DLP HW1

多工所碩一 楊奕儒
ID：312553038

(1) Introduction

In this lab, I implemented a simple neural network with forward and backward pass using two hidden layers to classify input data.

(2) Experiment setups

(a) Sigmoid functions

```
def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))

def der_sigmoid(y):
    # input y must be the value which is the output from sigmoid function
    return y * (1 - y)
```

(b) Neural network

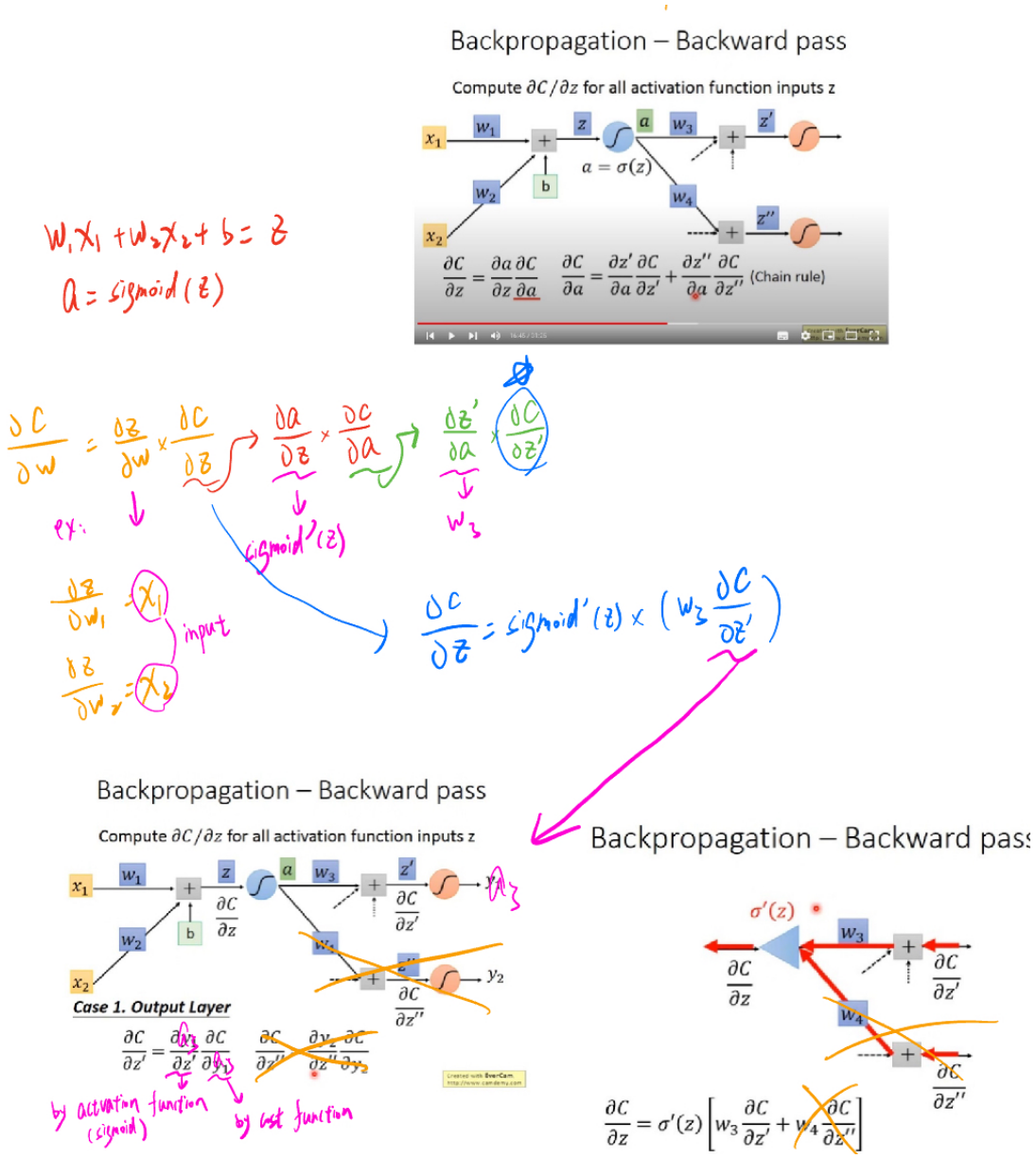
In my neural network, there are ten neurons in both two hidden layers , and the learning rate is set to 0.2.

Also, I chose cross entropy as loss function to calculate loss.

```
class NNwork:
    def __init__(self,neurons_per_layer = (10, 10)):
        self.lr = 0.2
        self.epoch = 20000
        self.EPS=1e-3

        self.w1 = np.random.rand(neurons_per_layer[0],2)
        self.w2 = np.random.rand(neurons_per_layer[1],neurons_per_layer[0])
        self.w3 = np.random.rand(1,neurons_per_layer[1])

    def loss_function(self, pred_y, label_y):
        '''
        choose cross entrophy as loss function
        '''
        loss=-(1/label_y.shape[0])*(label_y.T@np.log(pred_y+self.EPS).T+(1-labe
        return float(loss)


    def forward(self, inputs):
        '''
        x-> w1-> z1-> a1-> w2-> z2-> a2-> w3-> z3-> a3
        '''
```

(c) Backpropagation

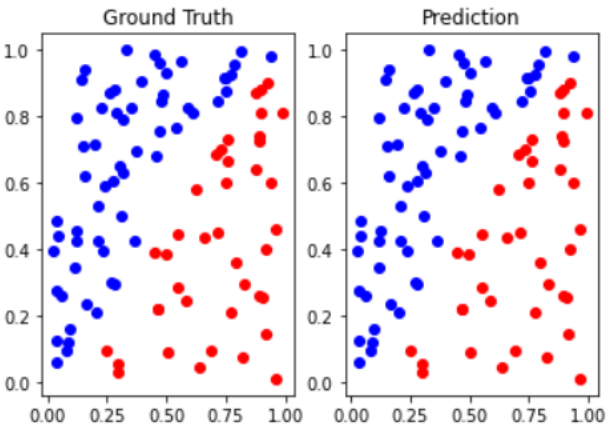
Based on these pictures below, I calculated the gradient and update the weight between layers.



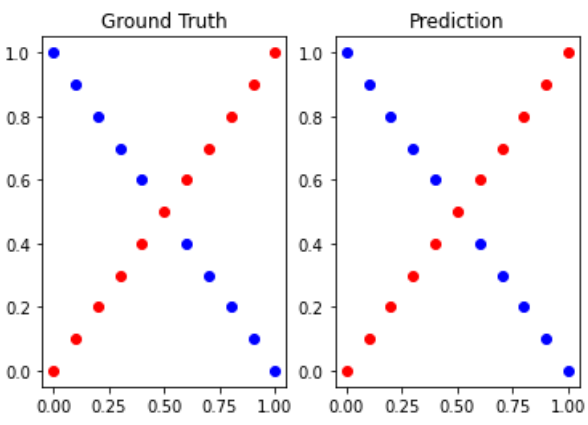
(3) Results of your testing

(a) Screenshot and comparison figure

Linear：



XOR：



(b) Show the accuracy of your prediction

Linear：

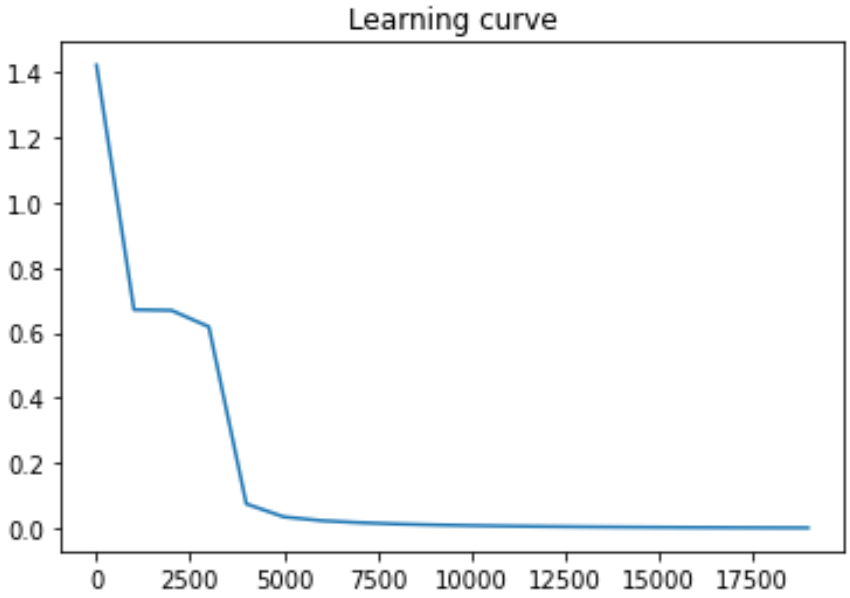
```
start testing:
[[9.99982642e-01 7.32882718e-03 9.99906312e-01 4.29273774e-06
  9.99979385e-01 9.99970647e-01 4.25659769e-02 9.96249019e-01
  9.99945372e-01 9.33236607e-03 9.99818769e-01 9.99980730e-01
  8.96820197e-01 5.99633887e-03 5.58938915e-05 9.99983104e-01
  9.99970313e-01 9.99962620e-01 2.40745805e-05 8.82220362e-06
  9.99946580e-01 4.89408638e-06 9.99980019e-01 1.30987035e-04
  9.99968810e-01 9.99978413e-01 9.99978802e-01 9.98696931e-01
  8.03852137e-06 9.76921287e-06 9.99982687e-01 9.99982544e-01
  9.99980989e-01 8.00202648e-06 9.98974733e-01 6.15200039e-06
  9.99889334e-01 9.99978289e-01 1.23680758e-04 9.99938459e-01
  9.99742948e-01 9.99980432e-01 5.10814696e-06 9.99932396e-01
  9.99976596e-01 9.99978643e-01 2.23021685e-05 9.99979364e-01
  8.91414605e-06 9.80341471e-01 4.48062997e-06 9.99982705e-01
  4.97810513e-06 9.99979944e-01 9.99982176e-01 4.22734544e-06
  9.99970048e-01 9.99982990e-01 4.01049872e-06 9.99982149e-01
  9.99748352e-01 9.99968982e-01 9.99983069e-01 4.18041098e-06
  9.99981429e-01 9.99972345e-01 9.99981812e-01 9.99866283e-01
  9.83843083e-01 1.31685587e-04 2.07563924e-02 4.64526304e-06
  1.54601241e-05 4.38311656e-06 9.99957043e-01 9.96172122e-01
  4.11800168e-06 1.29951166e-05 4.41213930e-06 9.99479532e-06
  9.99964360e-01 5.98679661e-06 9.99981960e-01 9.99981787e-01
  4.77637276e-06 9.99978359e-01 1.27244873e-04 9.99973544e-01
  9.99977665e-01 8.98977848e-06 3.54709504e-02 7.97638978e-03
  9.99982568e-01 1.43389127e-02 9.99979325e-01 9.99982142e-01
  9.99259739e-01 9.99982286e-01 7.91508533e-06 9.99981300e-01]]
Loss: 0.0020421567007200673, Accuracy: 100.00%
```

XOR：

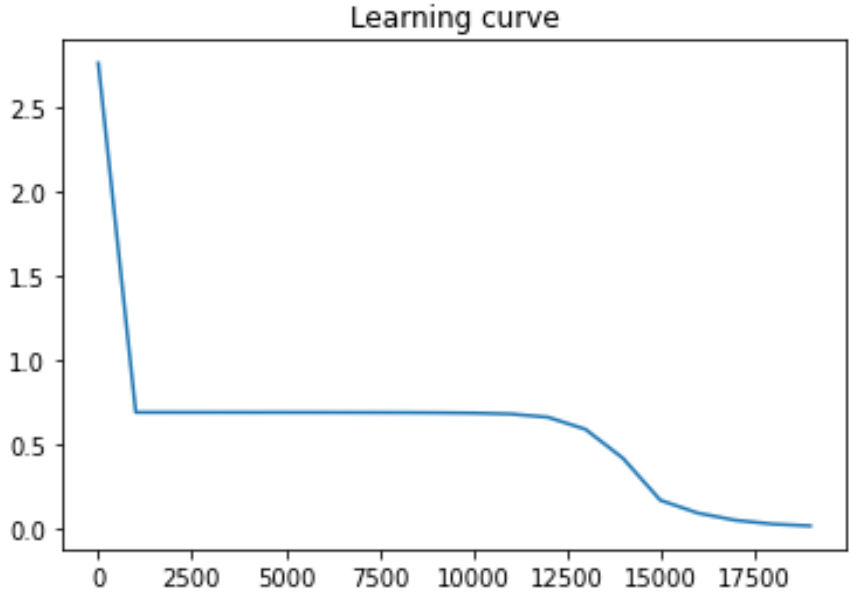
```
start testing:
[[1.17889858e-05 9.99999627e-01 1.43005961e-04 9.99999411e-01
  2.10241043e-03 9.99998396e-01 1.44637720e-02 9.99972555e-01
  3.23809053e-02 9.43208001e-01 2.99598451e-02 1.66029642e-02
  9.51078084e-01 7.47070327e-03 9.99930234e-01 3.33778806e-03
  9.99973800e-01 1.64056585e-03 9.99966633e-01 9.18905421e-04
  9.99950356e-01]]
Loss: 0.009425376282576095, Accuracy: 100.00%
```

(c) Learning curve (loss, epoch curve)

Linear：



Xor：



(d) Anything you want to present

Based on the output, we can see that both linear and XOR input data achieve 100% accuracy. However, in the XOR input case, the loss remains constant for a certain period of time before eventually converging.

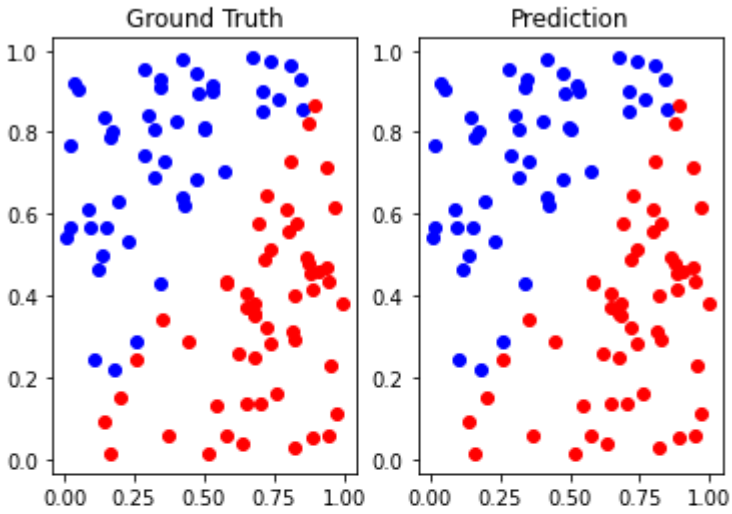
(3) Discussion

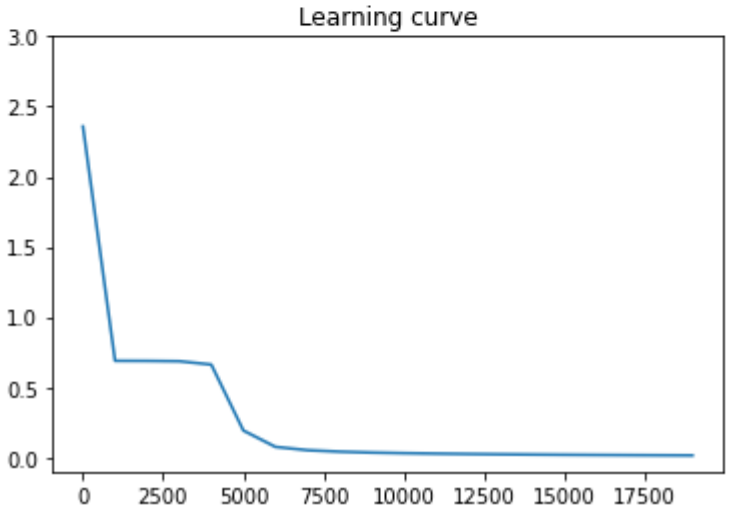
(a) Try different learning rates

After lowering the learning rate to 0.1, we observed that in the linear case, the loss still converges and the accuracy reaches 100%. However, in the XOR case, the loss does not converge. I suspect that the gradients are getting stuck in a local minimum, so we need to increase the learning rate in order for it to converge.

Linear：

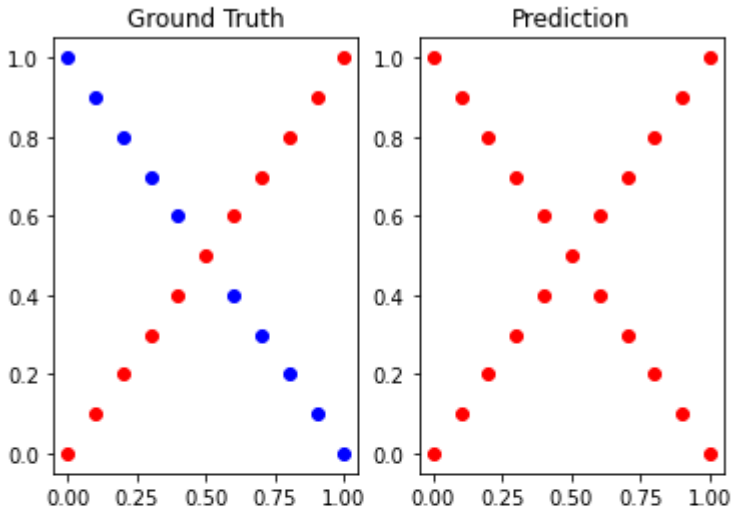
```
Loss: 0.014003304704918368, Accuracy: 100.00%
testing finished
```

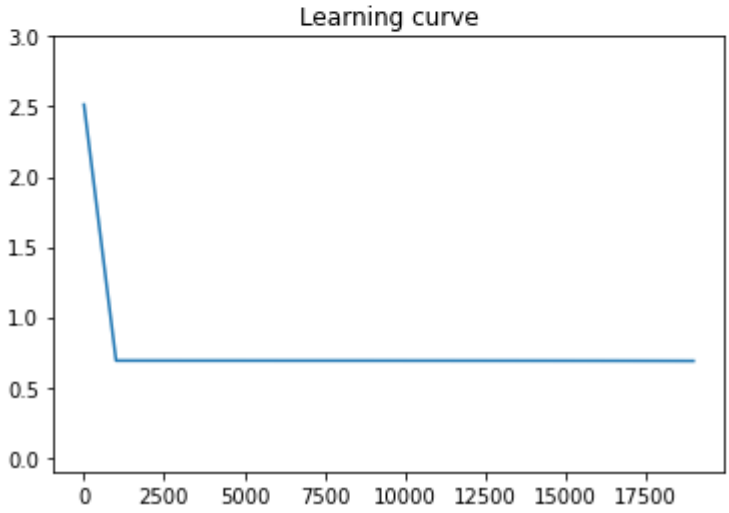




XOR：

```
Loss: 0.6869220195092655, Accuracy: 52.38%
testing finished
```

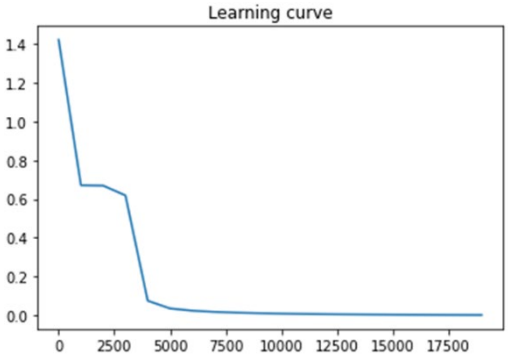




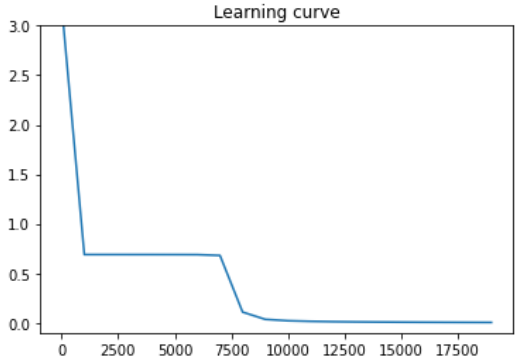
(b) Try different numbers of hidden units

Linear:

After doubling the number of neurons in each layer, we observed that although the loss can still converge, it requires more epochs to achieve convergence in linear case.



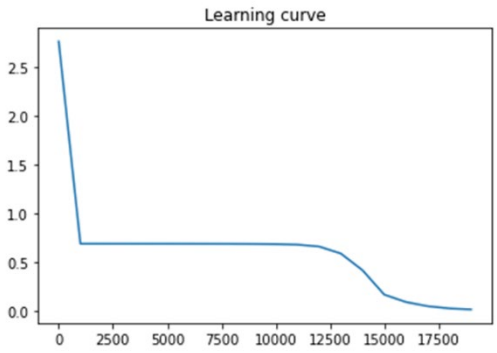
10 neurons in each layer



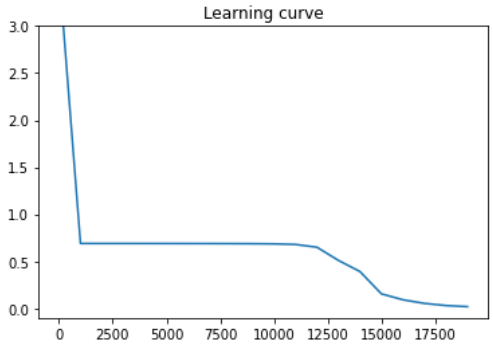
20 neurons in each layer

XOR：

After doubling the number of neurons in each layer, we observed that the number of epochs required to converge did not change significantly compared to the linear case.



10 neurons in each layer



20 neurons in each layer

(c) Try without activation functions

If I remove the activation functions from my neural network, the output may generate negative values. Consequently, when calculating the loss using cross entropy, the logarithm function cannot handle negative values, resulting in an incorrect loss calculation.

Reference：

https://www.youtube.com/watch?v=ibJpTrp5mcE&list=PLJV_el

3uVTsPy9oCRY30oBPNLCo89yu49&index=12&t=1193s