoch : 0.0014523667086459326
loss of 19001   epoch : 0.001436432546850787
loss of 19501   epoch : 0.0014211808872995444
loss of 20001   epoch : 0.0014065634679520976
loss of 20501   epoch : 0.0013925366816824017
loss of 21001   epoch : 0.001379061015315206
loss of 21501   epoch : 0.0013661005689618647
loss of 22001   epoch : 0.0013536226424428032
loss of 22501   epoch : 0.0013415973780273162
loss of 23001   epoch : 0.0013299974506643223
loss of 23501   epoch : 0.0013187977984324445
loss of 24001   epoch : 0.0013079753871916466
loss of 24501   epoch : 0.0012975090044329985
loss of 25001   epoch : 0.0012873790781436692
loss of 25501   epoch : 0.0012775675171871545
loss of 26001   epoch : 0.0012680575702428453
loss of 26501   epoch : 0.0012588337008128782
loss of 27001   epoch : 0.0012498814761800045
loss of 27501   epoch : 0.0012411874685146372
loss of 28001   epoch : 0.0012327391665924814
loss of 28501   epoch : 0.0012245248968055047
loss of 29001   epoch : 0.0012165337523314267
loss of 29501   epoch : 0.0012087555294871355
accuracy: 100.0 %
```

C.Learning curve (loss, epoch curve)

- Linear data



- Nonlinear data



D. Anything you want to present

- Linear data
    - NN architecture : [2 8 6 1]
    - epochs :10000
    - acitvation function : sigmoid
    - learning_rate : 0.1
    - optimizer: SGD
    - batch_size: 10
- Nonlinear data
    - NN architecture : [2 6 8 1]
    - epochs : 30000
    - acitvation function : sigmoid
    - learning_rate : 0.1
    - optimizer: SGD
    - batch_size: 10

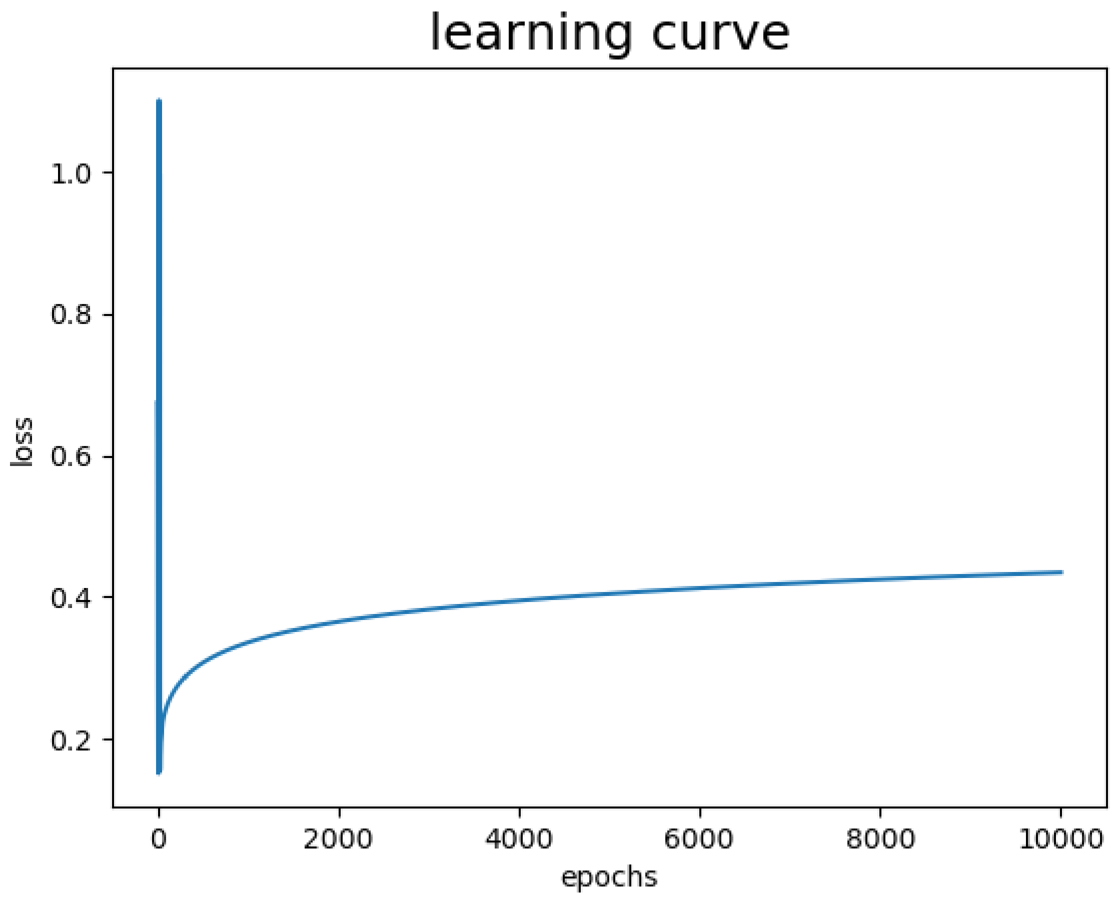
4. Discussion

A. Try different learning rates

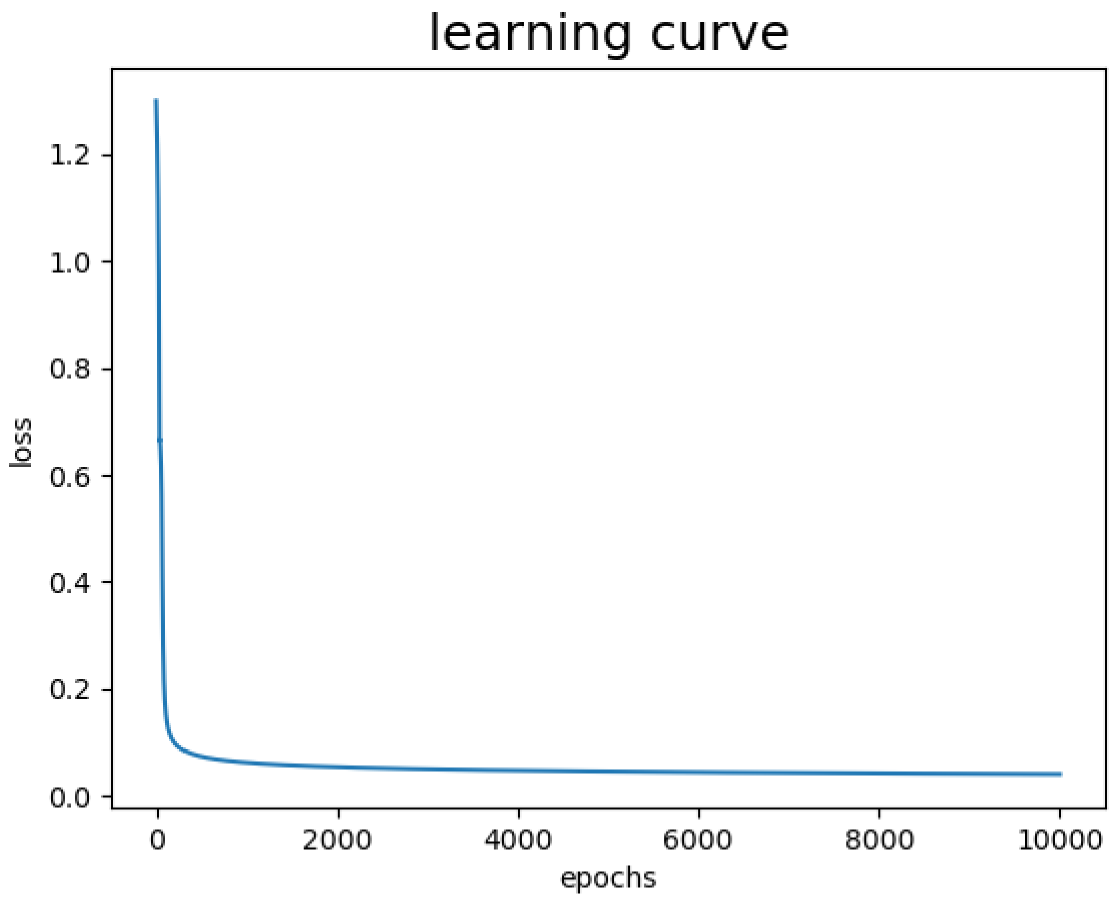
(take linear data for example)

learning rate	accuracy	loss
1	97.0%	0.43
0.1	100%	0.04
0.01	97%	0.085

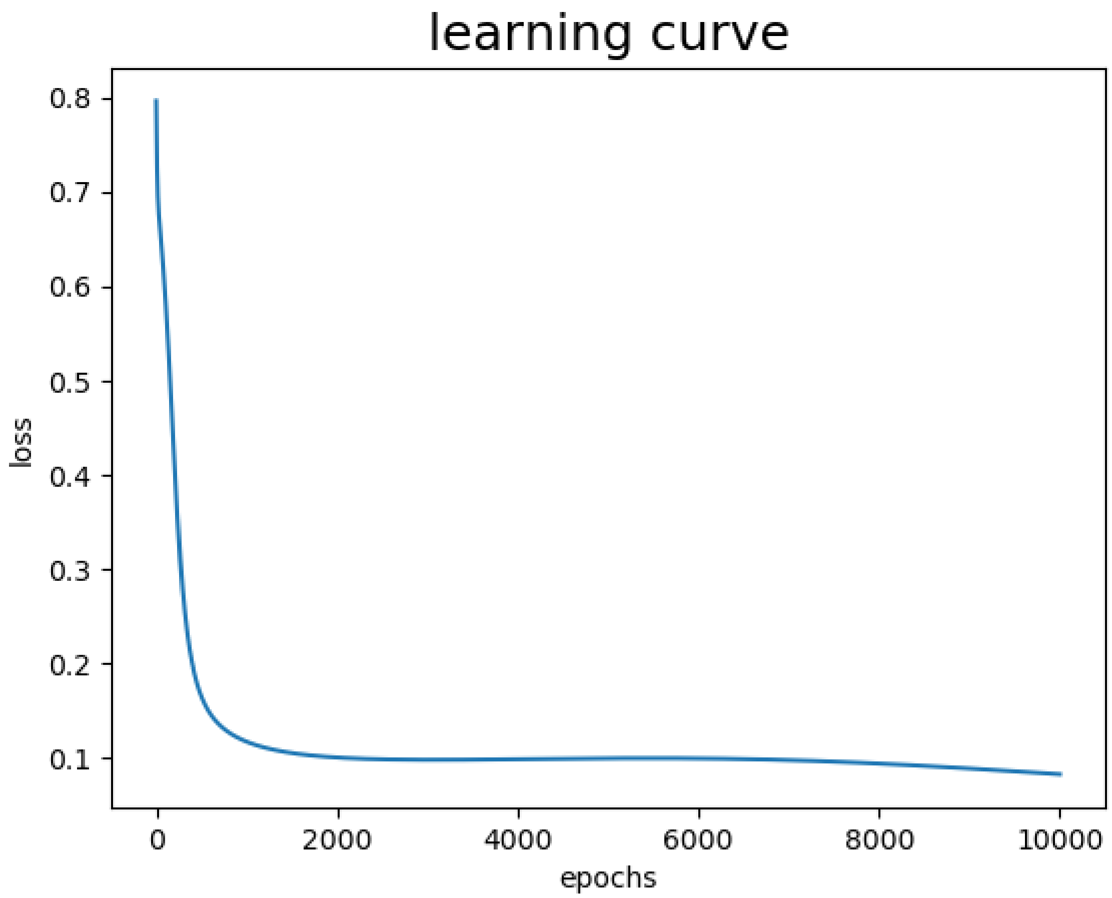
- learning_rate = 1



- learning_rate = 0.1



- learning_rate = 0.01

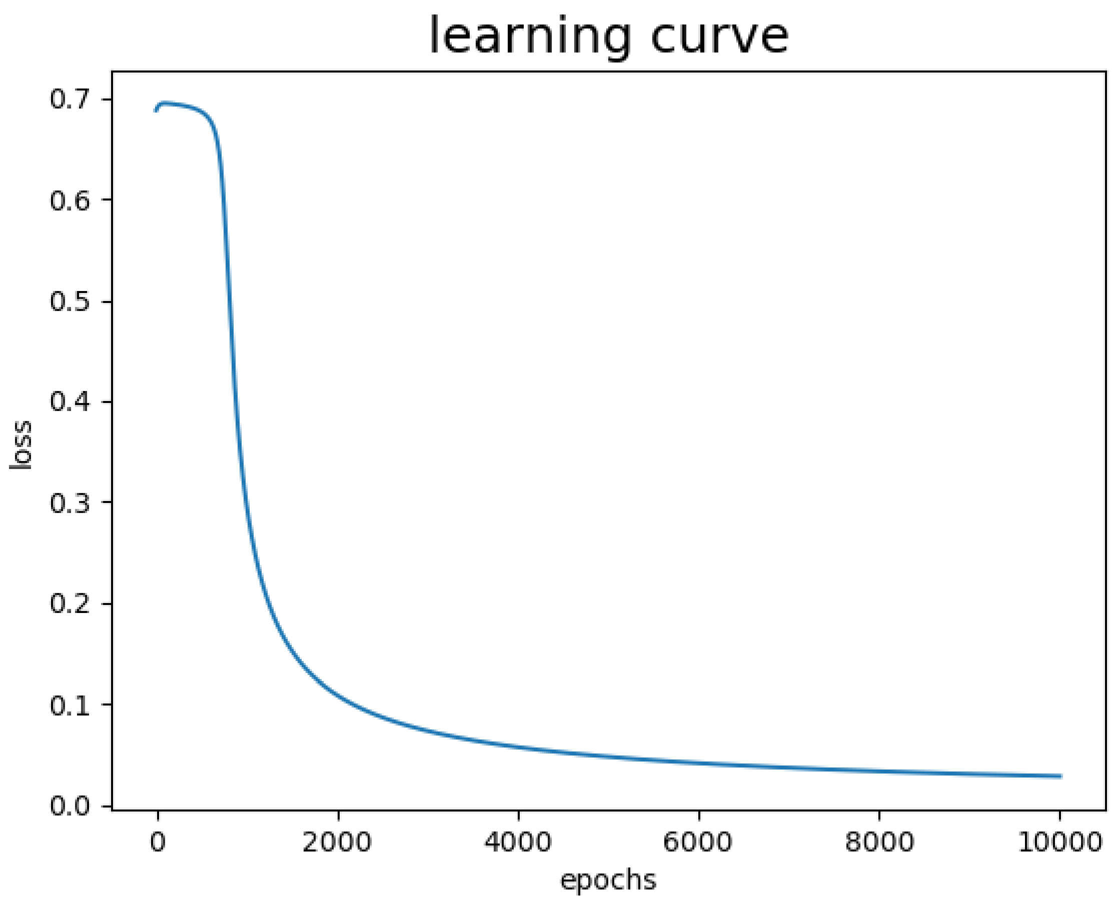


B. Try different numbers of hidden units

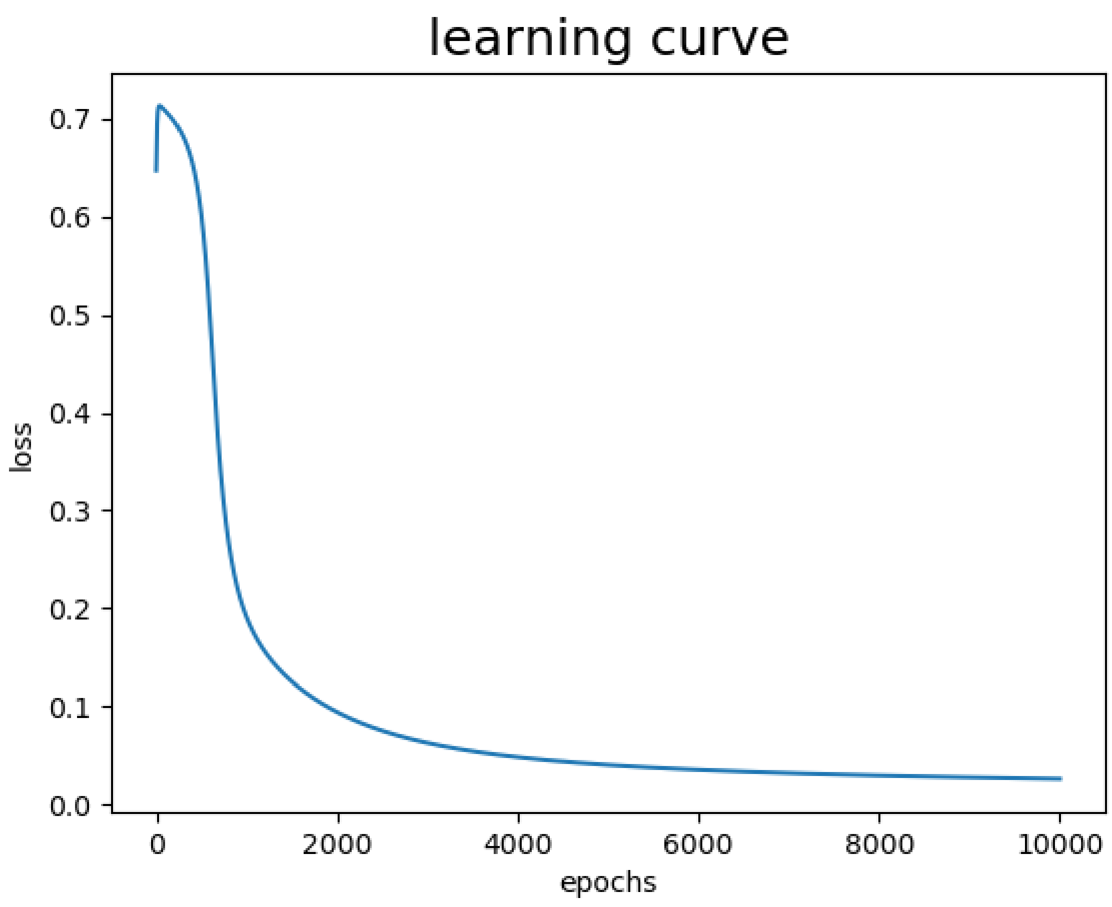
(take linear data for example)

NN architecture	accuracy	loss
[2 2 2 1]	100%	0.029
[2 5 5 1]	100%	0.0268
[2 8 8 1]	100%	0.008

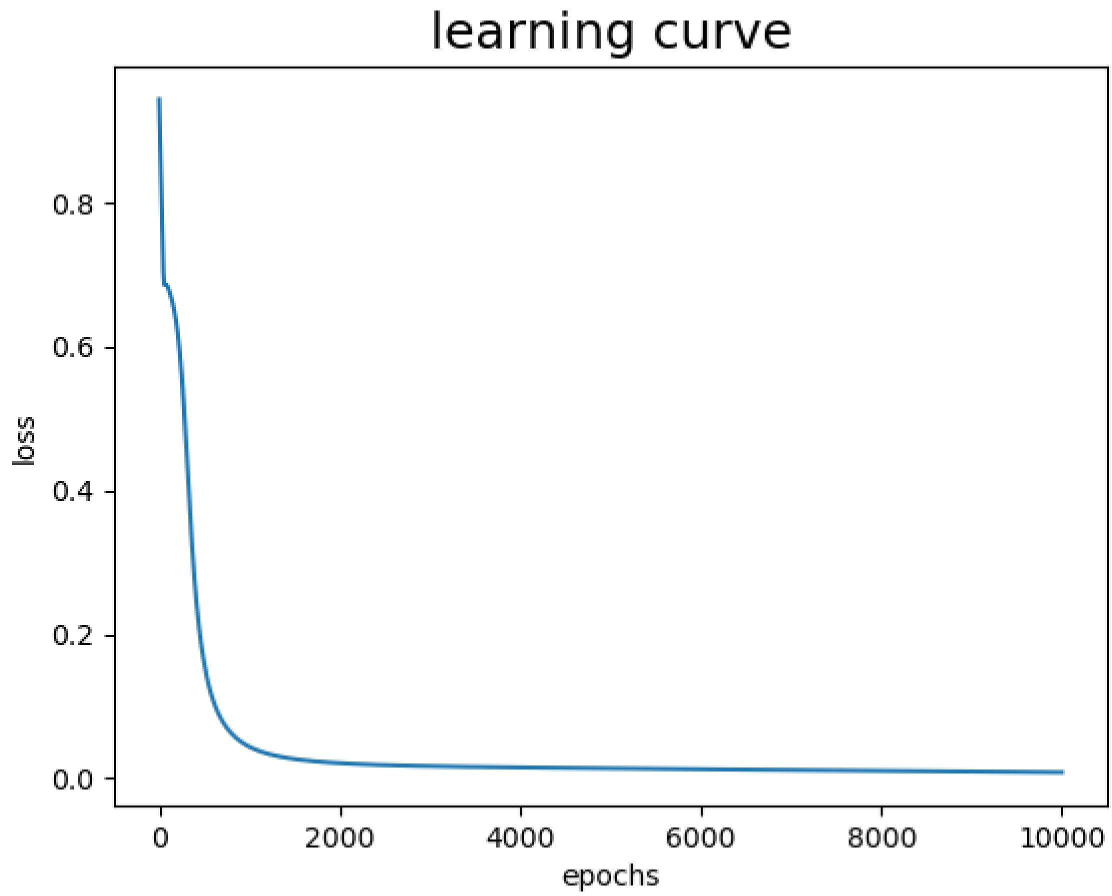
- NN architecture :[2 2 2 1]



- NN architecture :[2 5 5 1]



- NN architecture :[2 8 8 1]



C. Try without activation functions

Activation function	accuracy	loss
None	52% (basically do nothing)	this will make cross_entropy loss to nan
sigmoid	100%	0.071

- sigmoid activation function

```
loss of 9951   epoch : 0.07102493655882372
accuracy: 100.0 %
```

D. Anything you want to share

- In cross_entropy derivation, if you use sigmoid function, in the end you can get a very beautiful form of z_l, l is the last layer

$$\frac{\partial C}{\partial z_l} = -y + a_l$$

- However, due to the flexibility of changing the activation function, I write in more complicate form in my code .

```
self.derivatives['dz' + str(self.L)] = -((y*(1-al) - (1-y)*al)/al*(1-al))*derivative_activation(zl)
```

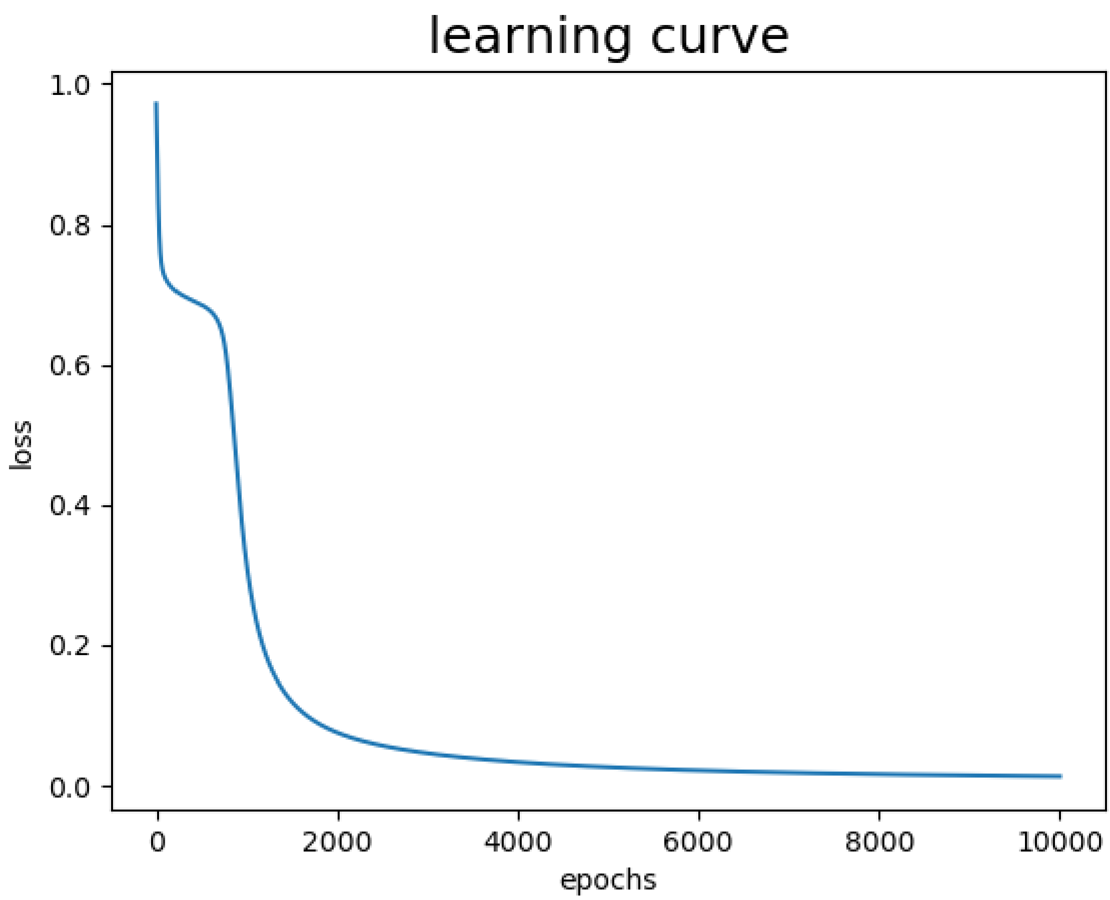
5. Extra

A. Implement different optimizers

Optimizer	accuracy	loss
SGD	100%	0.013
Momentum	100%	0.0013

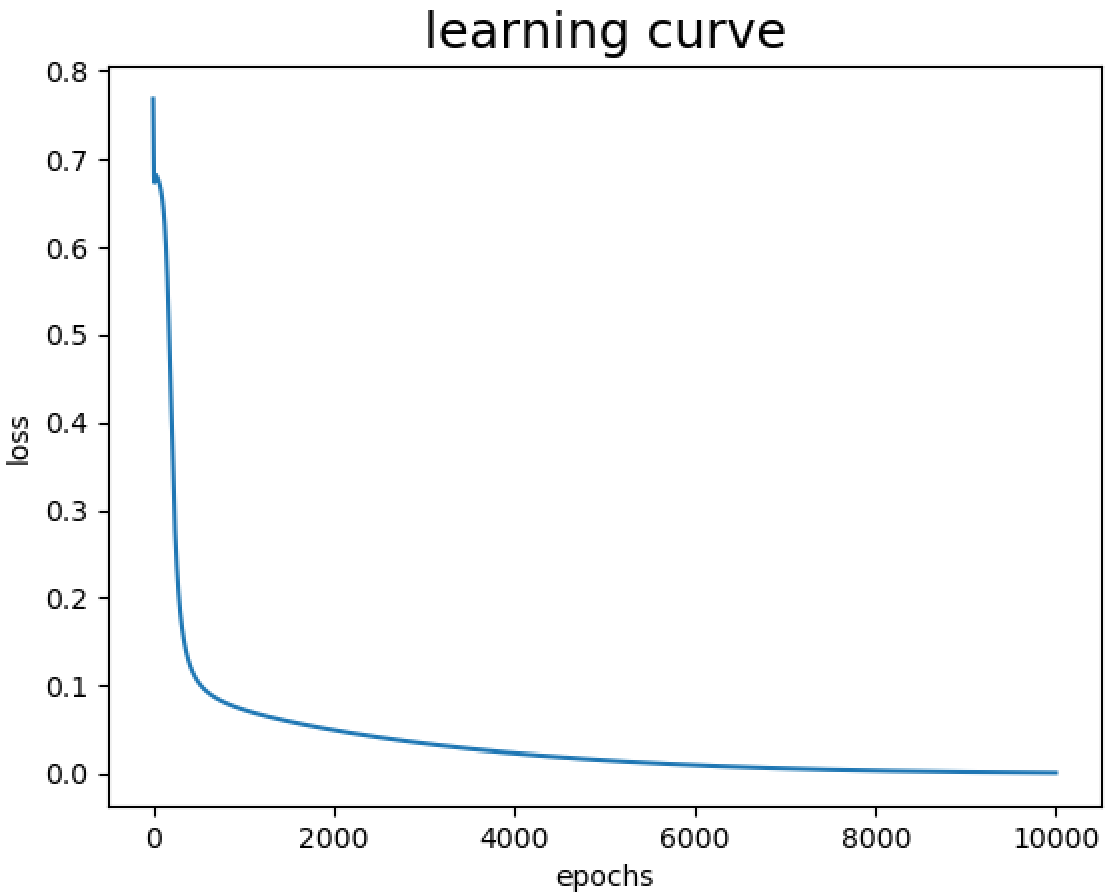
- SGD

```
loss of 9951   epoch : 0.013504929568568858
accuracy: 100.0 %
```



- Momentum

```
loss of 9951   epoch : 0.0013578157308374917
accuracy: 100.0 %
```

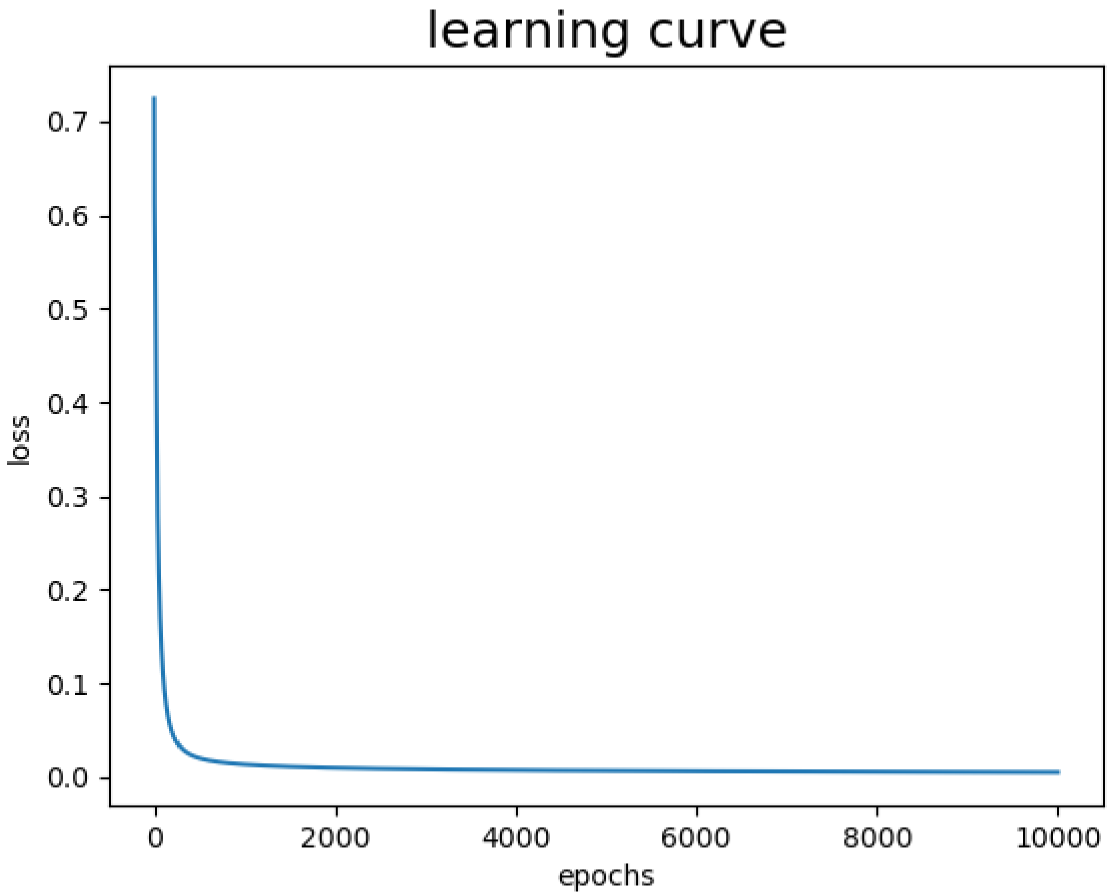


B. Implement different activation functions

activation function	accuracy	loss
tanh	100%	0.005
tanh	100%	0.0213

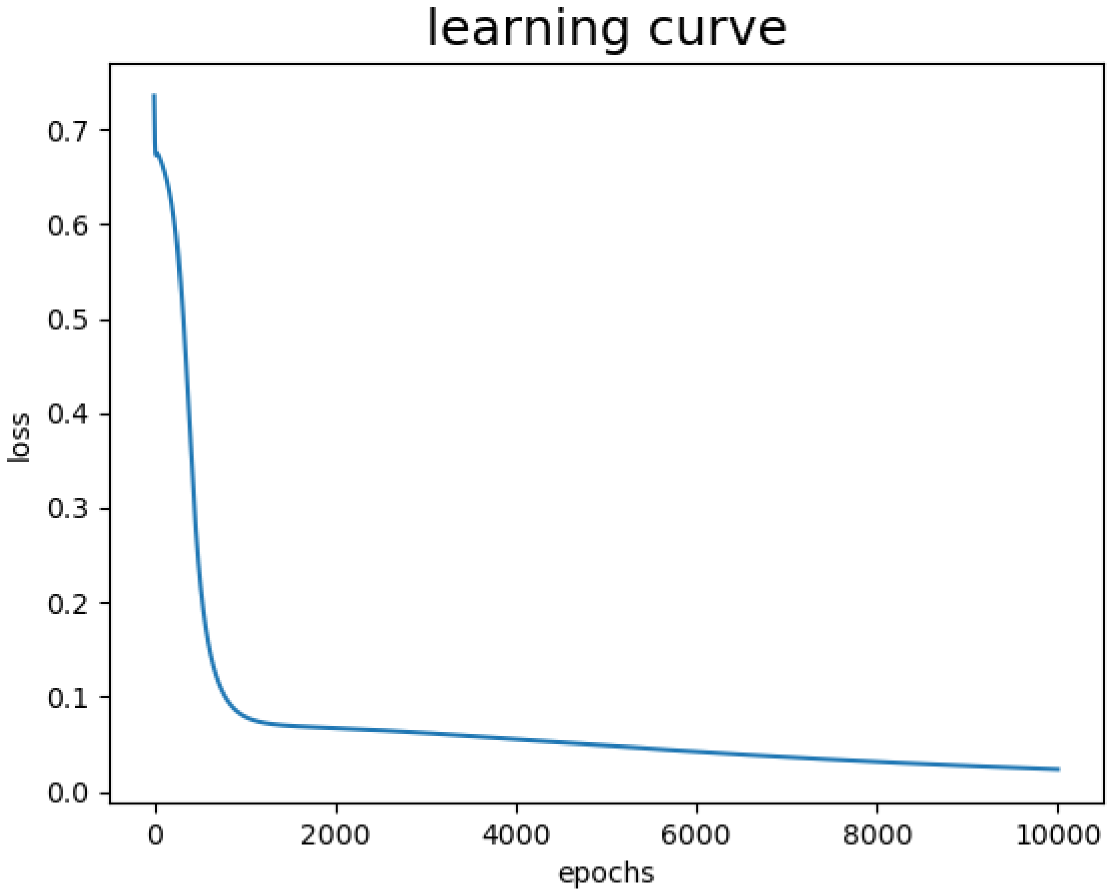
- tanh

```
loss of 9951   epoch : 0.005107314998680649
accuracy: 100.0 %
```



- sigmoid

```
loss of 9951   epoch : 0.023834822857344274
accuracy: 100.0 %
```



- tanh(投影版本) 推導
    - 由於 tanh 在 -1 ~ 1 之間
    - 因此我這邊的作法是讓它投影到 0 ~ 1 之間
    - 設$\overline{\tanh}$為投影版本的 $tanh$

$$\overline{\tanh}(x) = \left(\frac{\frac{e^x - e^{-x}}{e^x + e^{-x}} + 1}{2} \right)$$

    - 微分後可得

$$\begin{aligned} \frac{e^x}{e^x + e^{-x}} &= \frac{e^x\left(e^x + e^{-x}\right) - e^x\left(e^x - e^{-x}\right)}{\left(e^x + e^{-x}\right)^2} \\ &= \frac{e^{2x} + e^0 - e^{2x} + e^0}{\left(e^x + e^{-x}\right)^2} = \frac{2}{\left(e^x + e^{-x}\right)2} \end{aligned}$$