Experiment Report

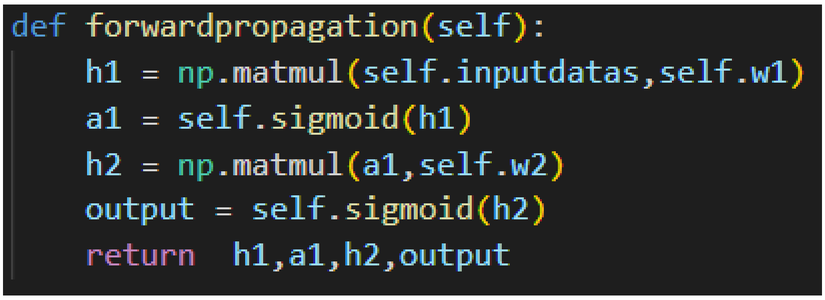
1. Introduction

We will do the backpropagation and forward propagation by Numpy (Python lib) in this lab. Before starting to do this lab, I go to understand the derivative of loss function and chain rule which are used to do backpropagation. And then I use a class to package all the function we need, and then build two models which are Linear model and Xor model. Finally, we can get two high accuracy models after we train those.

2. Experiment setups

A. Sigmoid functions

Because we are doing binary classification and sigmoid function can map the value into a range in [0,1], that can help us easily to classify the input data. I will use it after a linear layer.



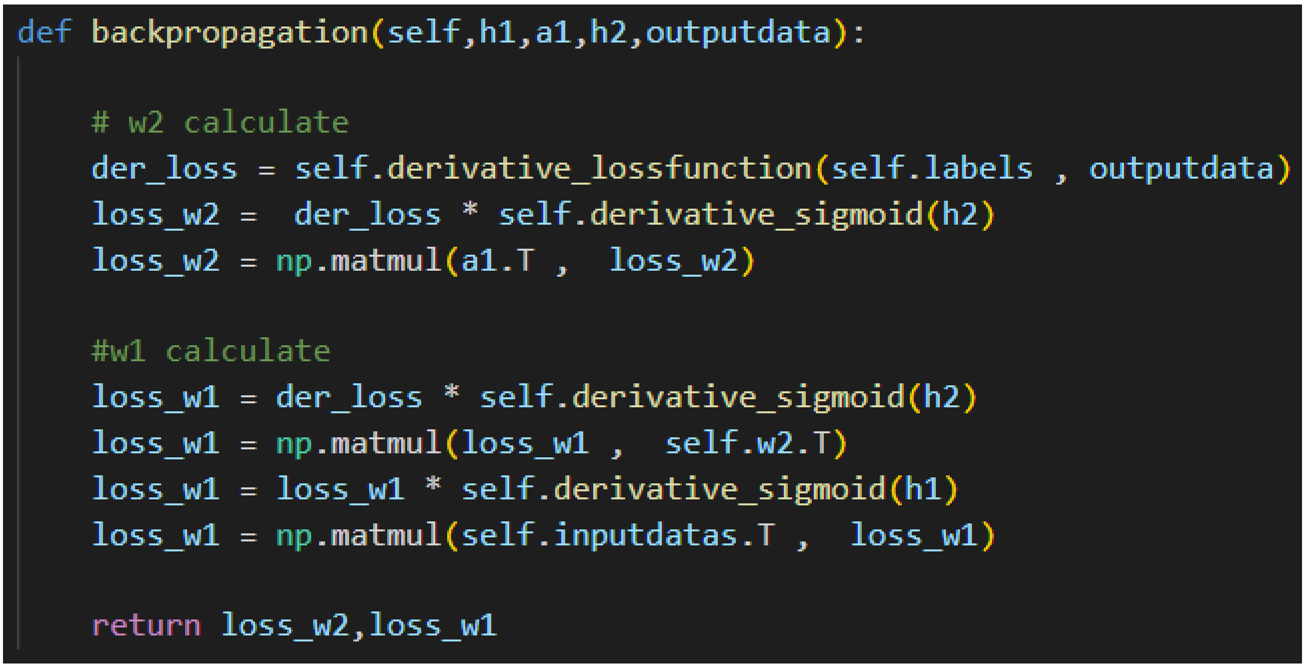
```
def forwardpropagation(self):
    h1 = np.matmul(self.inputdatas,self.w1)
    a1 = self.sigmoid(h1)
    h2 = np.matmul(a1,self.w2)
    output = self.sigmoid(h2)
    return  h1,a1,h2,output
```

B. Neural network

The above picture is my network architecture. I will use two hidden layers consist of a linear function and sigmoid function. The important thing is I save the output of every layer in order to do backpropagation. In this network, I only use two weights because the sigmoid function is one-to-one function, we don’t need to give it a weight.

C. Backpropagation

The below picture is the code of my backpropagation. I use chain rule to calculate what I need to multiply.

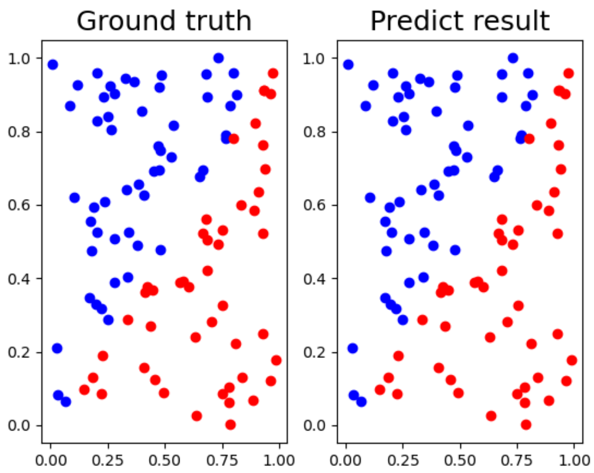


```
def backpropagation(self,h1,a1,h2,outputdata):

    # w2 calculate
    der_loss = self.derivative_lossfunction(self.labels , outputdata)
    loss_w2 =  der_loss * self.derivative_sigmoid(h2)
    loss_w2 = np.matmul(a1.T ,  loss_w2)

    #w1 calculate
    loss_w1 = der_loss * self.derivative_sigmoid(h2)
    loss_w1 = np.matmul(loss_w1 ,  self.w2.T)
    loss_w1 = loss_w1 * self.derivative_sigmoid(h1)
    loss_w1 = np.matmul(self.inputdatas.T ,  loss_w1)

    return loss_w2,loss_w1
```

3. Results of your testing

A. Screenshot and comparison figure

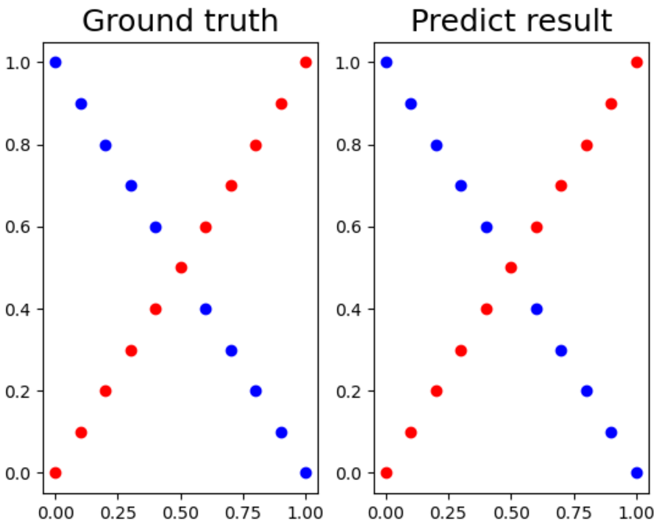
Linear :



Ground truth

Predict result

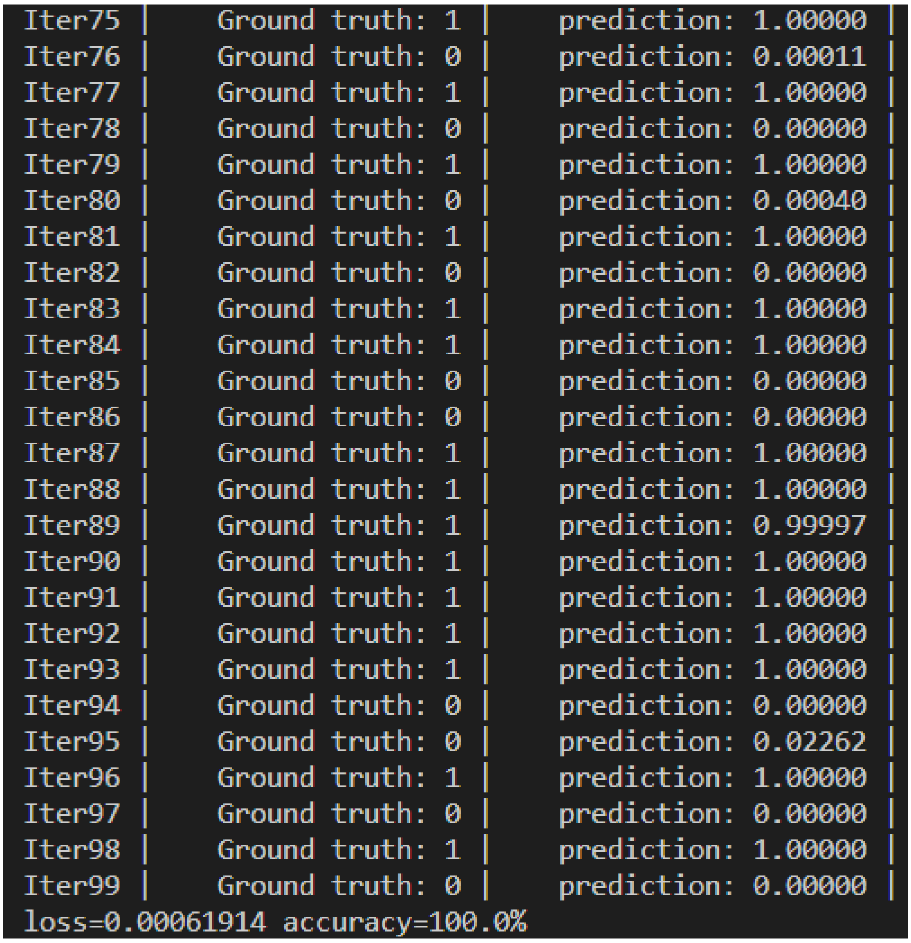
Xor:



B. Show the accuracy of your prediction

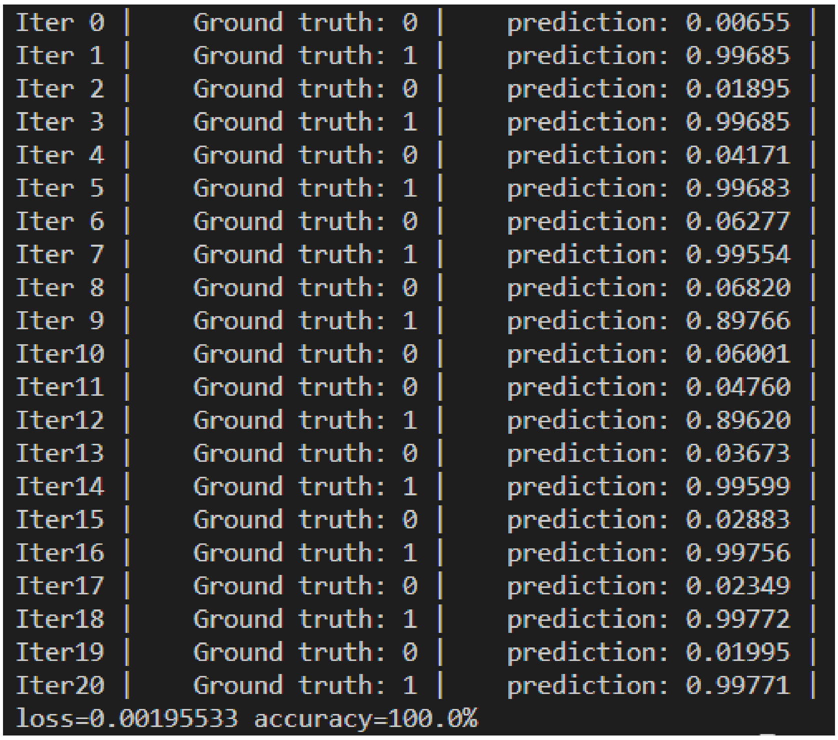
Linear:

Since the Iterator's results are too much that I can't screenshot all.



```
Iter75 |     Ground truth: 1 |     prediction: 1.00000 |
Iter76 |     Ground truth: 0 |     prediction: 0.00011 |
Iter77 |     Ground truth: 1 |     prediction: 1.00000 |
Iter78 |     Ground truth: 0 |     prediction: 0.00000 |
Iter79 |     Ground truth: 1 |     prediction: 1.00000 |
Iter80 |     Ground truth: 0 |     prediction: 0.00040 |
Iter81 |     Ground truth: 1 |     prediction: 1.00000 |
Iter82 |     Ground truth: 0 |     prediction: 0.00000 |
Iter83 |     Ground truth: 1 |     prediction: 1.00000 |
Iter84 |     Ground truth: 1 |     prediction: 1.00000 |
Iter85 |     Ground truth: 0 |     prediction: 0.00000 |
Iter86 |     Ground truth: 0 |     prediction: 0.00000 |
Iter87 |     Ground truth: 1 |     prediction: 1.00000 |
Iter88 |     Ground truth: 1 |     prediction: 1.00000 |
Iter89 |     Ground truth: 1 |     prediction: 0.99997 |
Iter90 |     Ground truth: 1 |     prediction: 1.00000 |
Iter91 |     Ground truth: 1 |     prediction: 1.00000 |
Iter92 |     Ground truth: 1 |     prediction: 1.00000 |
Iter93 |     Ground truth: 1 |     prediction: 1.00000 |
Iter94 |     Ground truth: 0 |     prediction: 0.00000 |
Iter95 |     Ground truth: 0 |     prediction: 0.02262 |
Iter96 |     Ground truth: 1 |     prediction: 1.00000 |
Iter97 |     Ground truth: 0 |     prediction: 0.00000 |
Iter98 |     Ground truth: 1 |     prediction: 1.00000 |
Iter99 |     Ground truth: 0 |     prediction: 0.00000 |
loss=0.00061914 accuracy=100.0%
```

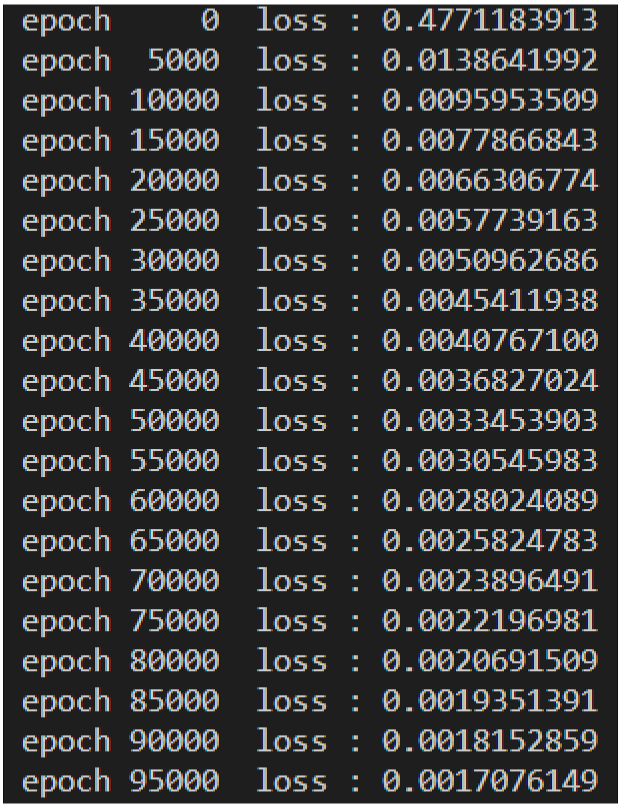
Xor:



```
Iter 0 |     Ground truth: 0 |     prediction: 0.00655 |
Iter 1 |     Ground truth: 1 |     prediction: 0.99685 |
Iter 2 |     Ground truth: 0 |     prediction: 0.01895 |
Iter 3 |     Ground truth: 1 |     prediction: 0.99685 |
Iter 4 |     Ground truth: 0 |     prediction: 0.04171 |
Iter 5 |     Ground truth: 1 |     prediction: 0.99683 |
Iter 6 |     Ground truth: 0 |     prediction: 0.06277 |
Iter 7 |     Ground truth: 1 |     prediction: 0.99554 |
Iter 8 |     Ground truth: 0 |     prediction: 0.06820 |
Iter 9 |     Ground truth: 1 |     prediction: 0.89766 |
Iter10 |     Ground truth: 0 |     prediction: 0.06001 |
Iter11 |     Ground truth: 0 |     prediction: 0.04760 |
Iter12 |     Ground truth: 1 |     prediction: 0.89620 |
Iter13 |     Ground truth: 0 |     prediction: 0.03673 |
Iter14 |     Ground truth: 1 |     prediction: 0.99599 |
Iter15 |     Ground truth: 0 |     prediction: 0.02883 |
Iter16 |     Ground truth: 1 |     prediction: 0.99756 |
Iter17 |     Ground truth: 0 |     prediction: 0.02349 |
Iter18 |     Ground truth: 1 |     prediction: 0.99772 |
Iter19 |     Ground truth: 0 |     prediction: 0.01995 |
Iter20 |     Ground truth: 1 |     prediction: 0.99771 |
loss=0.00195533 accuracy=100.0%
```

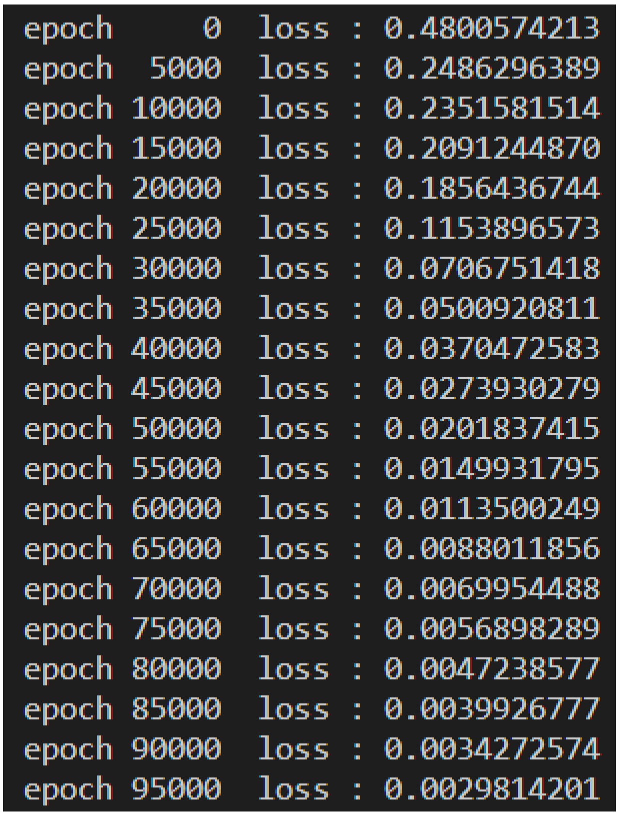
C. Learning curve (loss, epoch curve)

Linear:



```
epoch     0  loss : 0.4771183913
epoch  5000  loss : 0.0138641992
epoch 10000  loss : 0.0095953509
epoch 15000  loss : 0.0077866843
epoch 20000  loss : 0.0066306774
epoch 25000  loss : 0.0057739163
epoch 30000  loss : 0.0050962686
epoch 35000  loss : 0.0045411938
epoch 40000  loss : 0.0040767100
epoch 45000  loss : 0.0036827024
epoch 50000  loss : 0.0033453903
epoch 55000  loss : 0.0030545983
epoch 60000  loss : 0.0028024089
epoch 65000  loss : 0.0025824783
epoch 70000  loss : 0.0023896491
epoch 75000  loss : 0.0022196981
epoch 80000  loss : 0.0020691509
epoch 85000  loss : 0.0019351391
epoch 90000  loss : 0.0018152859
epoch 95000  loss : 0.0017076149
```

Xor:



```
epoch     0  loss : 0.4800574213
epoch  5000  loss : 0.2486296389
epoch 10000  loss : 0.2351581514
epoch 15000  loss : 0.2091244870
epoch 20000  loss : 0.1856436744
epoch 25000  loss : 0.1153896573
epoch 30000  loss : 0.0706751418
epoch 35000  loss : 0.0500920811
epoch 40000  loss : 0.0370472583
epoch 45000  loss : 0.0273930279
epoch 50000  loss : 0.0201837415
epoch 55000  loss : 0.0149931795
epoch 60000  loss : 0.0113500249
epoch 65000  loss : 0.0088011856
epoch 70000  loss : 0.0069954488
epoch 75000  loss : 0.0056898289
epoch 80000  loss : 0.0047238577
epoch 85000  loss : 0.0039926777
epoch 90000  loss : 0.0034272574
epoch 95000  loss : 0.0029814201
```

D. anything you want to present

If you use more hidden layers, you should be aware of the multiply the weights.

Because we choose 2x10, 10x10, 10x1. The middle layers will be aware of multiplying the weights, you can't transpose the wrong weights.

4. Discussion

A. Try different learning rates

The loss will decrease faster if I use higher learning rates.

B. Try different numbers of hidden units

At the beginning, Loss is more higher if I use the less hidden units. If we want to get the same loss, the more hidden units use more time to get the same loss.

C. Try without activation functions

I try to remove the first hidden layer's sigmoid function and keep second layer's sigmoid function because we need to output the value in range 0 to 1. The result show that the Linear can learn it and get 100% accuracy, but Xor can't. The loss in Xor model didn't decrease well, so that the accuracy just 52%.

D. Anything you want to share

I totally understand the backpropagation in this Lab01. I just can say this lab is so good!!