Lab2

March 17, 2022

機器人碩士學位學程 310605019 劉孟學

1 Introduction

In this lab, you will need to understand and implement simple neural networks with forwarding pass and backpropagation using two hidden layers. Notice that you can only use Numpy and the python standard libraries, any other frameworks (ex : Tensorflow、PyTorch) are not allowed in this lab.

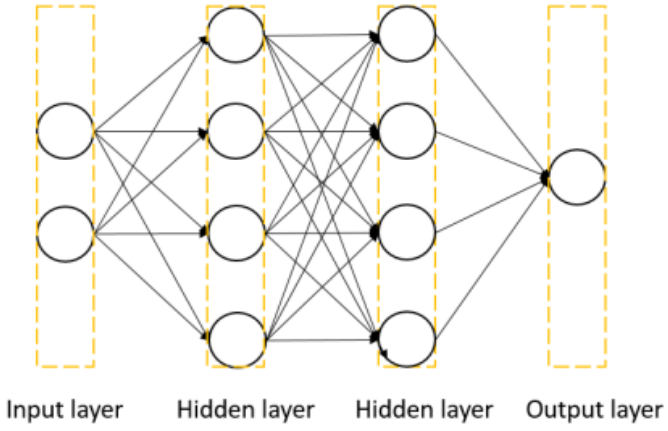


Figure 1. Two-layer neural network

2 Experiment setups

A. Sigmoid functions

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

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

$$= -(1+e^{-x})^2 \frac{d}{dx}(1+e^{-x})$$

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

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

B. Neural network

I use the setting of [2 6 4 1] cells in each layer

B.1. Define weight and weight gradient.

```
#weight in three layers
w1 = np.random.normal(size=(2,6))
w2 = np.random.normal(size=(6,4))
w3 = np.random.normal(size=(4,1))

w1_grad = np.zeros((2,6))
w2_grad = np.zeros((6,4))
w3_grad = np.zeros((4,1))
```

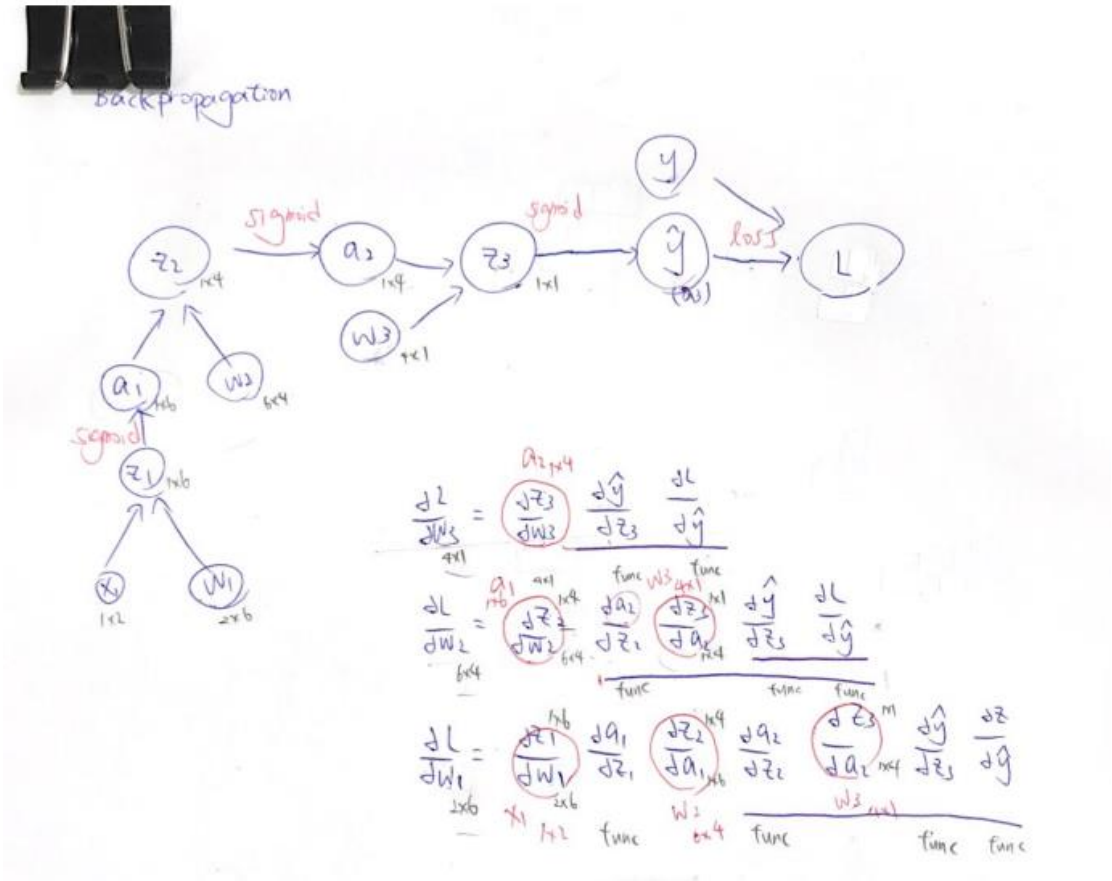
B.2 Forward calculation

```
#moving forward
a1 = sigmoid(np.matmul(x1[i], w1)).reshape((1,-1))
a2 = sigmoid(np.matmul(a1, w2))
a3 = sigmoid(np.matmul(a2, w3))
```

B.3 count loss

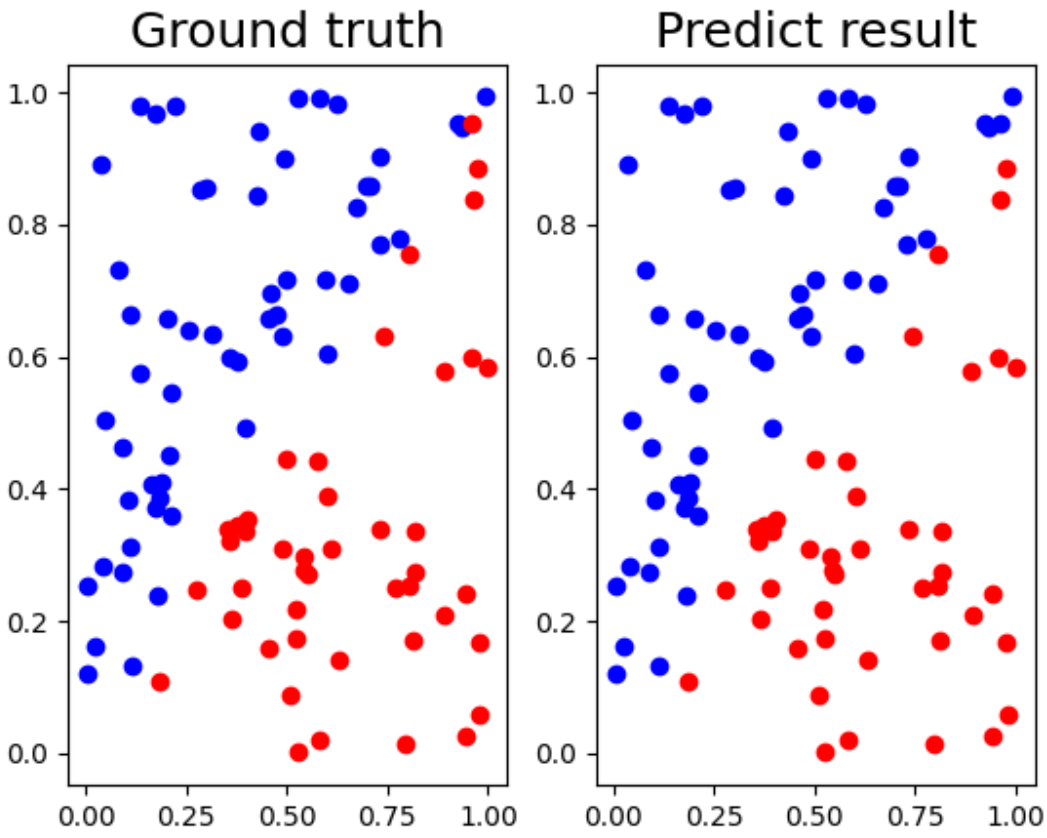
```
#count loss
loss = loss_func(a3.reshape((1)), y1[i])
total_loss += loss
print("epoch: ", epoch, "loss: ",total_loss/(i+1))
loss_arr1.append(total_loss/(i+1))
```

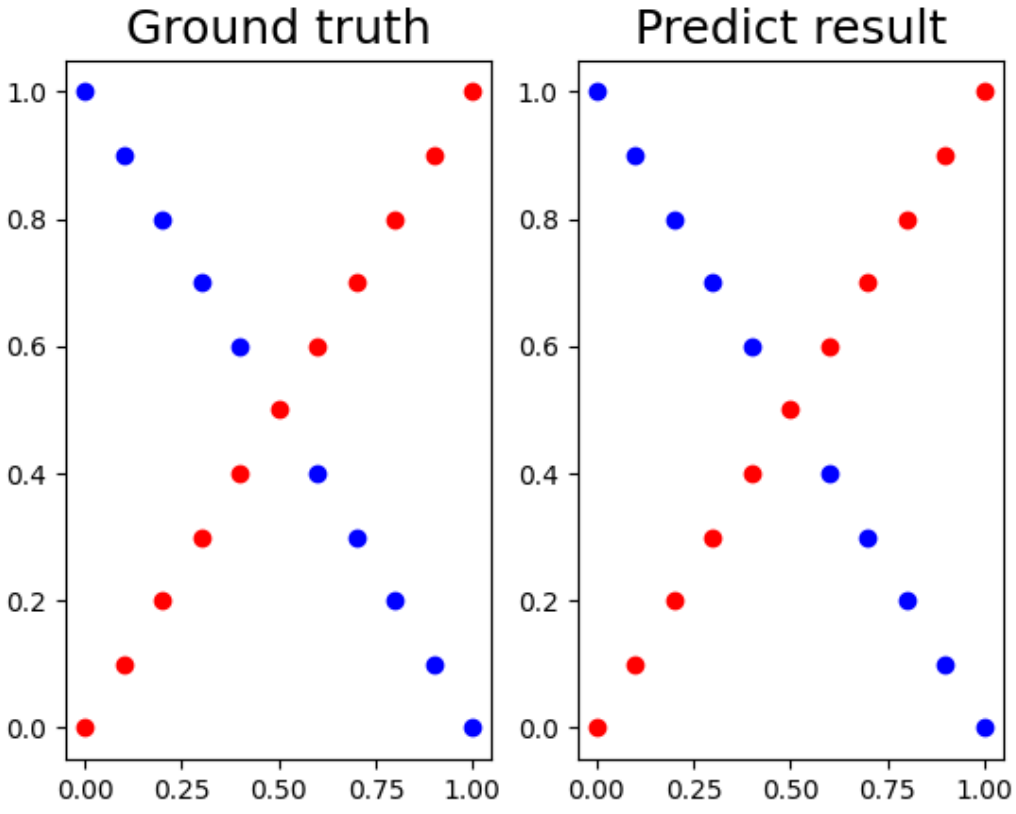
C. Backpropagation



3. Result of my testing

A. Screenshot and comparison figure


Ground truth
Predict result
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.25
0.50
0.75
1.00


Ground truth
Predict result
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.25
0.50
0.75
1.00

B. Show the accuracy of your prediction

```
linear data accuracy:  0.99 XOR data accuracy:  1.0
```

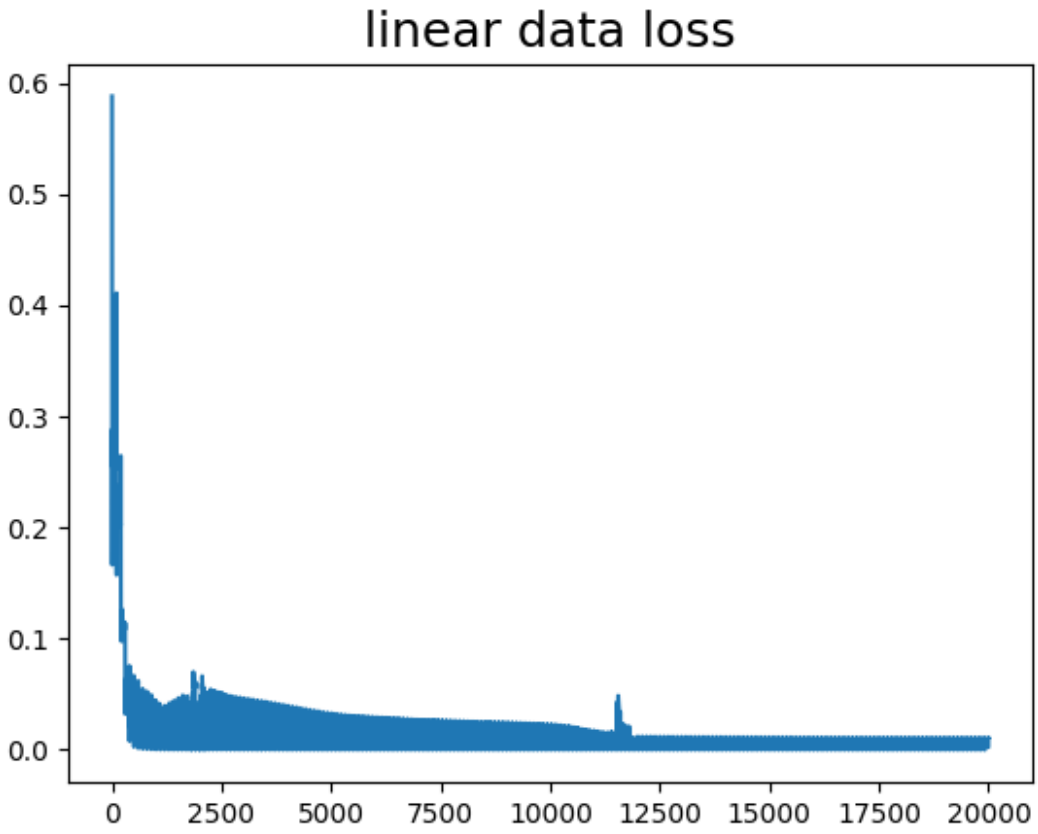
Linear data prediction

```
[[0.00010058]]
[[0.9999848]]
[[0.99995759]]
[[0.99980624]]
[[0.99998623]]
[[9.86092125e-05]]
[[0.00017818]]
[[0.99998625]]
[[9.86525487e-05]]
[[9.86025014e-05]]
[[0.9999857]]
[[9.86595919e-05]]
[[0.99998031]]
[[0.99998484]]
[[9.88966537e-05]]
[[0.73360286]]
[[0.99998495]]
[[0.99998604]]
[[9.88551781e-05]]
[[0.00010101]]
[[0.00037882]]
[[9.88608255e-05]]
[[0.99998519]]
[[9.86391456e-05]]
[[0.99998624]]
[[0.00830218]]
[[0.99997912]]
[[0.04306172]]
[[0.99926349]]
```

XOR data prediction

```
[[0.01175799]]
[[0.96636914]]
[[0.02403906]]
[[0.9662379]]
[[0.05563756]]
[[0.96646646]]
[[0.10389844]]
[[0.96628963]]
[[0.13316572]]
[[0.8938279]]
[[0.12774205]]
[[0.1043155]]
[[0.83850993]]
[[0.0800953]]
[[0.9418523]]
[[0.06159693]]
[[0.94896619]]
[[0.04894343]]
[[0.95039562]]
[[0.04054805]]
[[0.95090692]]
```

C. Learning curve (loss, epoch curve)



D. Anything you want to present

I found that the number of random seed can have great influence on the performance, the XOR data performance can even drop to about 75%, I think maybe more epoch and lower learning rate can save the issue.

4. Discussion

A. Try different learning rates

I try learning rate as 0.1, 0.001, 0.001, and it all result in the same performance, including accuracy and loss, I think it's because that the task is an easy one for neural network to make correct classification, so the learning rate does not have too much influence.

B. Try different numbers of hidden units

In my original setting, I use [2 6 4 1] cells in each layers and get 99% accuracy performance in linear data case, but when I try [2 10 10 1] and [2 6 6 1] it all decrease to 97%~96% in linear data case.

C. Try without activation functions

Since the perceptron is a linear model, if we do not use the activation function, the four-layer neural network is actually equal to just one-layer neural network, and it can only solve the linear data task, but XOR data is not linear seperatable, so it would have good performance in XOR data case.

D. Anything you want to share

During the coding part, some of the numpy function is really confusing like the function “@”,” numpy.dot()”,” numpy.matmul()” and “*”, it is sometimes equivalent, but sometimes totally different, so you have to read the official doc very clearly.