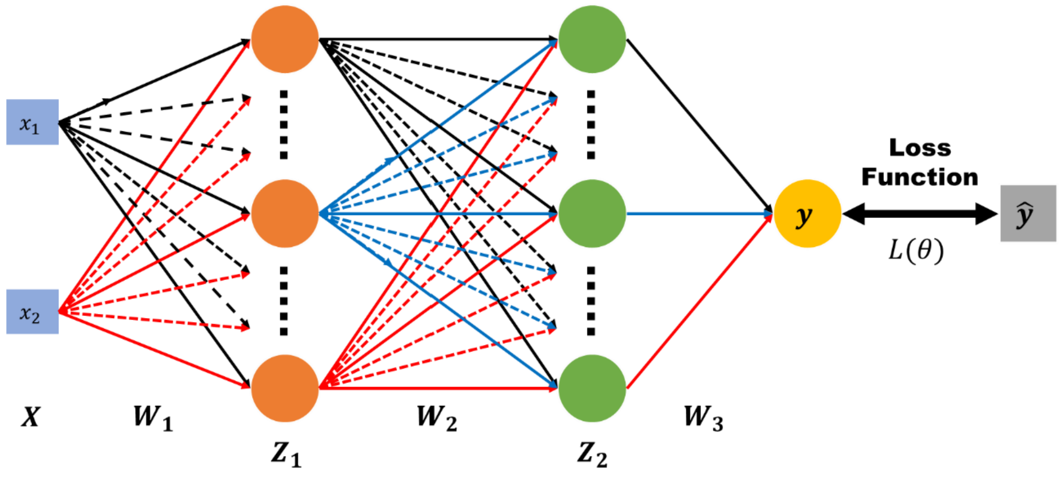
Deep Learning and Practice Lab 1

309554005 黃睿宇

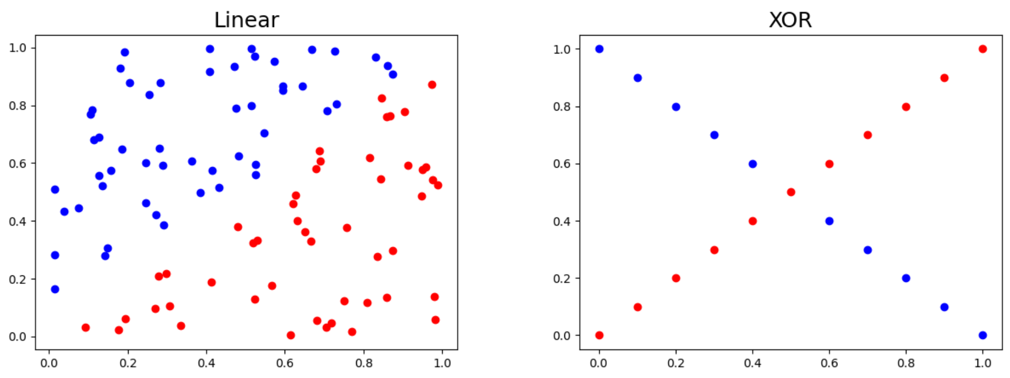
1 Introduction

In this lab, we implement simple neural networks with forwarding pass and backpropagation using two hidden layers.

Neural networks are composed of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or neuron, connects to another and has an associated weight. The input is fed forward to those hidden layers and their corresponding activation functions, and get the output in the final output layer.



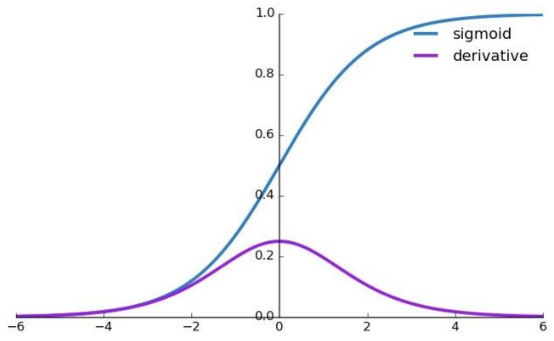
The input data including linear function and XOR function. Two kinds of data set are shown below.



2 Experiment Setups

2.1 Activation function: sigmoid function

We choose sigmoid function as the activation functions of the neural network. A sigmoid function is a mathematical function having a characteristic "S"-shaped curve. The first derivative of sigmoid function is bell shaped. The figure of sigmoid function and its first derivative is shown below.



The sigmoid function $\sigma(x)$ is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$, and its first derivative can be written as $\sigma'(x) = \sigma(x)(1 - \sigma(x))$.

2.2 Neural network

In my neural network, I use 4 neurons for each hidden layer, and the learning rate is set as 1.

In addition, I use mean square error (MSE) as my loss function. MSE of the predictor is computed as $\frac{\sum(y-\hat{y})^2}{n}$, and its first derivative is computed as $\frac{2(y-\hat{y})}{n}$.

2.3 Backpropagation

First, all weight parameters in the network are randomly initialized. The objective is to minimize cost *C* from loss function. Since the gradient $\frac{\partial C}{\partial w}$ is hard to compute, we use chain rules $\frac{\partial C}{\partial w} = \frac{\partial z}{\partial w}\frac{\partial C}{\partial z}$ to simplify.

In forward pass, $\frac{\partial z}{\partial w} = \frac{\partial x_i w}{\partial w} = x_i$ could be obtained easily. In backward pass, $\frac{\partial C}{\partial z} = \frac{\partial y}{\partial z}\frac{\partial C}{\partial y} = \frac{\partial \sigma(z)}{\partial z}\frac{\partial C}{\partial y} = \sigma'(z)\frac{\partial C}{\partial y}$. To obtain $\frac{\partial C}{\partial y}$, we consider two situation: 1) If *y* follows the output layer, then we know $\frac{\partial C}{\partial y} = \frac{\partial y\prime}{\partial y}\frac{\partial C}{\partial y\prime}$ easily, where *y'* is the output. 2) If *y* doesn't follow the output layer, then we go backward to find $\frac{\partial C}{\partial y} = \frac{\partial y\prime}{\partial y}\frac{\partial C}{\partial y\prime}$. We compute $\frac{\partial C}{\partial y'}$ from output layer, then we transfer to the sigmoid function. After that, we transfer the result to the previous layer and repeat the process.

Ultimately, we could use the gradient and choose proper learning rate to update the weights $w = w - \eta \frac{\partial L}{\partial w}$.

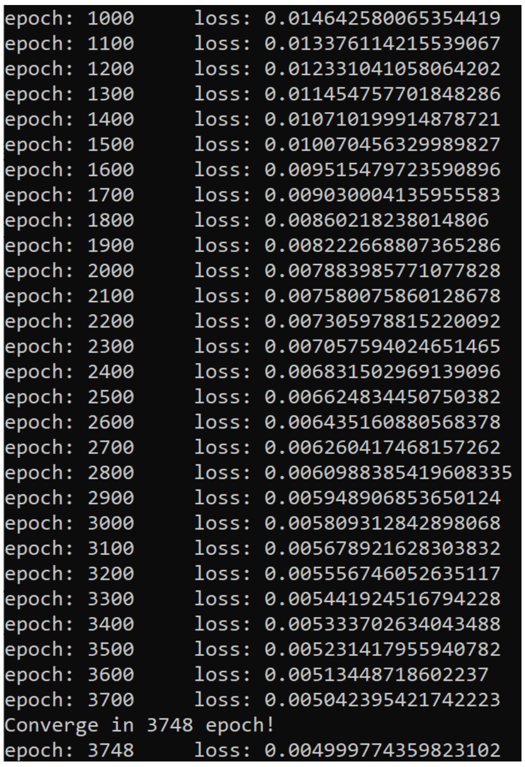
3 Results of Testing

The parameters for the two following results are:

1) 4 neurons / hidden layer
2) learning rate = 1
3) loss convergence: < 0.005

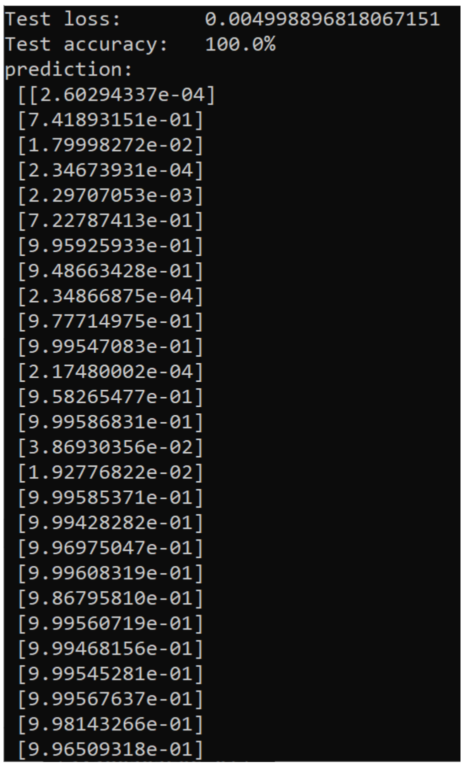
3.1 Linear

Training: converge in 3748 epoch

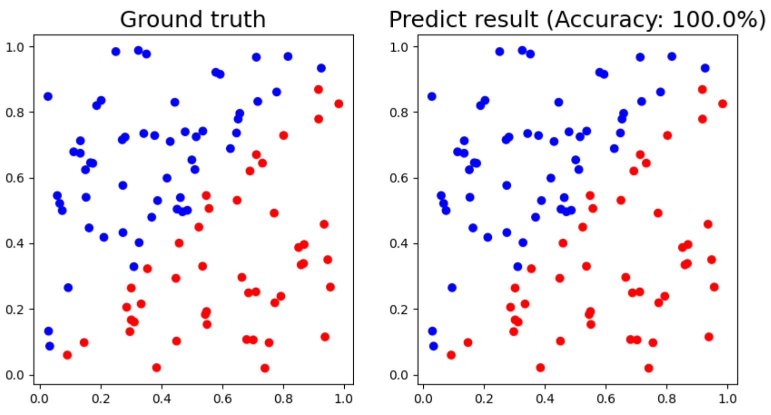


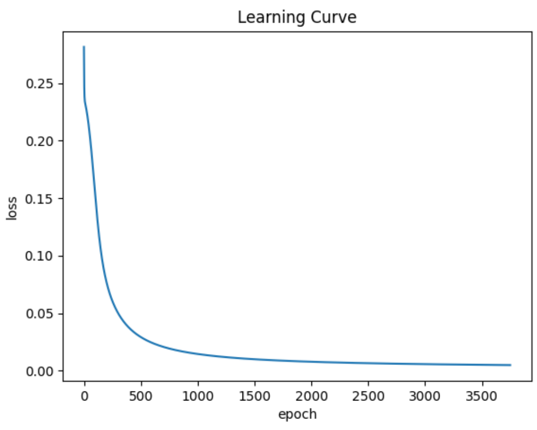
```
epoch: 1000     loss: 0.014642580065354419
epoch: 1100     loss: 0.013376114215539067
epoch: 1200     loss: 0.012331041058064202
epoch: 1300     loss: 0.011454757701848286
epoch: 1400     loss: 0.010710199914878721
epoch: 1500     loss: 0.010070456329989827
epoch: 1600     loss: 0.009515479723590896
epoch: 1700     loss: 0.009030004135955583
epoch: 1800     loss: 0.00860218238014806
epoch: 1900     loss: 0.008222668807365286
epoch: 2000     loss: 0.007883985771077828
epoch: 2100     loss: 0.007580075860128678
epoch: 2200     loss: 0.007305978815220092
epoch: 2300     loss: 0.007057594024651465
epoch: 2400     loss: 0.006831502969139096
epoch: 2500     loss: 0.006624834450750382
epoch: 2600     loss: 0.006435160880568378
epoch: 2700     loss: 0.006260417468157262
epoch: 2800     loss: 0.0060988385419608335
epoch: 2900     loss: 0.005948906853650124
epoch: 3000     loss: 0.005809312842898068
epoch: 3100     loss: 0.005678921628303832
epoch: 3200     loss: 0.005556746052635117
epoch: 3300     loss: 0.005441924516794228
epoch: 3400     loss: 0.005333702634043488
epoch: 3500     loss: 0.005231417955940782
epoch: 3600     loss: 0.00513448718602237
epoch: 3700     loss: 0.005042395421742223
Converge in 3748 epoch!
epoch: 3748     loss: 0.004999774359823102
```

Testing: loss = 0.00499, accuracy = 100.0%



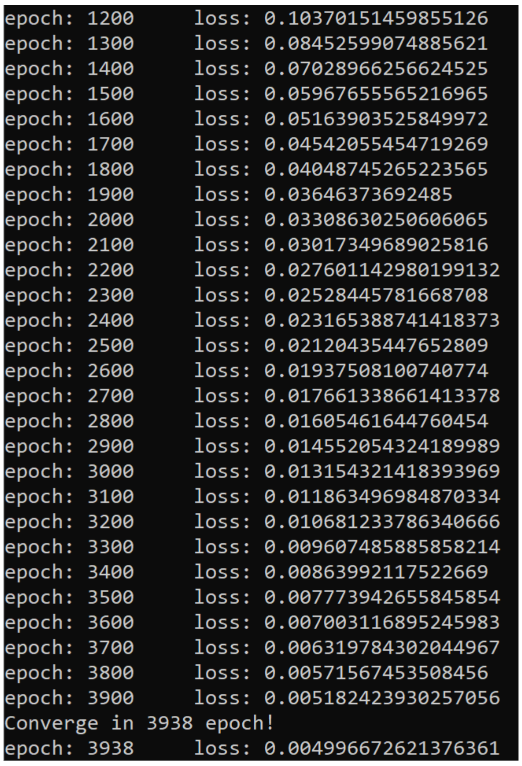
```
Test loss:        0.004998896818067151
Test accuracy:    100.0%
prediction:
 [[2.60294337e-04]
 [7.41893151e-01]
 [1.79998272e-02]
 [2.34673931e-04]
 [2.29707053e-03]
 [7.22787413e-01]
 [9.95925933e-01]
 [9.48663428e-01]
 [2.34866875e-04]
 [9.77714975e-01]
 [9.99547083e-01]
 [2.17480002e-04]
 [9.58265477e-01]
 [9.99586831e-01]
 [3.86930356e-02]
 [1.92776822e-02]
 [9.99585371e-01]
 [9.99428282e-01]
 [9.96975047e-01]
 [9.99608319e-01]
 [9.86795810e-01]
 [9.99560719e-01]
 [9.99468156e-01]
 [9.99545281e-01]
 [9.99567637e-01]
 [9.98143266e-01]
 [9.96509318e-01]
```





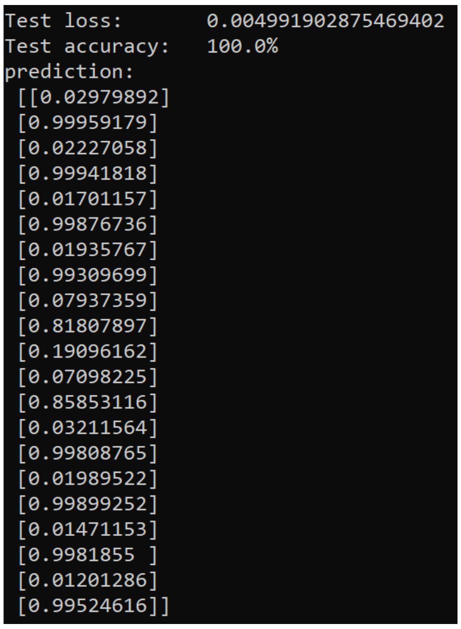
3.2 XOR

Training: converge in 3938 epoch

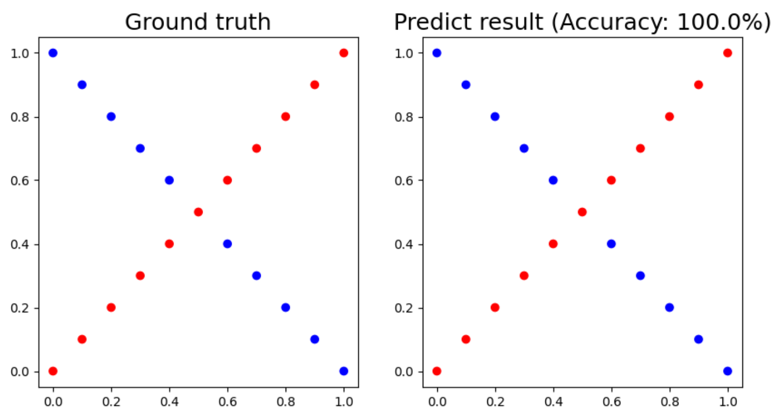


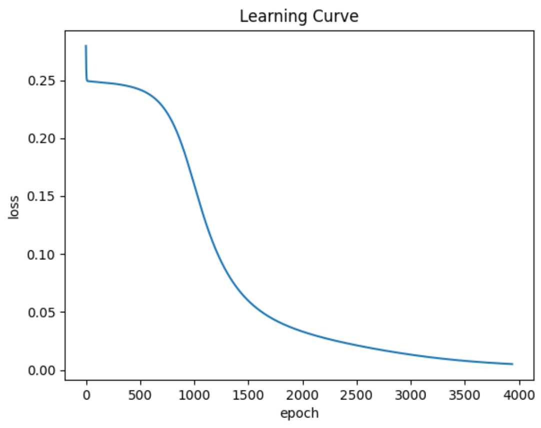
```
epoch: 1200     loss: 0.10370151459855126
epoch: 1300     loss: 0.08452599074885621
epoch: 1400     loss: 0.07028966256624525
epoch: 1500     loss: 0.05967655565216965
epoch: 1600     loss: 0.05163903525849972
epoch: 1700     loss: 0.04542055454719269
epoch: 1800     loss: 0.04048745265223565
epoch: 1900     loss: 0.03646373692485
epoch: 2000     loss: 0.03308630250606065
epoch: 2100     loss: 0.03017349689025816
epoch: 2200     loss: 0.027601142980199132
epoch: 2300     loss: 0.02528445781668708
epoch: 2400     loss: 0.023165388741418373
epoch: 2500     loss: 0.02120435447652809
epoch: 2600     loss: 0.01937508100740774
epoch: 2700     loss: 0.017661338661413378
epoch: 2800     loss: 0.01605461644760454
epoch: 2900     loss: 0.014552054324189989
epoch: 3000     loss: 0.013154321418393969
epoch: 3100     loss: 0.011863496984870334
epoch: 3200     loss: 0.010681233786340666
epoch: 3300     loss: 0.009607485885858214
epoch: 3400     loss: 0.00863992117522669
epoch: 3500     loss: 0.007773942655845854
epoch: 3600     loss: 0.007003116895245983
epoch: 3700     loss: 0.006319784302044967
epoch: 3800     loss: 0.00571567453508456
epoch: 3900     loss: 0.005182423930257056
Converge in 3938 epoch!
epoch: 3938     loss: 0.004996672621376361
```

Testing: loss = 0.00499, accuracy = 100.0%



```
Test loss:       0.004991902875469402
Test accuracy:   100.0%
prediction:
 [[0.02979892]
 [0.99959179]
 [0.02227058]
 [0.99941818]
 [0.01701157]
 [0.99876736]
 [0.01935767]
 [0.99309699]
 [0.07937359]
 [0.81807897]
 [0.19096162]
 [0.07098225]
 [0.85853116]
 [0.03211564]
 [0.99808765]
 [0.01989522]
 [0.99899252]
 [0.01471153]
 [0.9981855 ]
 [0.01201286]
 [0.99524616]]
```





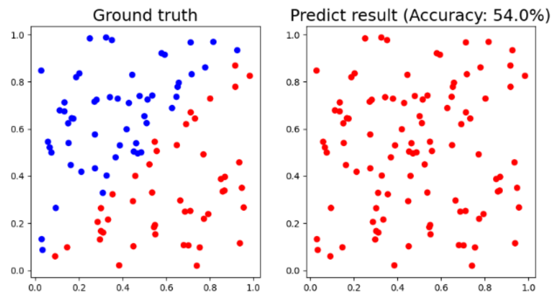
4 Discussion

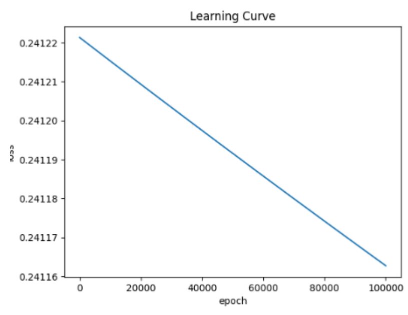
4.1 Different learning rates

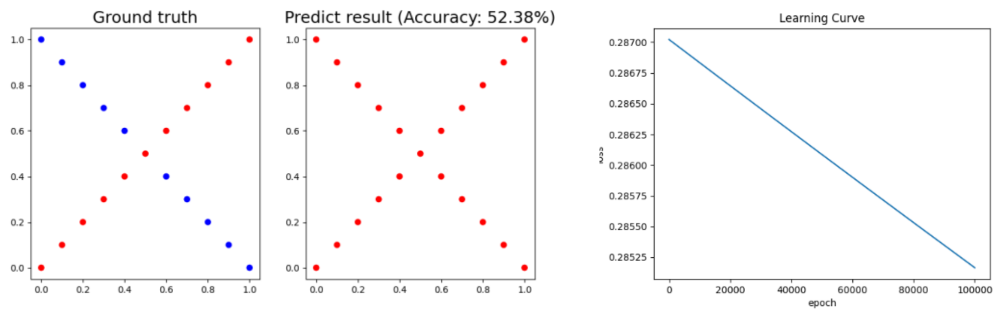
The parameters for the following results are:

1) 4 neurons / hidden layer
2) stop converge when loss < 0.005 or epoch = 100000

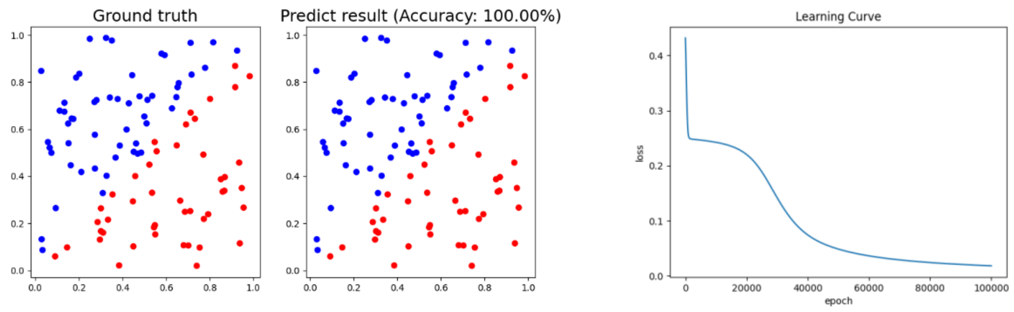
- learning rate = 1e-6

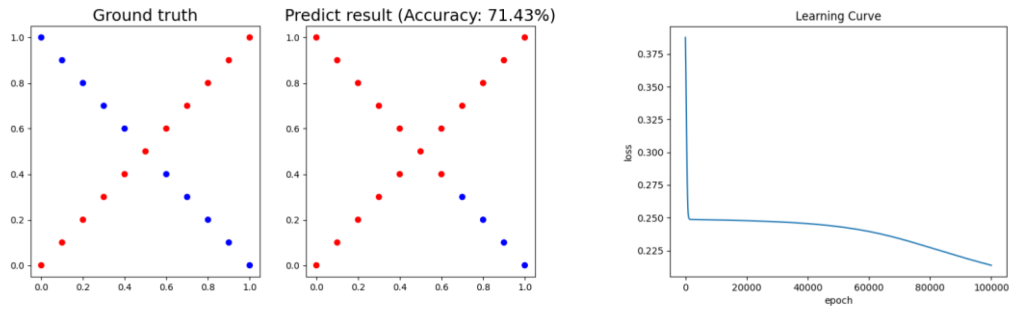




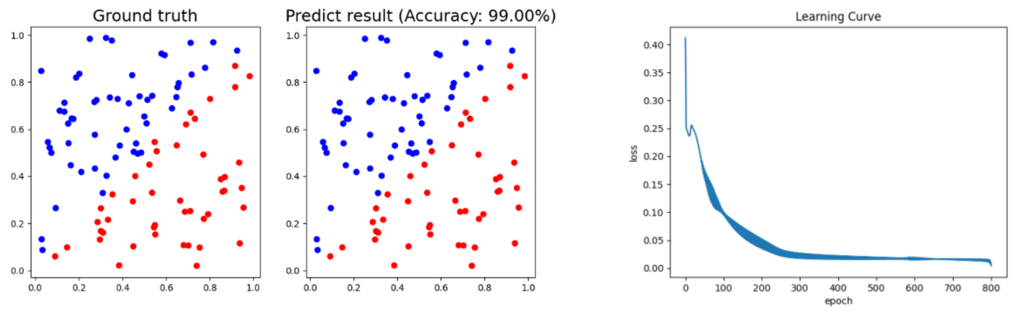


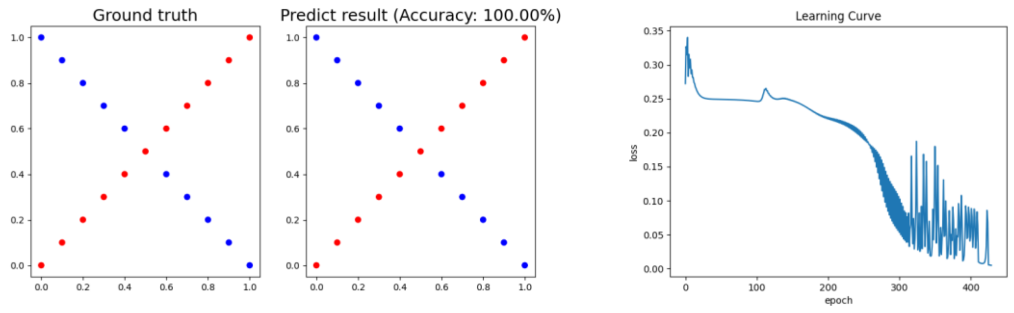
- learning rate = 1e-2



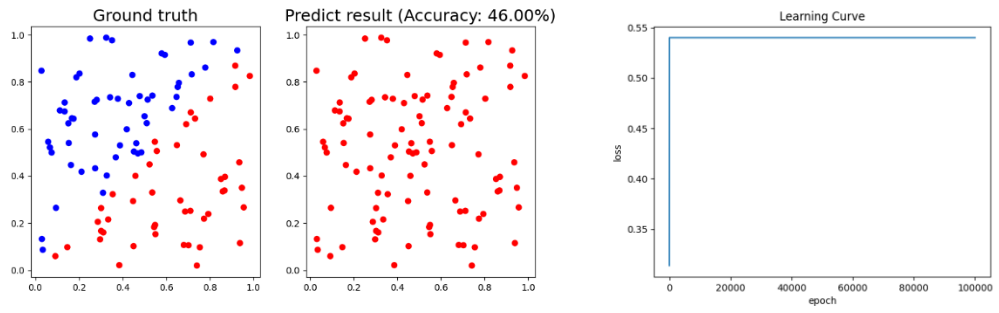


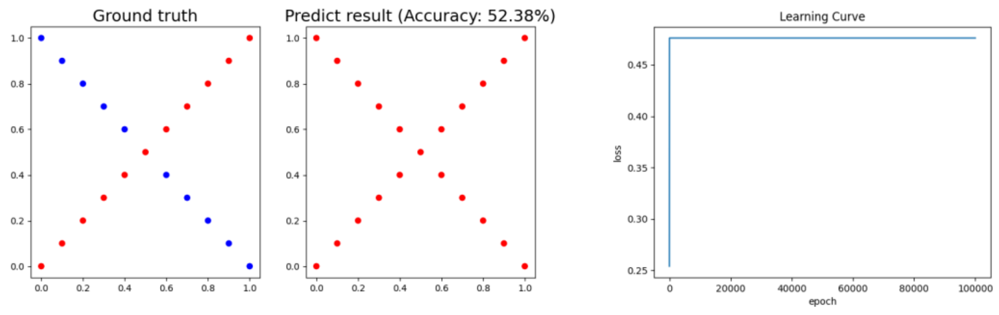
- learning rate = 10



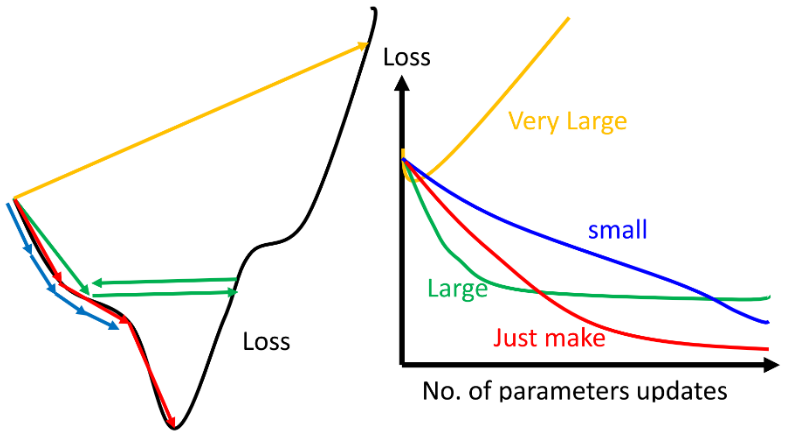


- learning rate = 1e6





If we choose a proper learning rate, the larger it is, the more quickly it converges. However, if the learning rate is too large, e.g., 1e6, it will fail to converge. Vice versa, if the learning rate is too small, e.g., 1e-6, the learning progress will be too slow.

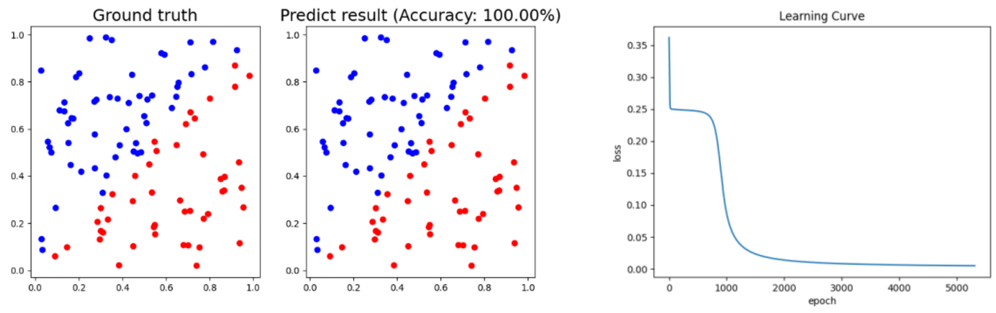


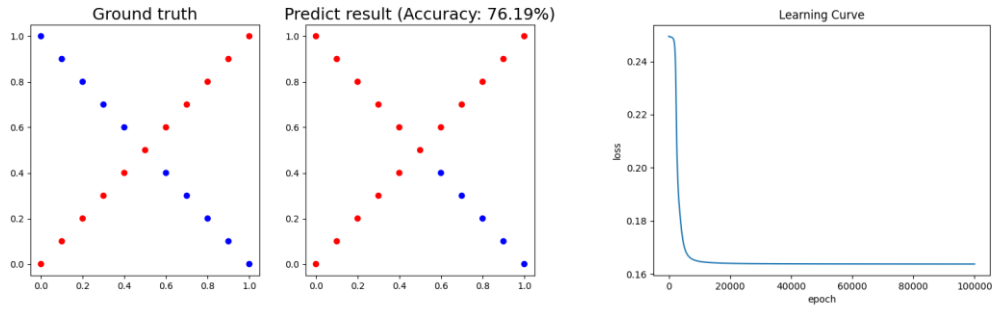
4.2 Different numbers of hidden units

The parameters for the following results are:

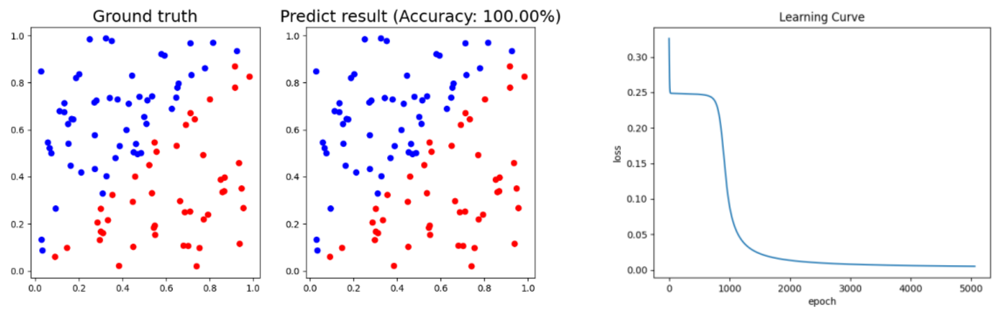
1) learning rate = 1
2) stop converge when loss < 0.005 or epoch = 100000

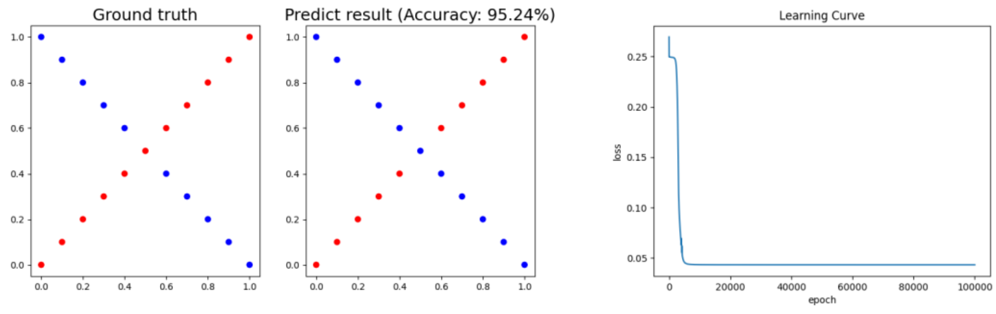
- number of hidden units in each layer: (1, 1)



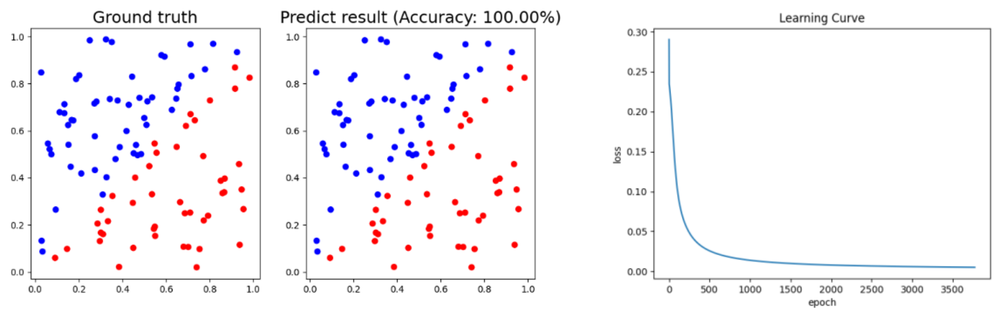


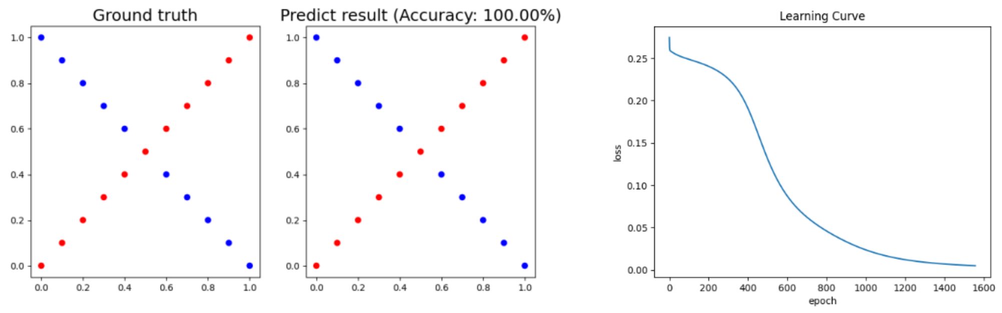
- number of hidden units in each layer: (2, 2)



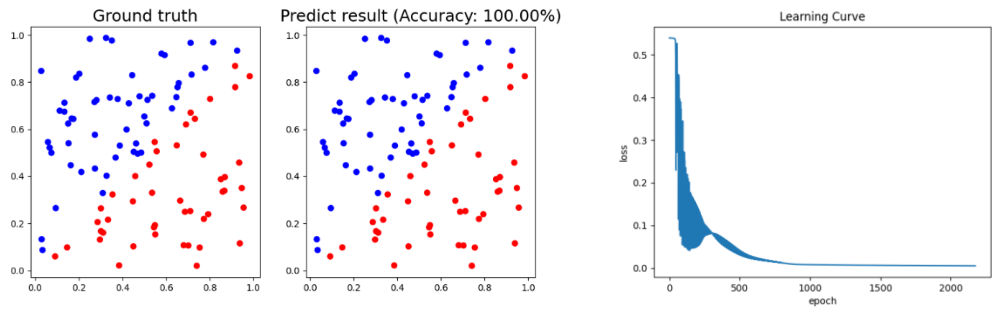


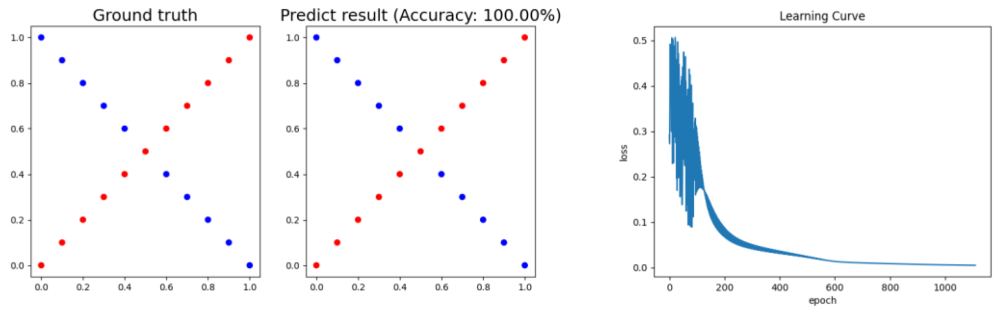
- number of hidden units in each layer: (10, 10)



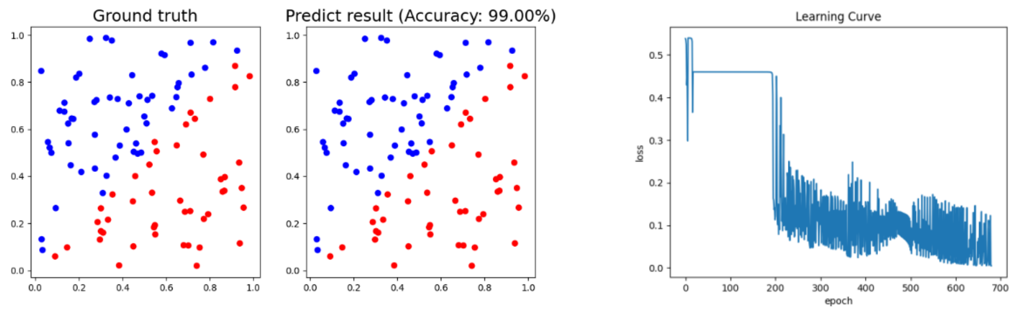


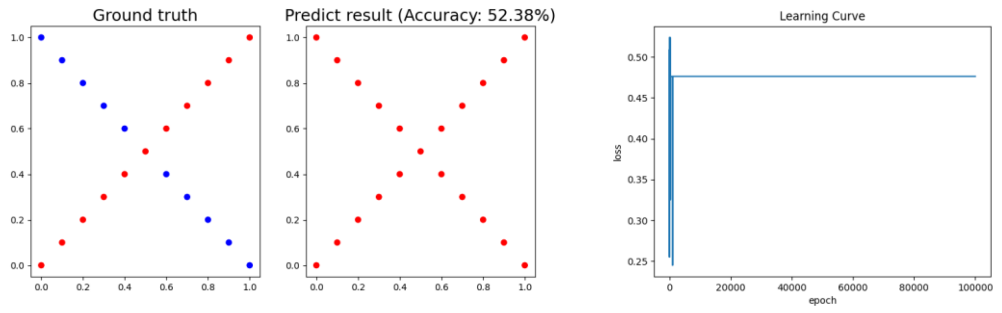
- number of hidden units in each layer: (100, 100)





- number of hidden units in each layer: (400, 400)





If the number of hidden units is not enough, difficult task might not be solved, e.g., the XOR task. However, if the number of hidden units is too large, e.g., (400, 400), the problem might not be solved as well. Also, the computation time will significantly increase due to the more complicated network architecture.

4.3 Without activation functions

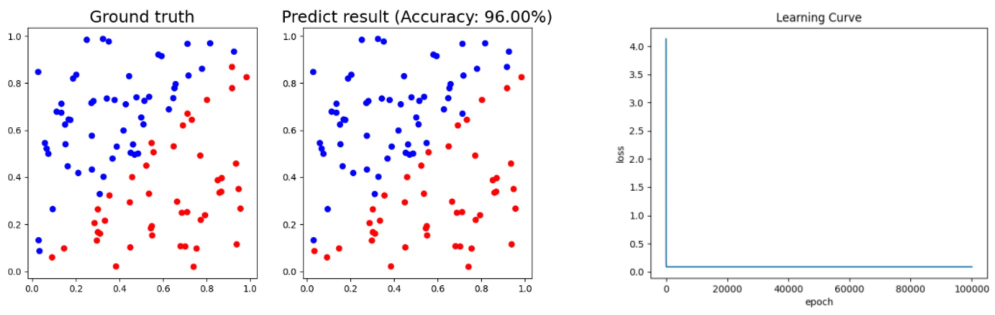
The parameters for the following results are:

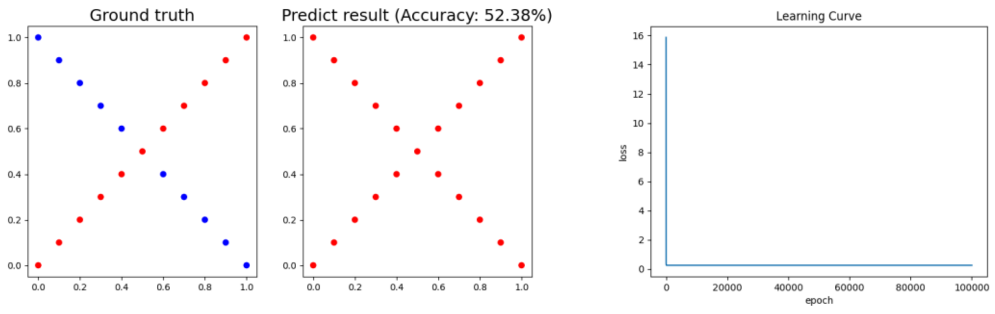
1) 4 neurons / hidden layer
2) stop converge when loss < 0.005 or epoch = 100000

- learning rate = 1

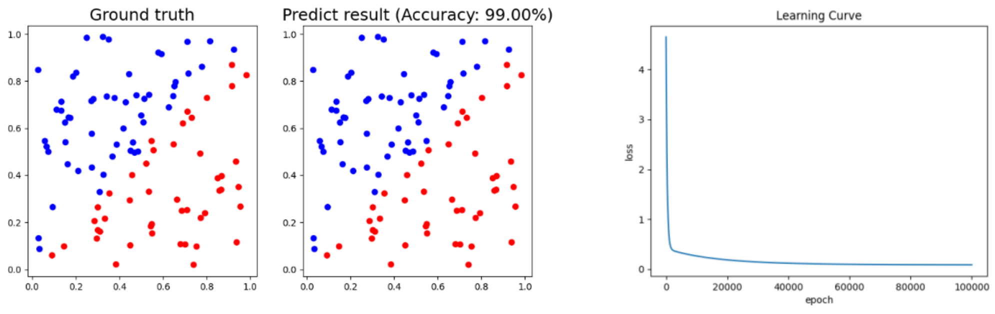
  Couldn't get the result.

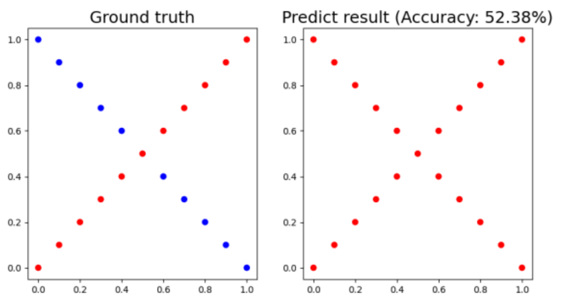
- learning rate = 1e-2

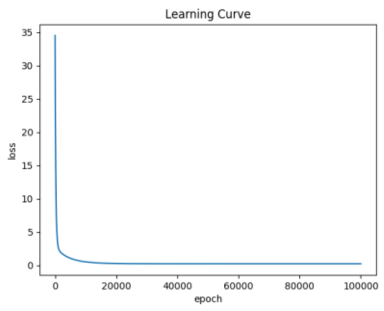




- learning rate = 1e-5







Without activation function, the gradient will possibly become infinite or minus infinite. In the linear task, it can somehow be solved by decreasing the learning rate. However, the XOR task cannot be solved without activation function.