Deep Learning Practice Lab 1

0856056 Yu-Wun Tseng

April 2020

1 Introduction

The goal of the lab is to implement a simple neural network with forward pass and backpropagation , which has two layers. I implement it with two class, HiddenLayer and NeuralNetwork. In HiddenLayer, I implement the forward and backward pass of a layer of neurons. In NeuralNetwork, the full forward and backward pass are implemented. Finally, test the network with linear data and xor data.

2 Experiment Setups

2.1 Sigmoid Functions

The activation function of this implement is sigmoid function which is named by its S-shapep curve shown on Figure 1. The equation of sigmoid function is below:

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

In the backpropagation, the derivative is used for calculating the gradient. The derivative of the sigmoid function is below:

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

2.2 Neural Network

Neural networks is an algorithm inspired by the neurons in our brain. It is designed to recognize patterns in complex data. A neural network simply consists of neurons which are connected. Each neuron holds a number and each connection holds a weight.

To calculate the output of each layer, we multiply the input of each neuron by weights and plus the bias. Then put the sum of the products into sigmoid function to generate the output as below:

$$z^{(L)} = w^{(L)} \times a + b$$

$$a^{(L)} = \sigma(z^{(L)})$$

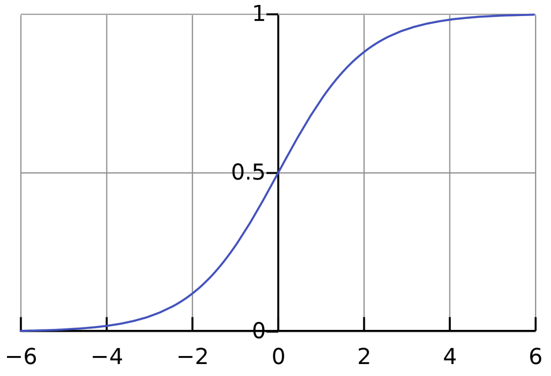


Figure 1: Sigmoid function

The a the inputs of each layers, w and b are the weights and the bias of the layer, L is the layer number, and the $a^{(L)}$ is the output of the layer.

After the output generated, the network passed it as input to next layer. The operation is called forward pass due to the direction of the data moving. After the forward pass is completed, the weights of each layer need to be modified to produce a more accurate prediction. This part is backpropagation.

2.3 Backpropagation

Backpropagation is used to calculate the gradients, starting from the output layer and propagating backwards, and updating weights and biases for each layer. The idea is that we nudge the weights and biases to get the desired output and minimize the cost function. The cost function is defined below:

$$C = (a^{(L)} - y)^2$$

Then use partial derivatives and chain rule to calculate the relationship between the neural network components and the cost function from last layer to first layer. When we know what affects it, we can effectively change the relevant weights and biases to minimize the cost function.

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}}$$

We calculate the gradients of each layer as above. Since the network contains not only the output layer, the gradients need to be propagated backwards.

$$\frac{\partial C}{\partial w^{(2)}} = \frac{\partial C}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial w^{(2)}}$$

$$\frac{\partial C}{\partial w^{(1)}} = \frac{\partial C}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial w^{(1)}}$$

As the equation shown, the part of $\frac{\partial C}{\partial w^{(1)}}$ is reused from $\frac{\partial C}{\partial w^{(2)}}$. We don't want to repeat the computation, thus, we pass the gradients to the upper layer as below.

$$dout^{(2)} = \frac{\partial C}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial a^{(1)}}$$

$$\frac{\partial C}{\partial w^{(1)}} = dout^{(2)} \frac{\partial a^{(1)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial w^{(1)}}$$

In the end, we subtract the weights from the weights multiply by the learning rate to update the weights as below:

$$w^{(L)} = w^{(L)} - learning\ rate \times \frac{\partial C}{\partial w^{(L)}}$$

3 Results

3.1 Loss and accuracy

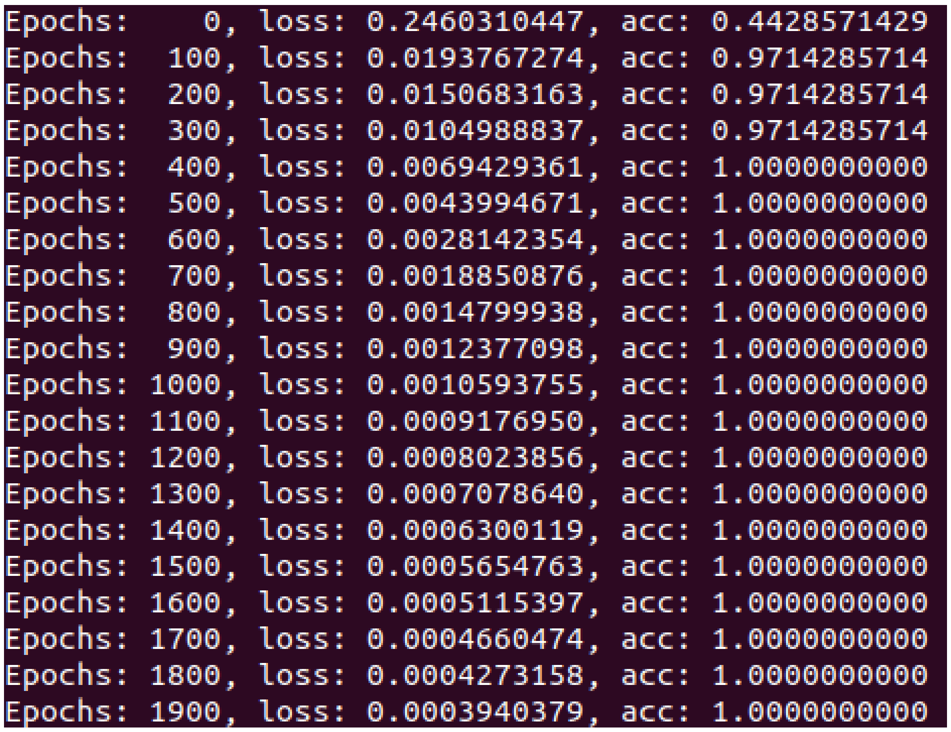
The training result of linear model is Figure 2 and the visual result is Figure 3. And the training result of xor model is Figure 4 and the visual result is Figure 5.

3.2 Comparison

The linear comparison result of prediction and ground truth is Figure 6. And the xor comparison result of prediction and ground truth is Figure 7.

3.3 Prediction

The prediction of linear model is Figure 8. And the prediction of xor model is Figure 9.



```
Epochs:    0, loss: 0.2460310447, acc: 0.4428571429
Epochs:  100, loss: 0.0193767274, acc: 0.9714285714
Epochs:  200, loss: 0.0150683163, acc: 0.9714285714
Epochs:  300, loss: 0.0104988837, acc: 0.9714285714
Epochs:  400, loss: 0.0069429361, acc: 1.0000000000
Epochs:  500, loss: 0.0043994671, acc: 1.0000000000
Epochs:  600, loss: 0.0028142354, acc: 1.0000000000
Epochs:  700, loss: 0.0018850876, acc: 1.0000000000
Epochs:  800, loss: 0.0014799938, acc: 1.0000000000
Epochs:  900, loss: 0.0012377098, acc: 1.0000000000
Epochs: 1000, loss: 0.0010593755, acc: 1.0000000000
Epochs: 1100, loss: 0.0009176950, acc: 1.0000000000
Epochs: 1200, loss: 0.0008023856, acc: 1.0000000000
Epochs: 1300, loss: 0.0007078640, acc: 1.0000000000
Epochs: 1400, loss: 0.0006300119, acc: 1.0000000000
Epochs: 1500, loss: 0.0005654763, acc: 1.0000000000
Epochs: 1600, loss: 0.0005115397, acc: 1.0000000000
Epochs: 1700, loss: 0.0004660474, acc: 1.0000000000
Epochs: 1800, loss: 0.0004273158, acc: 1.0000000000
Epochs: 1900, loss: 0.0003940379, acc: 1.0000000000
```

Figure 2: Linear loss and accuracy

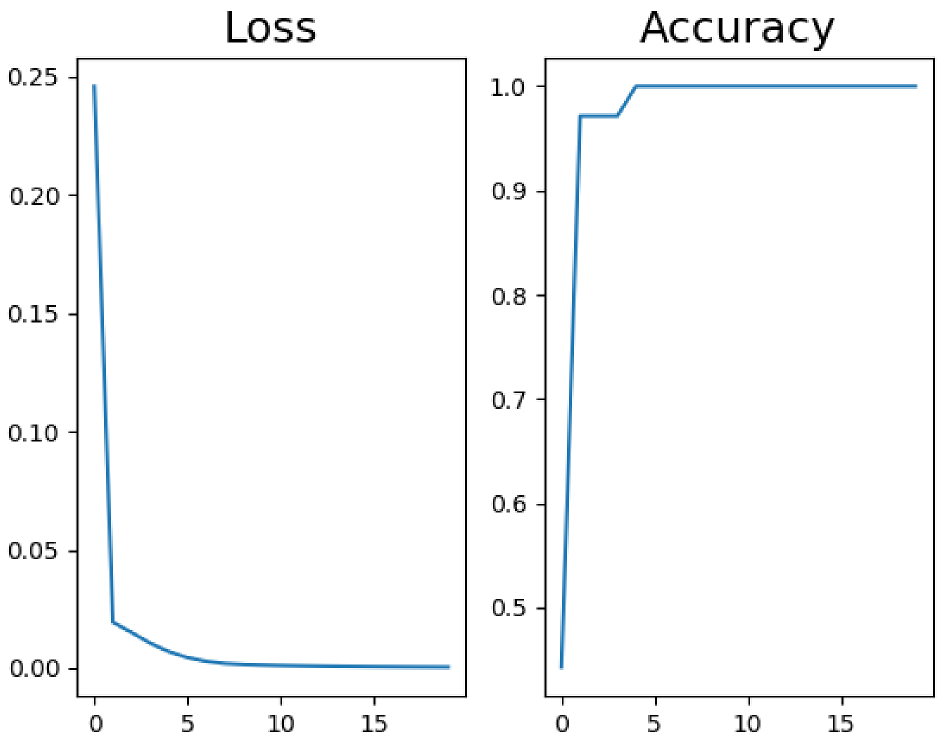
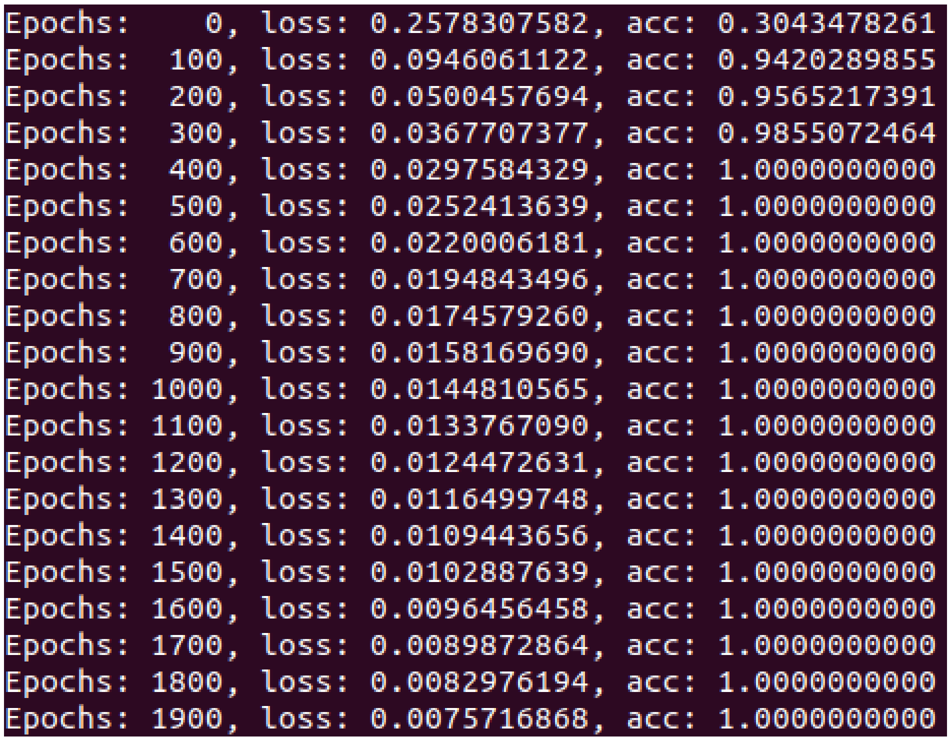


Figure 3: Linear loss and accuracy



```
Epochs:    0, loss: 0.2578307582, acc: 0.3043478261
Epochs:  100, loss: 0.0946061122, acc: 0.9420289855
Epochs:  200, loss: 0.0500457694, acc: 0.9565217391
Epochs:  300, loss: 0.0367707377, acc: 0.9855072464
Epochs:  400, loss: 0.0297584329, acc: 1.0000000000
Epochs:  500, loss: 0.0252413639, acc: 1.0000000000
Epochs:  600, loss: 0.0220006181, acc: 1.0000000000
Epochs:  700, loss: 0.0194843496, acc: 1.0000000000
Epochs:  800, loss: 0.0174579260, acc: 1.0000000000
Epochs:  900, loss: 0.0158169690, acc: 1.0000000000
Epochs: 1000, loss: 0.0144810565, acc: 1.0000000000
Epochs: 1100, loss: 0.0133767090, acc: 1.0000000000
Epochs: 1200, loss: 0.0124472631, acc: 1.0000000000
Epochs: 1300, loss: 0.0116499748, acc: 1.0000000000
Epochs: 1400, loss: 0.0109443656, acc: 1.0000000000
Epochs: 1500, loss: 0.0102887639, acc: 1.0000000000
Epochs: 1600, loss: 0.0096456458, acc: 1.0000000000
Epochs: 1700, loss: 0.0089872864, acc: 1.0000000000
Epochs: 1800, loss: 0.0082976194, acc: 1.0000000000
Epochs: 1900, loss: 0.0075716868, acc: 1.0000000000
```

Figure 4: XOR loss and accuracy

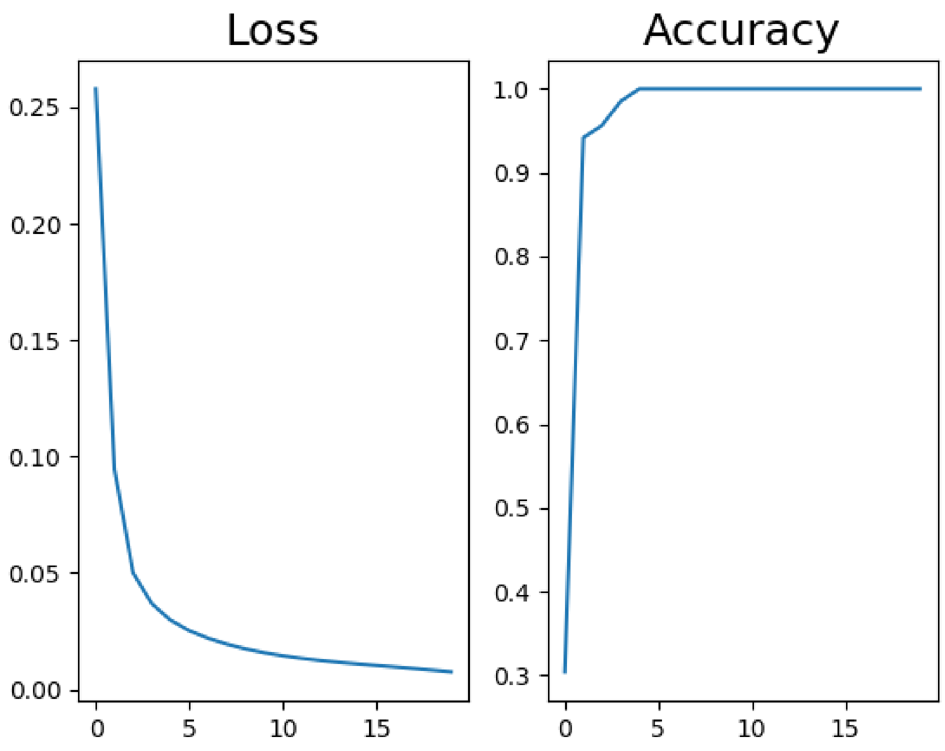


Figure 5: XOR loss and accuracy

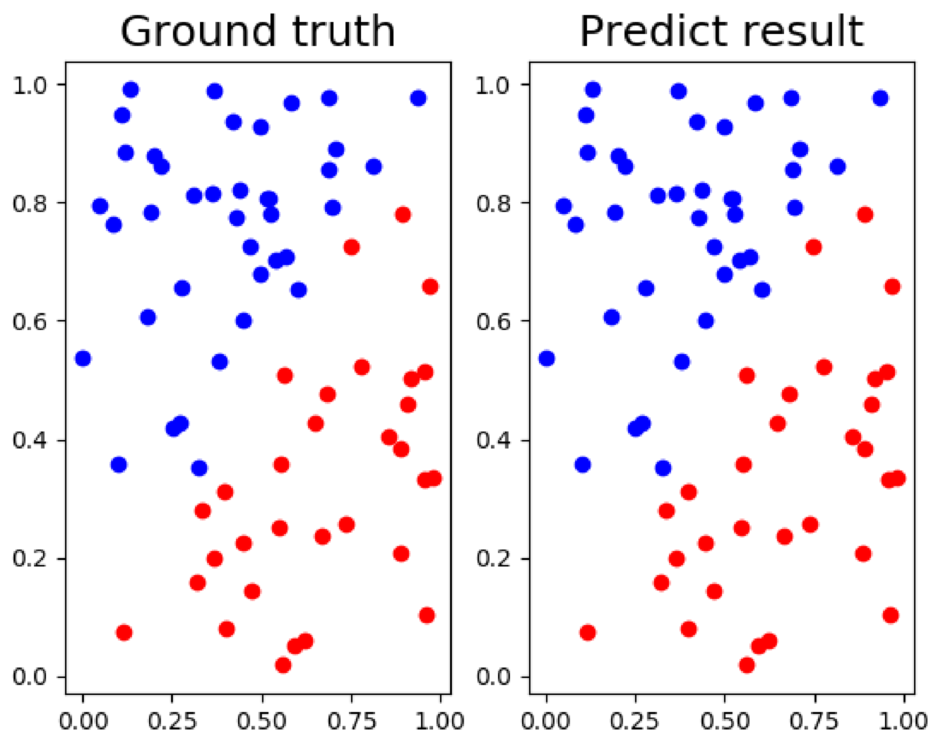


Figure 6: Linear comparison

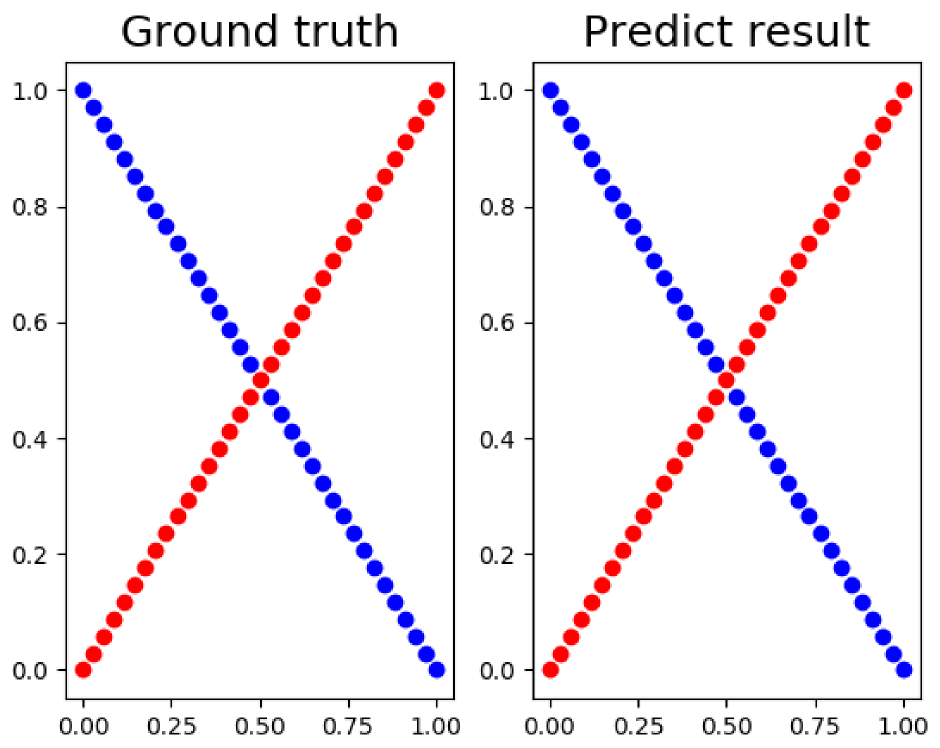
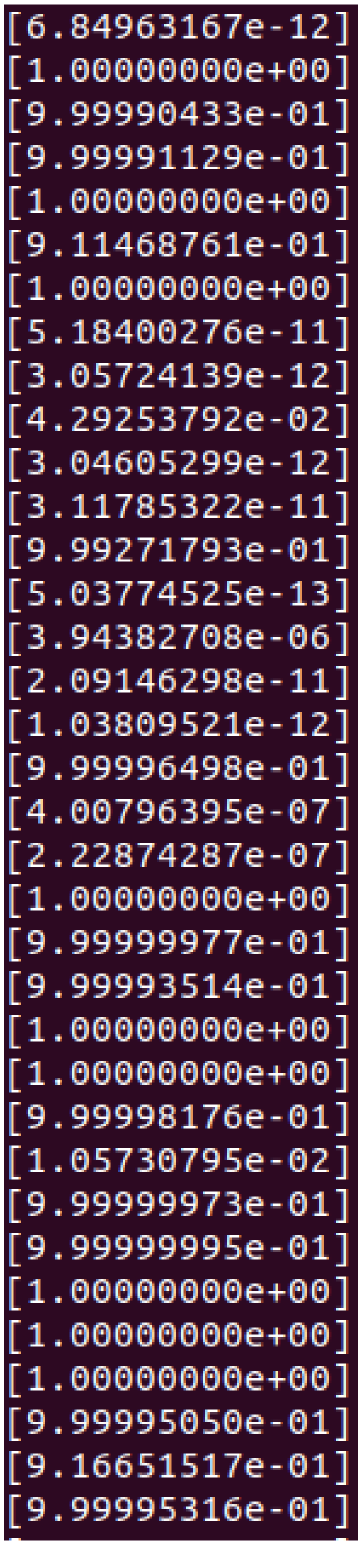
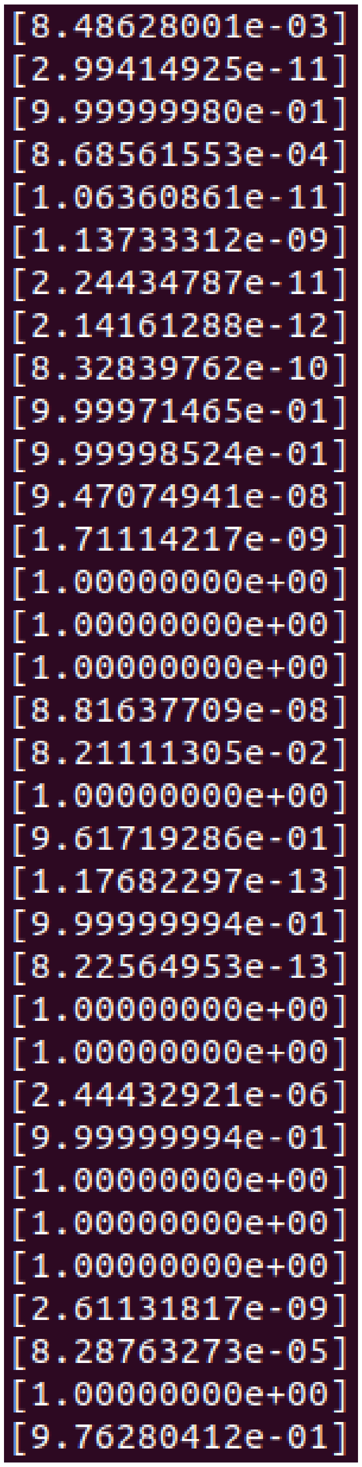


Figure 7: XOR comparison



```
[6.84963167e-12]
[1.00000000e+00]
[9.99990433e-01]
[9.99991129e-01]
[1.00000000e+00]
[9.11468761e-01]
[1.00000000e+00]
[5.18400276e-11]
[3.05724139e-12]
[4.29253792e-02]
[3.04605299e-12]
[3.11785322e-11]
[9.99271793e-01]
[5.03774525e-13]
[3.94382708e-06]
[2.09146298e-11]
[1.03809521e-12]
[9.99996498e-01]
[4.00796395e-07]
[2.22874287e-07]
[1.00000000e+00]
[9.99999977e-01]
[9.99993514e-01]
[1.00000000e+00]
[1.00000000e+00]
[9.99998176e-01]
[1.05730795e-02]
[9.99999973e-01]
[9.99999995e-01]
[1.00000000e+00]
[1.00000000e+00]
[1.00000000e+00]
[9.99995050e-01]
[9.16651517e-01]
[9.99995316e-01]
```

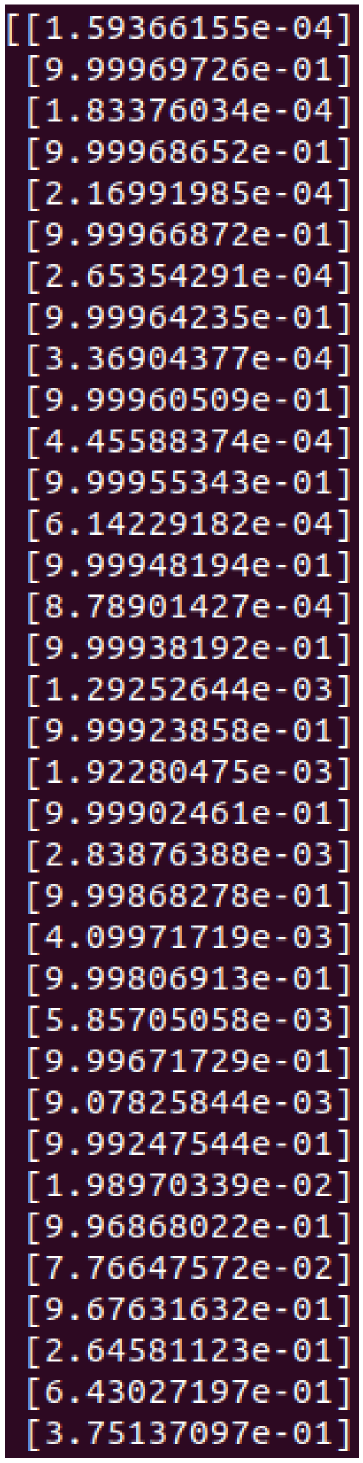
(a)



```
[8.48628001e-03]
[2.99414925e-11]
[9.99999980e-01]
[8.68561553e-04]
[1.06360861e-11]
[1.13733312e-09]
[2.24434787e-11]
[2.14161288e-12]
[8.32839762e-10]
[9.99971465e-01]
[9.99998524e-01]
[9.47074941e-08]
[1.71114217e-09]
[1.00000000e+00]
[1.00000000e+00]
[1.00000000e+00]
[8.81637709e-08]
[8.21111305e-02]
[1.00000000e+00]
[9.61719286e-01]
[1.17682297e-13]
[9.99999994e-01]
[8.22564953e-13]
[1.00000000e+00]
[1.00000000e+00]
[2.44432921e-06]
[9.99999994e-01]
[1.00000000e+00]
[1.00000000e+00]
[1.00000000e+00]
[2.61131817e-09]
[8.28763273e-05]
[1.00000000e+00]
[9.76280412e-01]
```

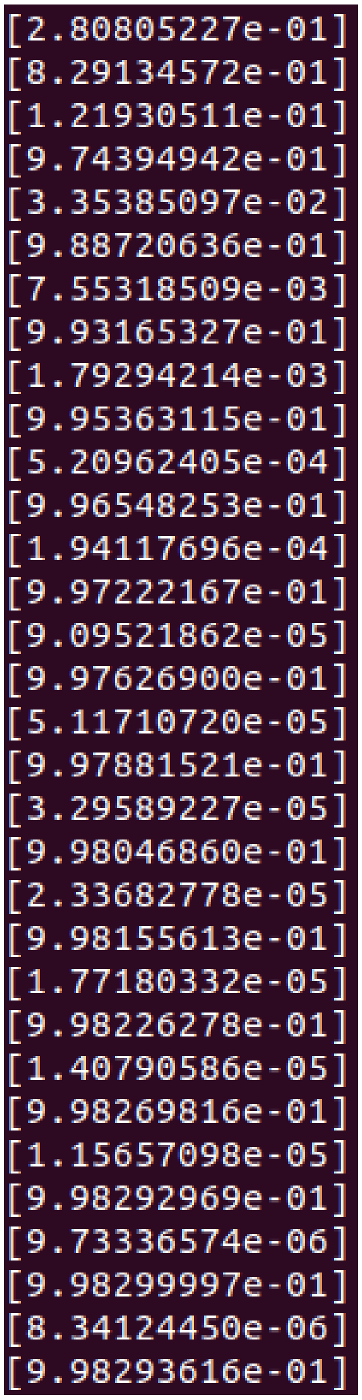
(b)

Figure 8: Linear prediction



```
[[1.59366155e-04]
 [9.99969726e-01]
 [1.83376034e-04]
 [9.99968652e-01]
 [2.16991985e-04]
 [9.99966872e-01]
 [2.65354291e-04]
 [9.99964235e-01]
 [3.36904377e-04]
 [9.99960509e-01]
 [4.45588374e-04]
 [9.99955343e-01]
 [6.14229182e-04]
 [9.99948194e-01]
 [8.78901427e-04]
 [9.99938192e-01]
 [1.29252644e-03]
 [9.99923858e-01]
 [1.92280475e-03]
 [9.99902461e-01]
 [2.83876388e-03]
 [9.99868278e-01]
 [4.09971719e-03]
 [9.99806913e-01]
 [5.85705058e-03]
 [9.99671729e-01]
 [9.07825844e-03]
 [9.99247544e-01]
 [1.98970339e-02]
 [9.96868022e-01]
 [7.76647572e-02]
 [9.67631632e-01]
 [2.64581123e-01]
 [6.43027197e-01]
 [3.75137097e-01]
```

(a)



```
[2.80805227e-01]
[8.29134572e-01]
[1.21930511e-01]
[9.74394942e-01]
[3.35385097e-02]
[9.88720636e-01]
[7.55318509e-03]
[9.93165327e-01]
[1.79294214e-03]
[9.95363115e-01]
[5.20962405e-04]
[9.96548253e-01]
[1.94117696e-04]
[9.97222167e-01]
[9.09521862e-05]
[9.97626900e-01]
[5.11710720e-05]
[9.97881521e-01]
[3.29589227e-05]
[9.98046860e-01]
[2.33682778e-05]
[9.98155613e-01]
[1.77180332e-05]
[9.98226278e-01]
[1.40790586e-05]
[9.98269816e-01]
[1.15657098e-05]
[9.98292969e-01]
[9.73336574e-06]
[9.98299997e-01]
[8.34124450e-06]
[9.98293616e-01]
```

(b)

Figure 9: XOR prediction

4 Discussion

At the beginning, I used a hidden size equal to 100 as the sample code, but no matter how I changed the number of steps and the learning rate, the model still can't learn well. The loss stop decreasing after few epochs. I searched the cause of this situation on Google and found that if the data is too simple, the model can't learn well with large hidden size. This is because the model needs to learn too many arguments, but the data can provide very little information. As a result, I changed the hidden size into 4 or 10. However, there were only a few chances that the accuracy reaches 100%.

After I discussed this problem with my classmate, we found the problem is not about the hidden size but the initialization weight. I used np.random.uniform function and set interval [0, 1]. It made the initial values close to 1 so the gradients were almost 0. Of course, the model couldn't learn properly. Therefore, I set the interval into [-1, 1] and the model can learn well with large hidden size.

References

[1] Neural Networks: Feedforward and Backpropagation Explained & Optimization.
`https://mlfromscratch.com/neural-networks-explained/#what-is-a-neural-network`

[2] Sigmoid function
`https://en.wikipedia.org/wiki/Sigmoid-function`