Lab2

310551094 郭家佑

1. Introduction (20%):

In the lab, we need to train and test the neural network for the two case. In the training model, we only use two hidden layer with non-linear function (Sigmoid) but it still can get good performance (about 20000 epoch to get 100% accuracy)

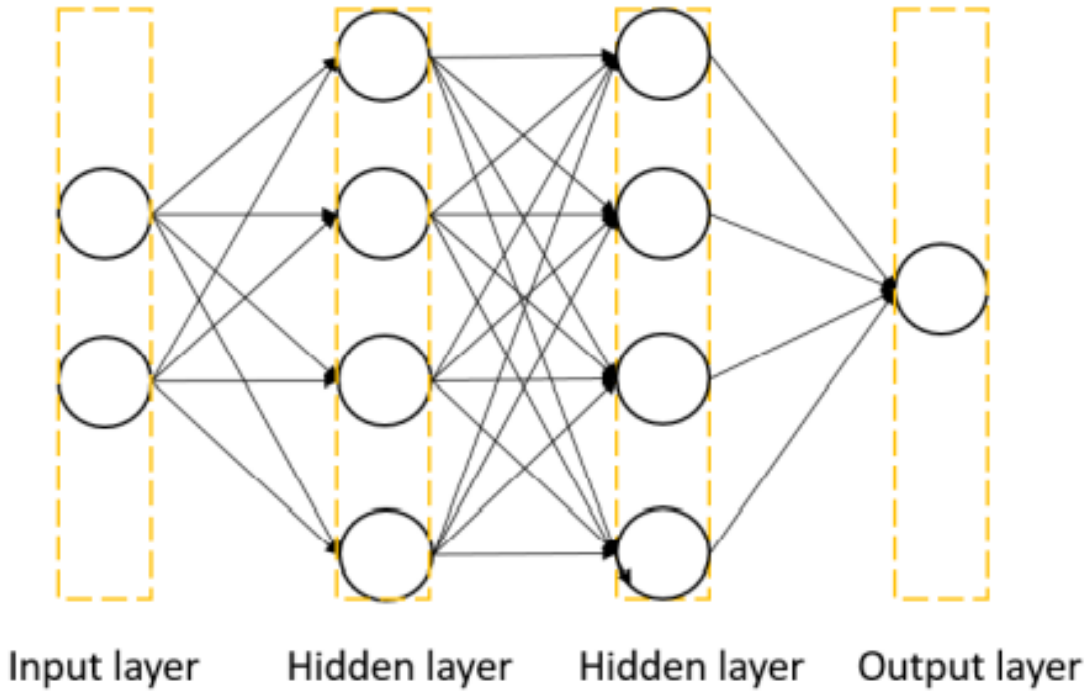


Figure 1. Two-layer neural network

2. Experiment setups (30%):

A. Sigmoid functions

To get non-linear feature, the lab using Sigmoid for threshold in default.

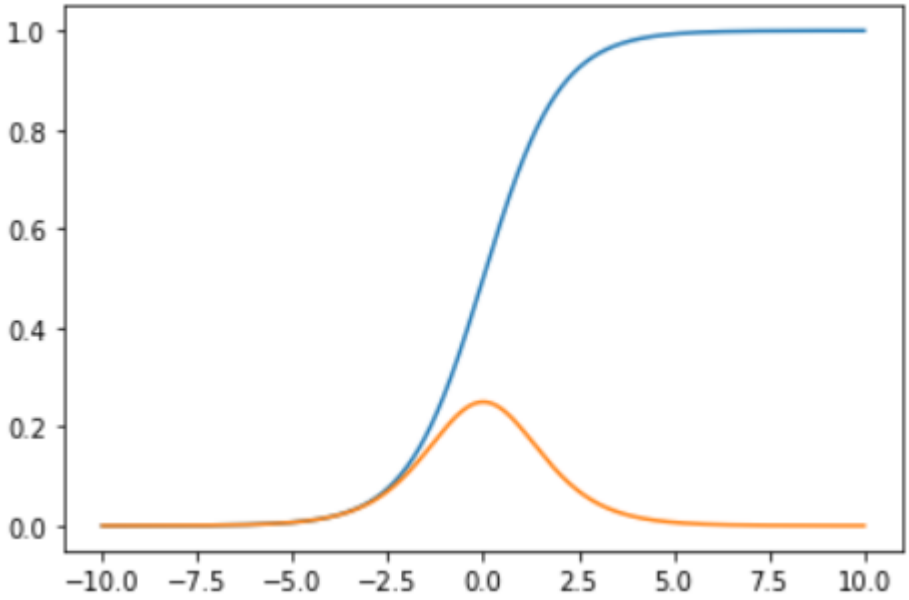
And the function in define is:

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

Also we need to get the derivative Sigmoid for “Backpropagation” step, then:

$$\sigma'(x) = \frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

Which plot on graph is look like:



We can find out that the segment near to 0 is similar to linear, also the derivative function has a hike there.

B. Neural network

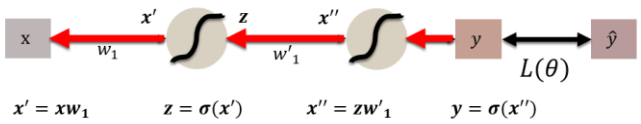
In the problem, I use the Figure. 1 to build my model. That is, only two hidden layer and each have 4 neural. And the part we can train the weight between each two continue layer which is W1, W2, W3.

I choose 20 for the learning rate, and it seem good enough.
And I run 30000 epochs to train the network. We can see that it converge after about 20000 epochs.

C. Backpropagation

The final target is make the prediction be more accurate, so we try to get the

$\frac{\partial L(\theta)}{\partial w_1}$ to know the “way” for the matrix W modified.



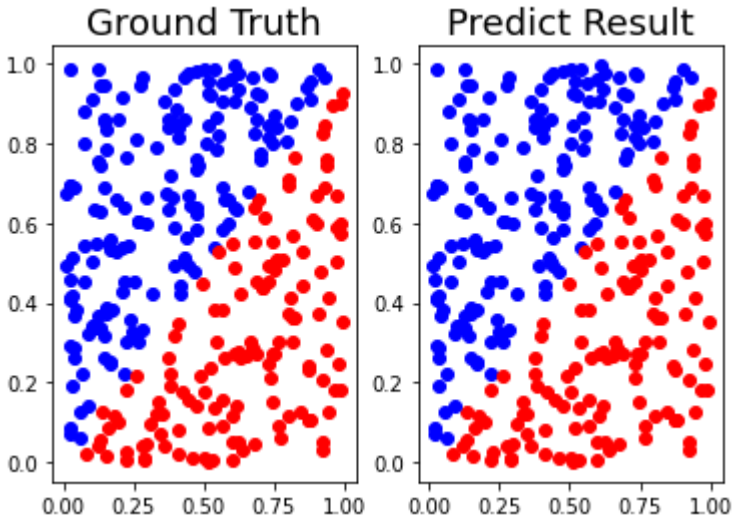
$$\frac{\partial L(\theta)}{\partial w_1} = \frac{\partial y}{\partial w_1}\frac{\partial L(\theta)}{\partial y}$$
$$= \frac{\partial x''}{\partial w_1}\frac{\partial y}{\partial x''}\frac{\partial L(\theta)}{\partial y}$$
$$= \frac{\partial z}{\partial w_1}\frac{\partial x''}{\partial z}\frac{\partial y}{\partial x''}\frac{\partial L(\theta)}{\partial y}$$
$$= \frac{\partial x'}{\partial w_1}\frac{\partial z}{\partial x'}\frac{\partial x''}{\partial z}\frac{\partial y}{\partial x''}\frac{\partial L(\theta)}{\partial y}$$

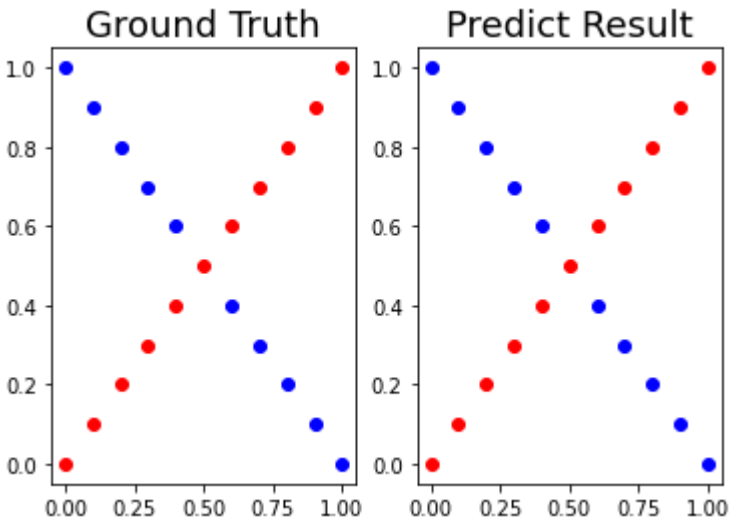
In the slide's figure we know that we can use chain rule to reduce computation. Each item only need to compute once. So in one time backward computation, we can get the value that all of W1, W2, W3 partial derivative to the $L(\theta)$.

Then using my learning rate (I set 20 in the lab) try to get better matrix.

2. Results of your testing (20%)

A. Screenshot and comparison figure



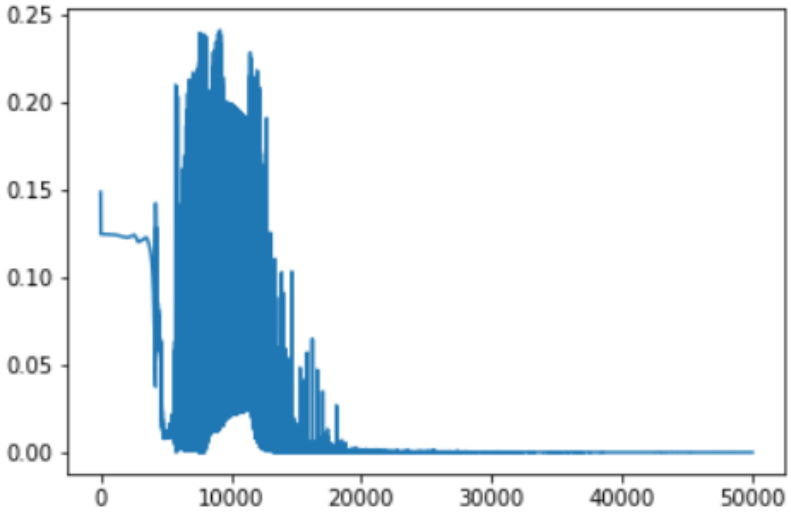


B. Show the accuracy of your prediction

As the upper picture show, both are 100% accuracy.

C. Learning curve (loss, epoch curve)

XOR case:



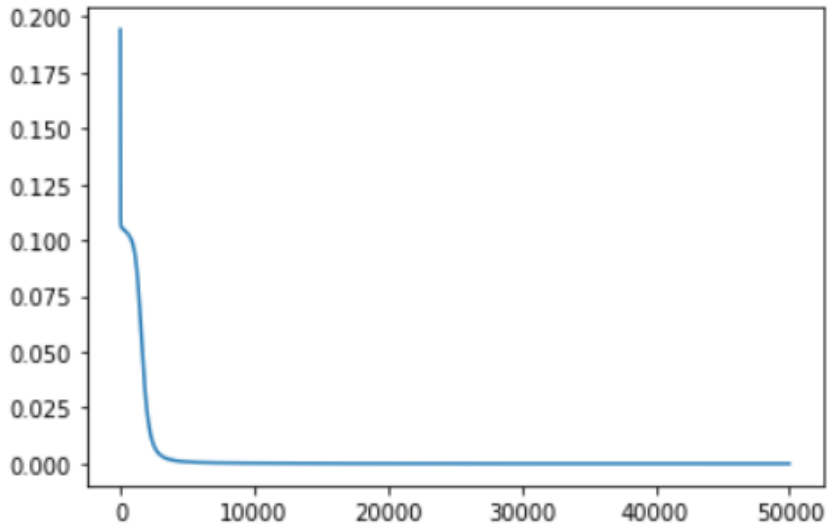
The left table are the epochs and the cost, the upper figure is the visualization.

The right table are the what it’s prediction at last.

```
This is XOR case:
cur epoch num is : 0 , cost: 0.14858156321212168
cur epoch num is : 1000 , cost: 0.12425785643492533
cur epoch num is : 2000 , cost: 0.12262139341896365
cur epoch num is : 3000 , cost: 0.12044888445024976
cur epoch num is : 4000 , cost: 0.10102670681249884
cur epoch num is : 5000 , cost: 0.008807945465424207
cur epoch num is : 6000 , cost: 0.007906893794334388
cur epoch num is : 7000 , cost: 0.0086392671379806
cur epoch num is : 8000 , cost: 0.008800509048932901
cur epoch num is : 9000 , cost: 0.10730157136852195
cur epoch num is : 10000 , cost: 0.05591086471830149
cur epoch num is : 11000 , cost: 0.0552281756193206
cur epoch num is : 12000 , cost: 0.012881339567868704
cur epoch num is : 13000 , cost: 0.001813435443554765
cur epoch num is : 14000 , cost: 0.0058018154605543075
cur epoch num is : 15000 , cost: 0.00044875612597560253
cur epoch num is : 16000 , cost: 0.00014824337767191539
cur epoch num is : 17000 , cost: 6.568463695262769e-05
cur epoch num is : 18000 , cost: 0.000276368248346733
cur epoch num is : 19000 , cost: 6.3568429963384e-05
cur epoch num is : 20000 , cost: 5.199714457882501e-05
cur epoch num is : 21000 , cost: 2.74591569795139e-05
cur epoch num is : 22000 , cost: 3.362300435022734e-05
cur epoch num is : 23000 , cost: 0.00012014937583599386
cur epoch num is : 24000 , cost: 2.40465751722516e-05
cur epoch num is : 25000 , cost: 1.3914440791675918e-05
cur epoch num is : 26000 , cost: 1.3691890750446639e-05
cur epoch num is : 27000 , cost: 1.2232772803360705e-05
cur epoch num is : 28000 , cost: 1.4665326778190639e-05
cur epoch num is : 29000 , cost: 7.4041102284531e-05
cur epoch num is : 30000 , cost: 6.91919260233977e-05
```

```
[[1.29466958e-09]
 [9.96850406e-01]
 [5.63786827e-03]
 [9.96850406e-01]
 [4.51784496e-03]
 [9.96850406e-01]
 [3.63808644e-03]
 [9.96850406e-01]
 [2.94594103e-03]
 [9.96850406e-01]
 [2.40018726e-03]
 [1.96866703e-03]
 [9.97044833e-01]
 [1.62635607e-03]
 [9.97044833e-01]
 [1.35381299e-03]
 [9.97044833e-01]
 [1.13594584e-03]
 [9.97044833e-01]
 [9.61039879e-04]
 [9.97044833e-01]]
```

Linear case:



The left table are the epochs and the cost, the upper figure is the visualization.

The right table are the what it's prediction at last.

```
This is linear case:
cur epoch num is : 0 , cost: 0.19417747009610573
cur epoch num is : 1000 , cost: 0.09662674996482155
cur epoch num is : 2000 , cost: 0.023189571322107676
cur epoch num is : 3000 , cost: 0.003833999950975423
cur epoch num is : 4000 , cost: 0.00146645225375344
cur epoch num is : 5000 , cost: 0.0007994105618566027
cur epoch num is : 6000 , cost: 0.0005263673047131222
cur epoch num is : 7000 , cost: 0.0003895866425930934
cur epoch num is : 8000 , cost: 0.00031073520595373494
cur epoch num is : 9000 , cost: 0.000259648531183119
cur epoch num is : 10000 , cost: 0.00022289425734800227
cur epoch num is : 11000 , cost: 0.00019398337206111906
cur epoch num is : 12000 , cost: 0.0001697017345321446
cur epoch num is : 13000 , cost: 0.00014849562260806656
cur epoch num is : 14000 , cost: 0.00012965611380560003
cur epoch num is : 15000 , cost: 0.0001128735483189468
cur epoch num is : 16000 , cost: 9.799595947064719e-05
cur epoch num is : 17000 , cost: 8.490984868545891e-05
cur epoch num is : 18000 , cost: 7.34923880324203e-05
cur epoch num is : 19000 , cost: 6.360098041941236e-05
cur epoch num is : 20000 , cost: 5.5078863210476524e-05
cur epoch num is : 21000 , cost: 4.776505265363542e-05
cur epoch num is : 22000 , cost: 4.150328176527217e-05
cur epoch num is : 23000 , cost: 3.6148190283683755e-05
cur epoch num is : 24000 , cost: 3.15687458662329e-05
cur epoch num is : 25000 , cost: 2.7649507350132964e-05
cur epoch num is : 26000 , cost: 2.4290438809918086e-05
cur epoch num is : 27000 , cost: 2.1405869714370155e-05
cur epoch num is : 28000 , cost: 1.892303211474554e-05
cur epoch num is : 29000 , cost: 1.6780457163344665e-05
cur epoch num is : 30000 , cost: 1.492640009352128e-05
```

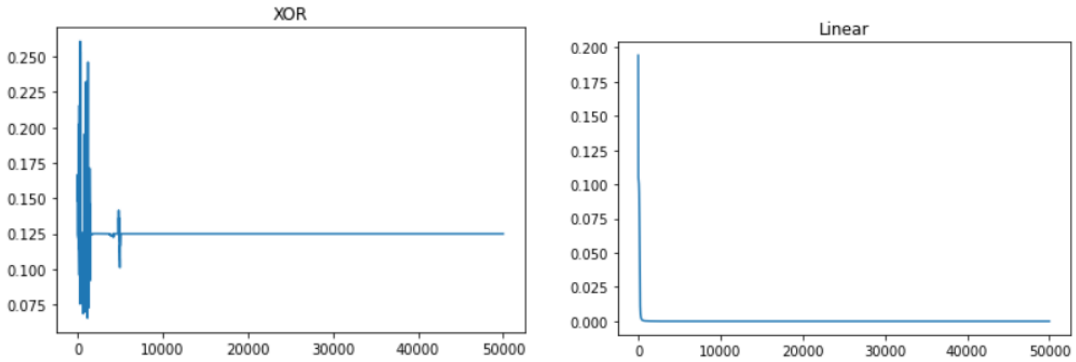
```
[[9.99447211e-01]
 [9.99714085e-01]
 [7.30845262e-03]
 [9.99708646e-01]
 [9.99714085e-01]
 [9.99714085e-01]
 [9.99714085e-01]
 [9.99714085e-01]
 [9.99714085e-01]
 [2.23648420e-05]
 [9.99714085e-01]
 [2.22966752e-05]
 [9.99714085e-01]
 [1.39382439e-03]
 [9.99714085e-01]
 [4.99700958e-05]
 [9.90955685e-01]
 [9.99714085e-01]
 [5.79012312e-05]
 [2.38493353e-03]
 [9.99704910e-01]
 [9.29845067e-05]
 [9.99714085e-01]
 [6.75101671e-05]
 [9.99493783e-01]
 [9.99714085e-01]
 [9.99714085e-01]
 [9.99714085e-01]
 [9.99714085e-01]
 [9.99714085e-01]]
```

D. anything you want to present

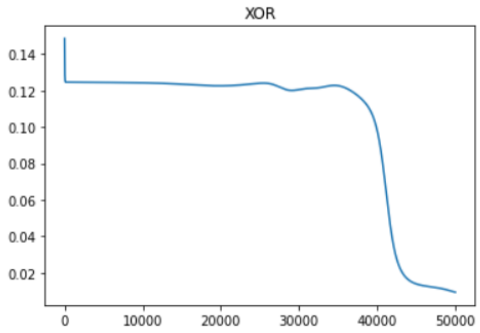
4. Discussion (30%)

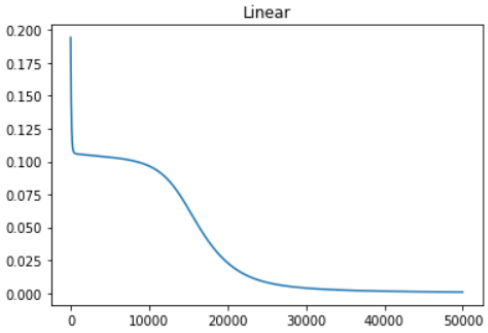
A. Try different learning rates

When we use 10 times learning rate, both case learning more quickly. However, XOR case can only stop at the 1.25 location due to it can't converge as successfully as we want.



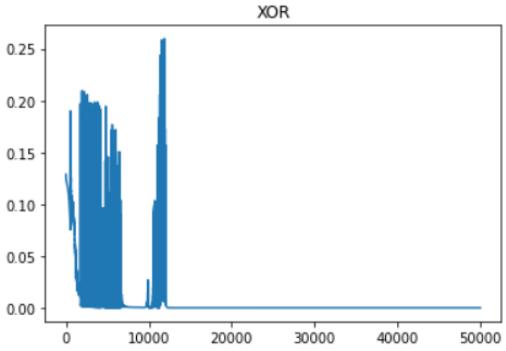
Then we tried the 0.1 time learning rate, both two case still can converge successfully, but it cost almost 50000 epochs to converge.

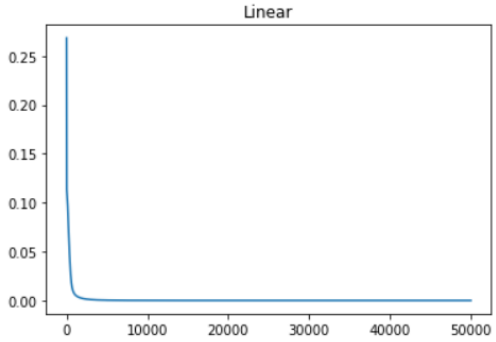




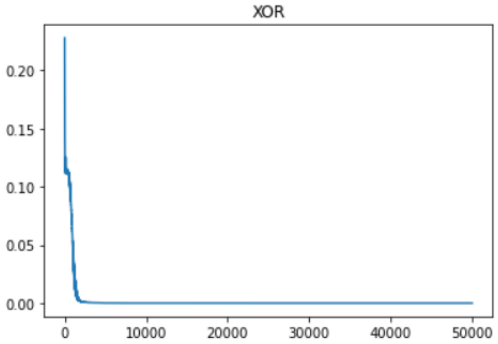
B. Try different numbers of hidden units

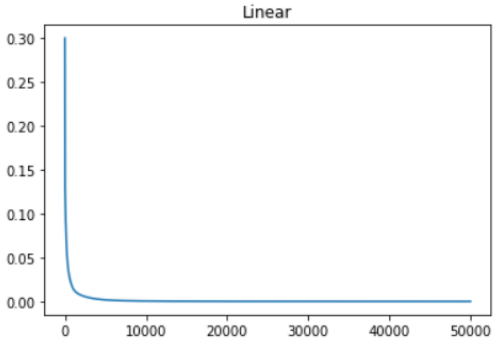
When we modified the unit to 10, then we get the following figures we find that is converge more quickly.





When we modified the unit to 30, then we get the following figures we find that is converge more quickly than the 10 case.



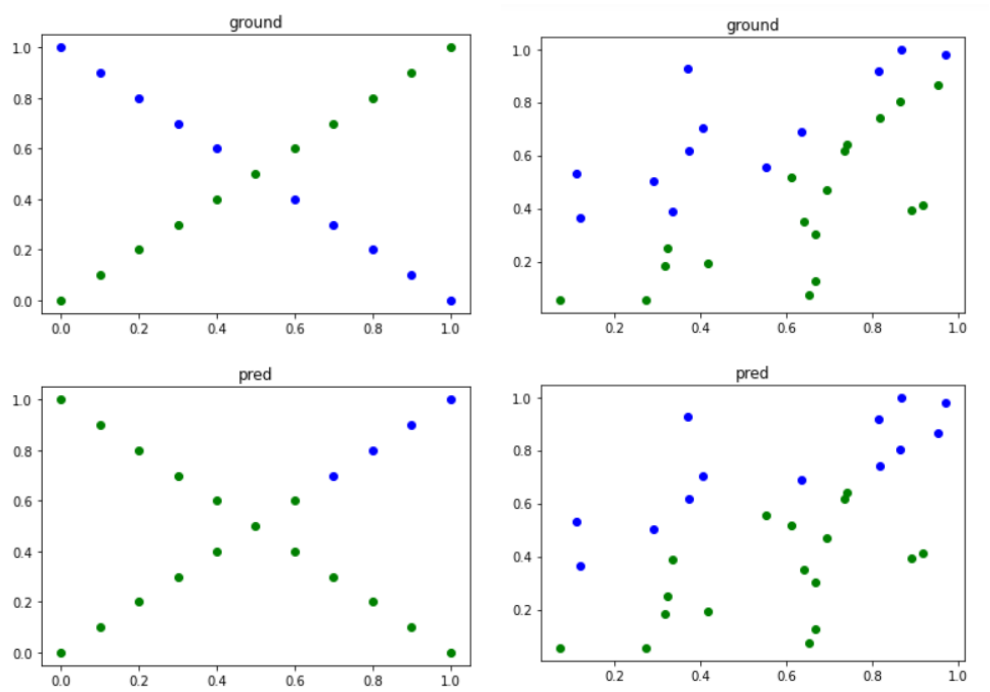


It should be trained more quickly when we use more unit.

C. Try without activation functions

Without the linear function, the case would be totally linear case. Because A * B can be represent to A' with combination. And the area can only represent a line in a planar range.

The result show that the linear case can be learning more accurate than XOR case, also the miss only happened near by the baseline.



D. Anything you want to share

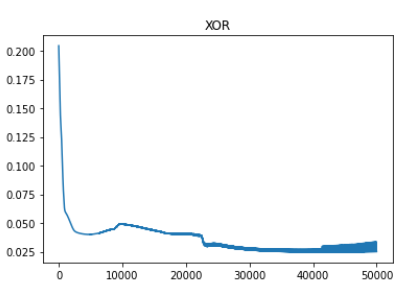
5. Extra (10%)

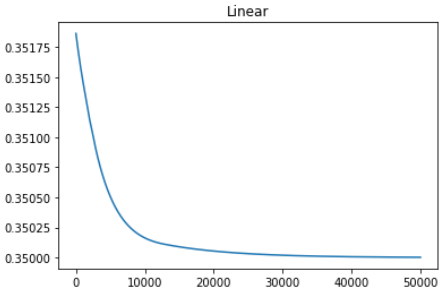
A. Implement different optimizers. (2)

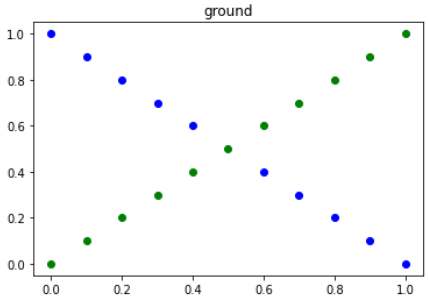
B. Implement different activation functions. (3)

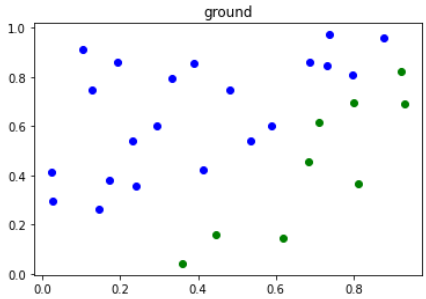
I tried to use the two common activation to tried.

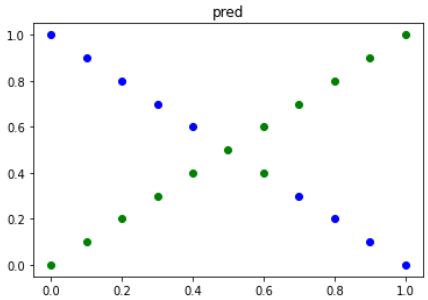
Fist one is ReLU, both of case can be converged, but the Linear case obviously locate at partial optimal place, and that easy class all vertex to same class (even if n be larger)

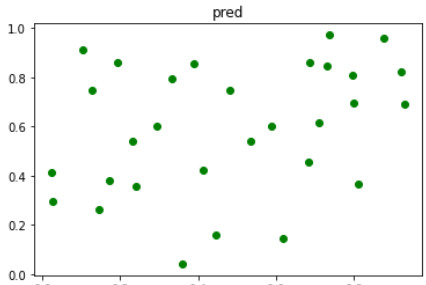






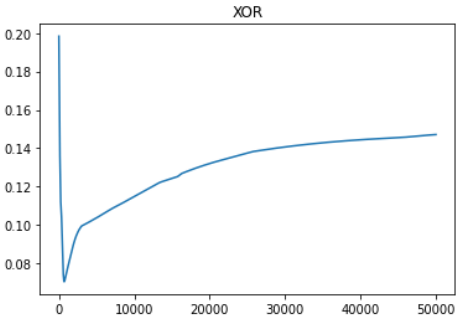


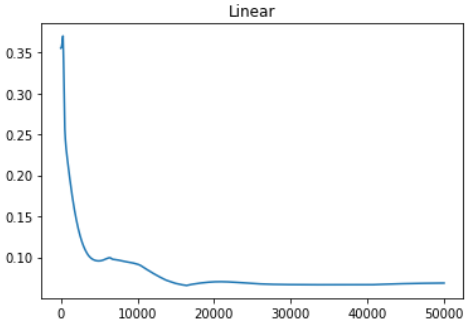


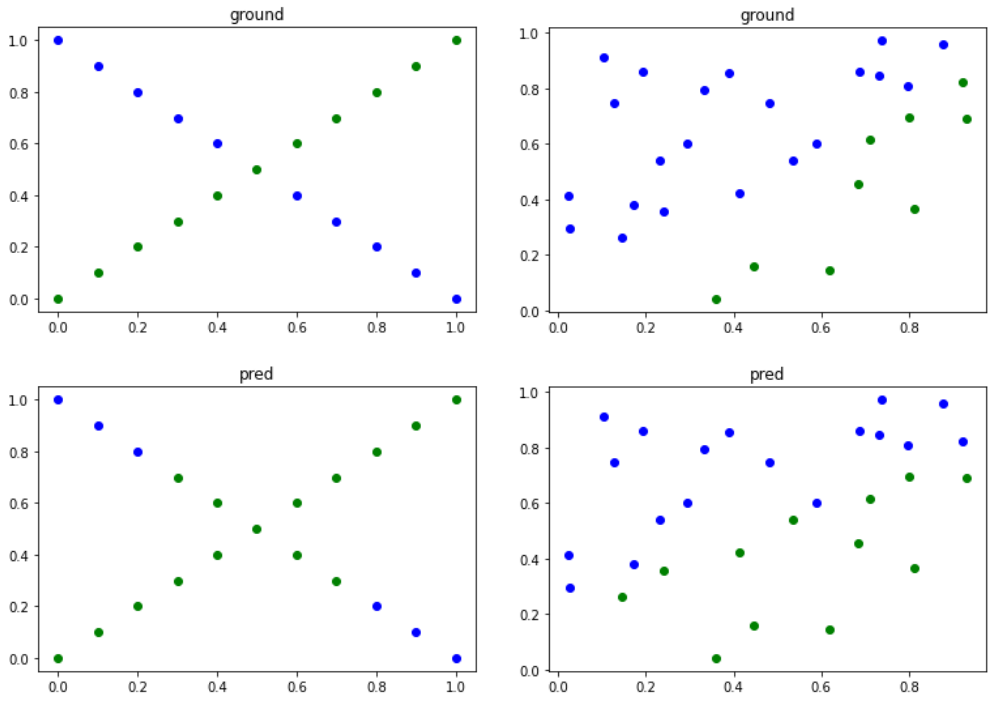


Second one is Leaky ReLU

Not sure why but the XOR case obviously not have good performance, converge to a weird place.







C. Implement convolutional layers. (5)