DL Lab1 – Backpropagation

Introduction:

In this lab I implemented a neural network from scratch that can solve linear task and the XOR problem. I also tried different training parameters and layouts to compare the differences with screenshots.

Experiment setup:

A. Sigmoid function

```
def sigmoid(self,Z):
    return 1.0/(1.0 + np.exp(-Z))

def dsigmoid(self, x):
    return np.multiply(x, 1.0 - x)
```

I implemented the sigmoid function at the last layer of the neural network to squash the result between 0 and 1. I also implemented the derivative of the sigmoid for back propagation.

B. Neural Network

```
def forward_propagation(self):
    '''
    Performs the forward propagation
    '''
    Z1 = self.X.dot(self.params['W1']) + self.params['b1']
    A1 = Z1
    Z2 = A1.dot(self.params['W2']) + self.params['b2']
    A2 = self.relu(Z2)
    Z3 = A2.dot(self.params['W3']) + self.params['b3']
    yhat = self.sigmoid(Z3)
    loss = self.entropy_loss(self.y, yhat)

    self.params['Z1'] = Z1
    self.params['Z2'] = Z2
    self.params['Z3'] = Z3
    self.params['A1'] = A1
    self.params['A2'] = A2

    return yhat,loss
```

Here I implemented the forward pass of a neural network with two hidden layers, and cross entropy as my loss function.

C. Backpropagation

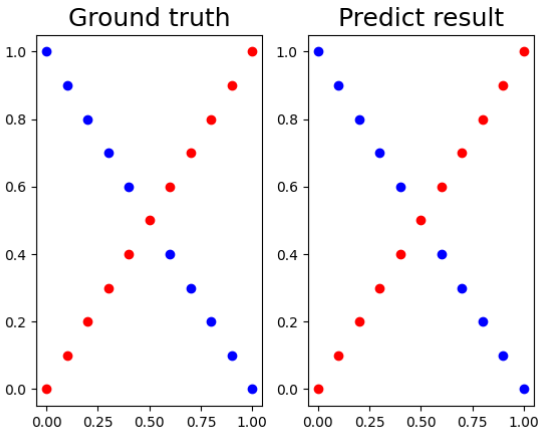
```
dl_wrt_yhat = np.divide(1 - self.y, self.eta(1 - yhat)) - np.divide(self.y, self.eta(yhat))
dl_wrt_sig = self.dsigmoid(yhat)
dl_wrt_z3 = dl_wrt_yhat * dl_wrt_sig
dl_wrt_b3 = dl_wrt_z3
dl_wrt_w3 = self.params['A2'].T.dot(dl_wrt_z3)

dl_wrt_A2 = dl_wrt_z3.dot(self.params['W3'].T)
dl_wrt_z2 = dl_wrt_A2 * self.dRelu(self.params['Z2'])
dl_wrt_b2 = dl_wrt_z2
dl_wrt_w2 = self.params['A1'].T.dot(dl_wrt_z2)

dl_wrt_A1 = dl_wrt_z2.dot(self.params['W2'].T)
dl_wrt_z1 = dl_wrt_A1
dl_wrt_b1 = dl_wrt_z1
dl_wrt_w1 = self.X.T.dot(dl_wrt_z1)
```

Calculate the derivatives backwards to do gradient descent.

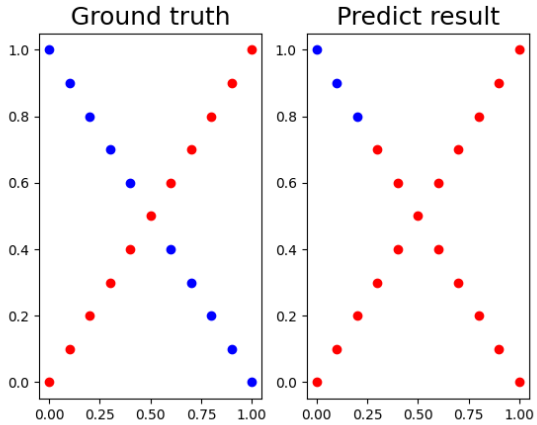
Results of your testing:

Update after the whole dataset v.s Update after one sample

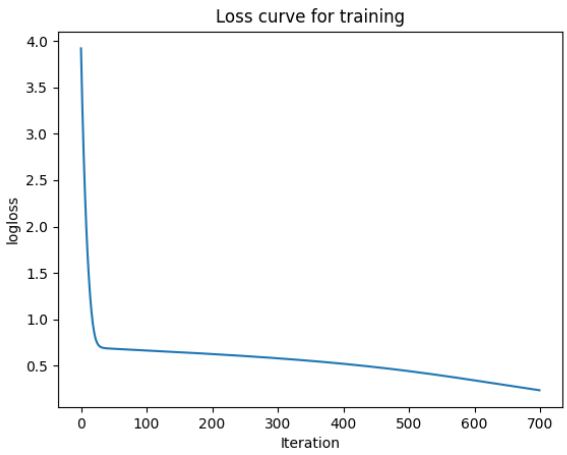
xor:

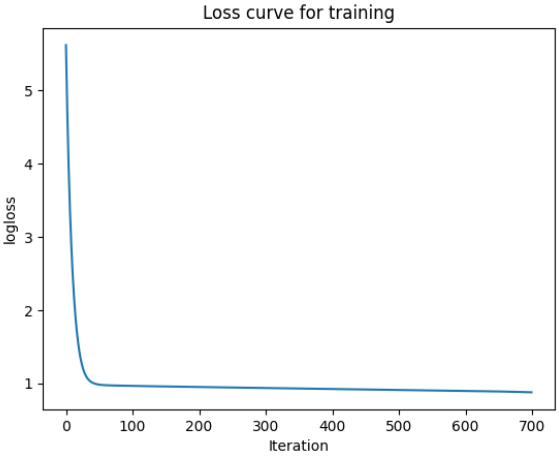


accuracy is 100



accuracy is 66



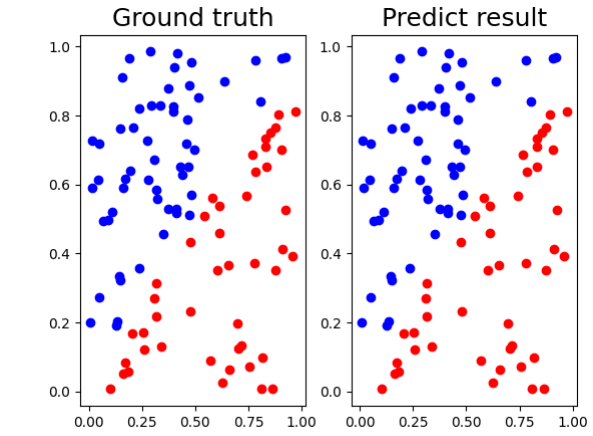



```
[[0.39 ]  [[0.468]
 [0.823]   [0.531]
 [0.348]   [0.474]
 [0.82 ]   [0.519]
 [0.308]   [0.479]
 [0.816]   [0.508]
 [0.27 ]   [0.479]
 [0.813]   [0.496]
 [0.234]   [0.476]
 [0.809]   [0.485]
 [0.202]   [0.473]
 [0.172]   [0.47 ]
 [0.802]   [0.461]
 [0.145]   [0.467]
 [0.798]   [0.45 ]
 [0.121]   [0.463]
 [0.795]   [0.438]
 [0.1  ]   [0.46 ]
 [0.791]   [0.427]
 [0.087]   [0.457]
 [0.788]]  [0.416]]
```

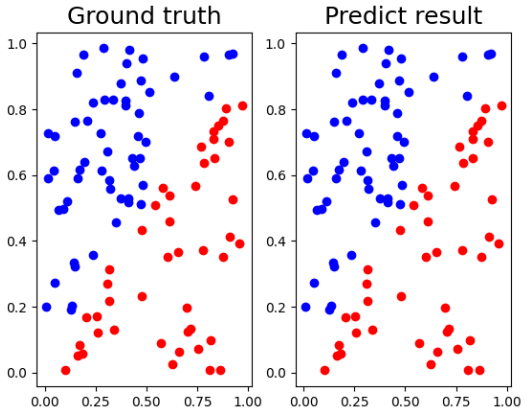
```
epoch 689 loss : 0.2462248245832326
epoch 690 loss : 0.24517705980795174
epoch 691 loss : 0.24413642103306288
epoch 692 loss : 0.24309173471901346
epoch 693 loss : 0.24205484859614992
epoch 694 loss : 0.24101341658849537
epoch 695 loss : 0.23998034627187498
epoch 696 loss : 0.2389423383540188
epoch 697 loss : 0.2379131502160636
epoch 698 loss : 0.2368787312201362
epoch 699 loss : 0.23585349381134033
```

```
epoch 689 loss : 0.8841632812455896
epoch 690 loss : 0.8838900684538183
epoch 691 loss : 0.883617690631203
epoch 692 loss : 0.883345875115504
epoch 693 loss : 0.8830743810133532
epoch 694 loss : 0.8828029952808565
epoch 695 loss : 0.8825315292876154
epoch 696 loss : 0.8822598158044654
epoch 697 loss : 0.8819877063626015
epoch 698 loss : 0.881715068938251
epoch 699 loss : 0.8814417859227005
```

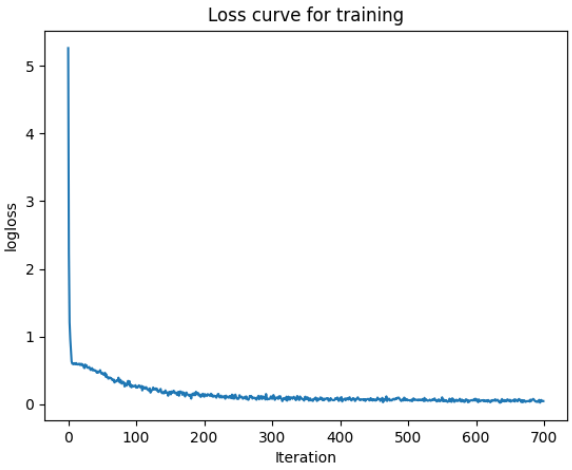
Linear:

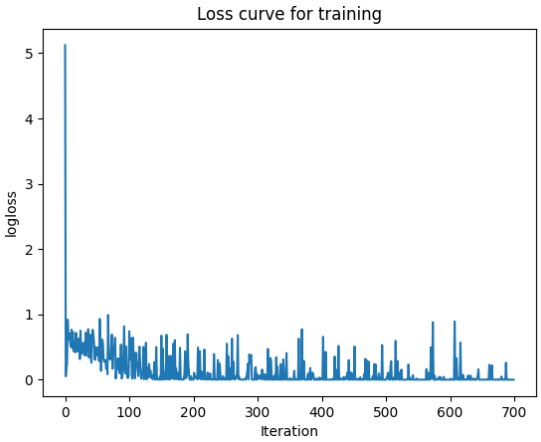


accuracy is 100



accuracy is 100



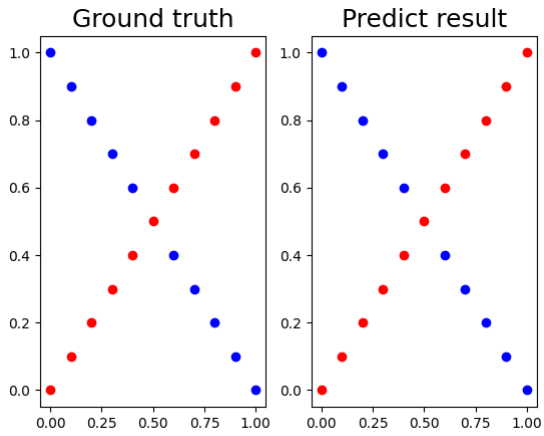


```
[0.997]  [0.998]
[0.   ]  [0.   ]
[0.679]  [0.705]
[0.971]  [0.974]
[0.611]  [0.691]
[0.   ]  [0.   ]
[0.84 ]  [0.858]
[0.002]  [0.001]
[1.   ]  [1.   ]
[1.   ]  [1.   ]
[0.997]  [0.998]
[1.   ]  [1.   ]
[0.014]  [0.02 ]
[0.169]  [0.136]
[0.998]  [0.998]
[0.014]  [0.013]
[0.005]  [0.005]
[0.177]  [0.173]
[1.   ]  [1.   ]
[0.004]  [0.003]
[1.   ]  [1.   ]
[0.001]  [0.001]
[1.   ]  [1.   ]
[0.016]  [0.017]
[0.215]] [0.234]]
```

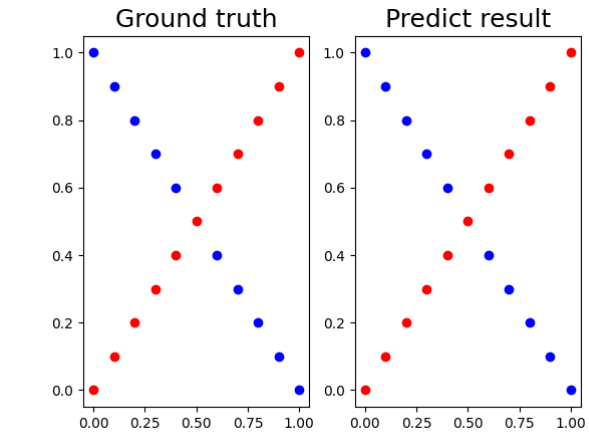
```
epoch 686 loss : 0.054519815889939836  epoch 686 loss : 2.7067593509821473e-05
epoch 687 loss : 0.03946056901835811   epoch 687 loss : 0.259821662087738
epoch 688 loss : 0.04280865511881737   epoch 688 loss : 0.003862479667980215
epoch 689 loss : 0.03262623299434872   epoch 689 loss : 0.0009665122248297324
epoch 690 loss : 0.034708071922369514  epoch 690 loss : 0.0002473167158310313
epoch 691 loss : 0.03264219635664099   epoch 691 loss : 0.00015043835479316352
epoch 692 loss : 0.05560056886403581   epoch 692 loss : 0.00022564638456914002
epoch 693 loss : 0.06182715332610179   epoch 693 loss : 4.240829909472165e-12
epoch 694 loss : 0.020950840620454655  epoch 694 loss : 1.0381595971628e-06
epoch 695 loss : 0.04361696244291182   epoch 695 loss : 1.0734464152999837e-08
epoch 696 loss : 0.06029565742538862   epoch 696 loss : 7.017944412530096e-05
epoch 697 loss : 0.046484029640714644  epoch 697 loss : 1.407606391713631e-05
epoch 698 loss : 0.04127385695060533   epoch 698 loss : 0.0006583394239180478
epoch 699 loss : 0.04423107770643497   epoch 699 loss : 6.75015598972118e-14
```

With sigmoid function v.s Without sigmoid function when predicting

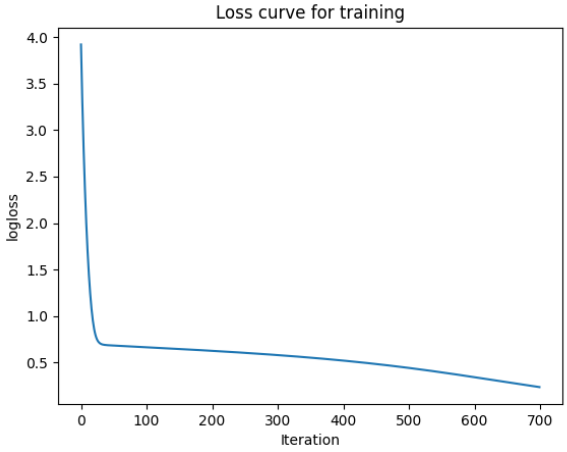
xor:

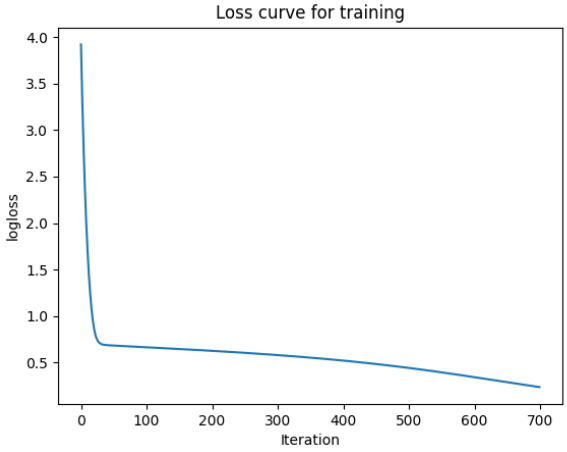


accuracy is 100



accuracy is 100



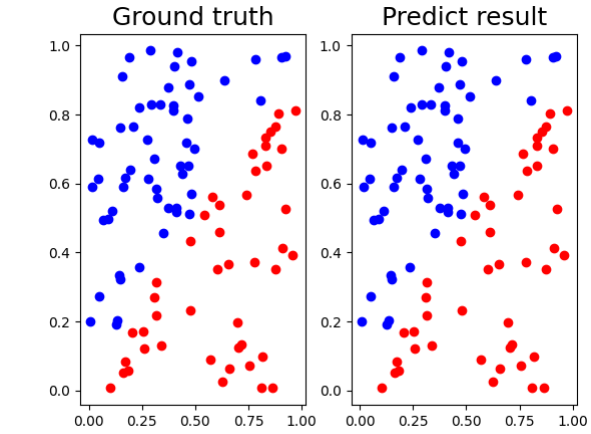



```
[[0.39 ]   [[-0.633]
 [0.823]    [ 1.212]
 [0.348]   [-0.812]
 [0.82 ]    [ 1.291]
 [0.308]   [-0.993]
 [0.816]    [ 1.371]
 [0.27 ]   [-1.179]
 [0.813]    [ 1.453]
 [0.234]   [-1.369]
 [0.809]    [ 1.535]
 [0.202]   [-1.565]
 [0.172]   [-1.767]
 [0.802]    [ 1.703]
 [0.145]   [-1.975]
 [0.798]    [ 1.79 ]
 [0.121]   [-2.19 ]
 [0.795]    [ 1.878]
 [0.1  ]   [-2.411]
 [0.791]    [ 1.968]
 [0.087]   [-2.639]
 [0.788]]   [ 2.06 ]]
```

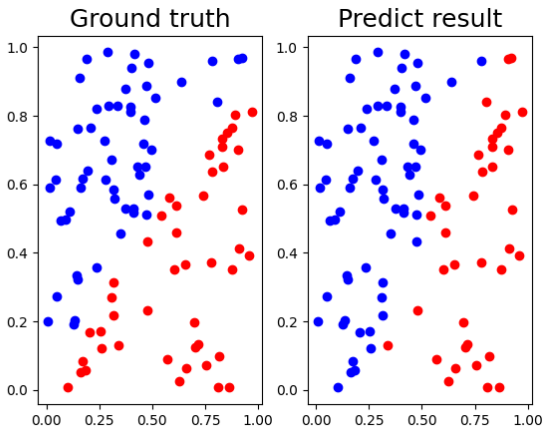
```
epoch 689 loss : 0.2462248245832326
epoch 690 loss : 0.24517705980795174
epoch 691 loss : 0.24413642103306288
epoch 692 loss : 0.24309173471901346
epoch 693 loss : 0.24205484859614992
epoch 694 loss : 0.24101341658849537
epoch 695 loss : 0.23998034627187498
epoch 696 loss : 0.2389423383540188
epoch 697 loss : 0.2379131502160636
epoch 698 loss : 0.2368787312201362
epoch 699 loss : 0.23585349381134033
```

```
epoch 689 loss : 0.20623597687378759
epoch 690 loss : 0.20562436748125007
epoch 691 loss : 0.20501397274920455
epoch 692 loss : 0.2044047954632363
epoch 693 loss : 0.20379683837798293
epoch 694 loss : 0.2031901042170719
epoch 695 loss : 0.2025845956730583
epoch 696 loss : 0.2019803154073651
epoch 697 loss : 0.20137726605022574
epoch 698 loss : 0.20077545020062812
epoch 699 loss : 0.20017487042625962
```

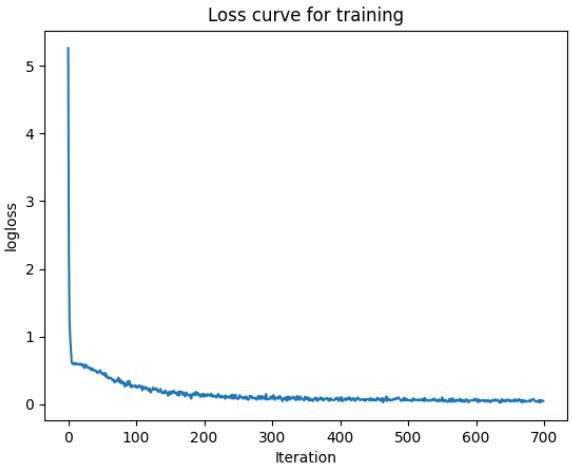
Linear:

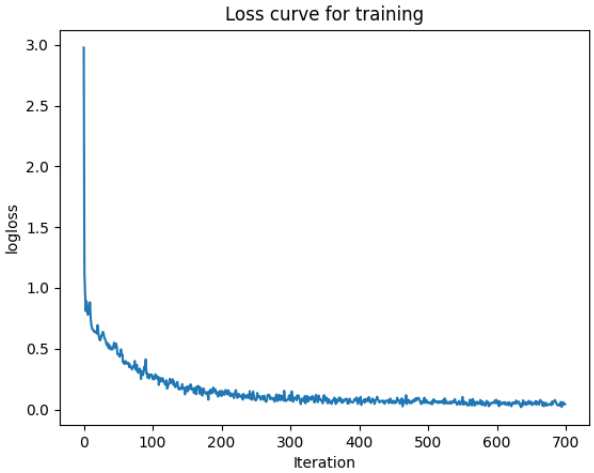


accuracy is 100



accuracy is 86





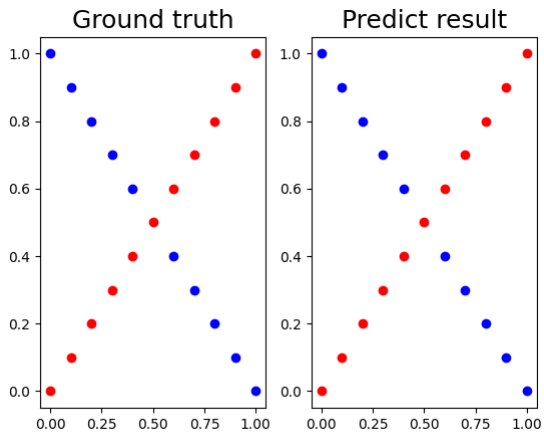
```
[0.997]  [  0.668]
[0.   ]  [  3.402]
[0.679]  [  0.356]
[0.971]  [-14.604]
[0.611]  [  1.583]
[0.   ]  [ -6.236]
[0.84 ]  [ 18.291]
[0.002]  [ 15.869]
[1.   ]  [  5.853]
[1.   ]  [ 23.119]
[0.997]  [ -4.344]
[1.   ]  [ -1.641]
[0.014]  [  5.905]
[0.169]  [ -4.274]
[0.998]  [ -5.266]
[0.014]  [ -1.605]
[0.005]  [ 25.465]
[0.177]  [ -5.588]
[1.   ]  [ 13.695]
[0.004]  [ -8.801]
[1.   ]  [ 19.193]
[0.001]  [ -4.2  ]
[1.   ]  [ -1.351]]
[0.016]
[0.215]]
```

```
epoch 686 loss : 0.054519815889939836
epoch 687 loss : 0.03946056901835811
epoch 688 loss : 0.04280865511881737
epoch 689 loss : 0.03262623299434872
epoch 690 loss : 0.034708071922369514
epoch 691 loss : 0.03264219635664099
epoch 692 loss : 0.05560056886403581
epoch 693 loss : 0.06182715332610179
epoch 694 loss : 0.020950840620454655
epoch 695 loss : 0.04361696244291182
epoch 696 loss : 0.06029565742538862
epoch 697 loss : 0.046484029640714644
epoch 698 loss : 0.04127385695060533
epoch 699 loss : 0.04423107770643497
```

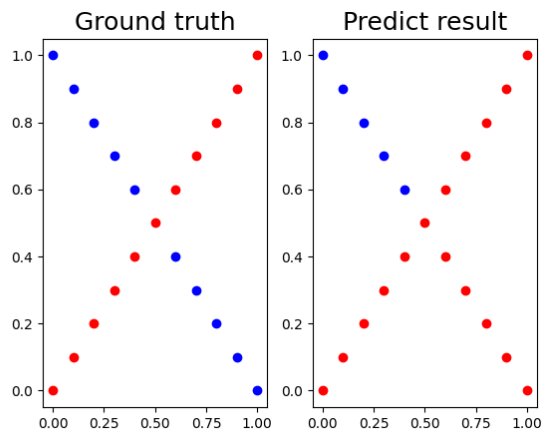
```
epoch 689 loss : 0.032748292840919124
epoch 690 loss : 0.03483563589127151
epoch 691 loss : 0.03294610570727348
epoch 692 loss : 0.0561183903714788
epoch 693 loss : 0.06228371782011705
epoch 694 loss : 0.02107793429350643
epoch 695 loss : 0.043996783614895084
epoch 696 loss : 0.06064515768635174
epoch 697 loss : 0.04759557969494921
epoch 698 loss : 0.04132159670002754
epoch 699 loss : 0.0448557932299296
```

Learning rate 0.001 v.s Learning rate 0.0001

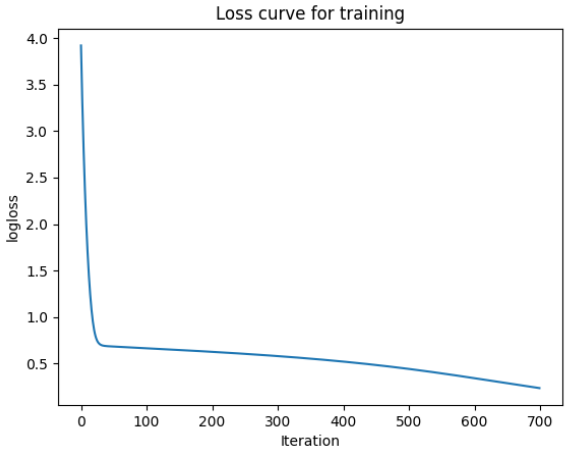
xor:

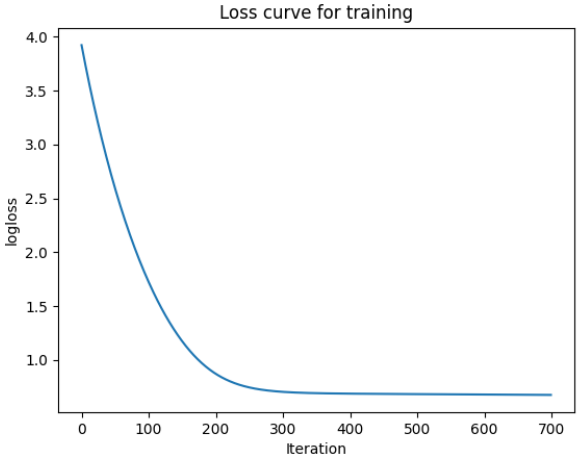


accuracy is 100



accuracy is 76



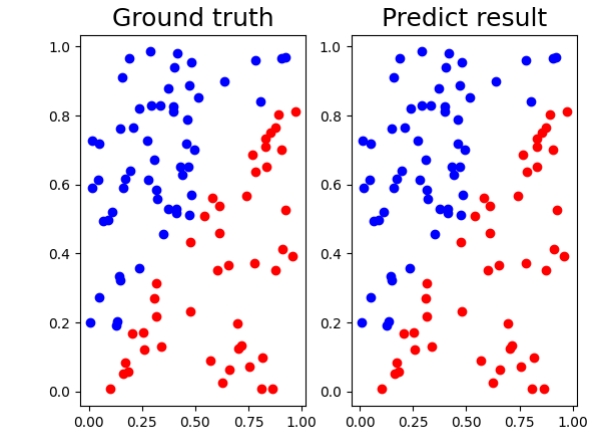



```
[[0.39 ]  [[0.456]
 [0.823]   [0.589]
 [0.348]   [0.457]
 [0.82 ]   [0.567]
 [0.308]   [0.457]
 [0.816]   [0.545]
 [0.27 ]   [0.457]
 [0.813]   [0.523]
 [0.234]   [0.457]
 [0.809]   [0.501]
 [0.202]   [0.457]
 [0.172]   [0.457]
 [0.802]   [0.457]
 [0.145]   [0.457]
 [0.798]   [0.435]
 [0.121]   [0.457]
 [0.795]   [0.413]
 [0.1  ]   [0.457]
 [0.791]   [0.392]
 [0.087]   [0.457]
 [0.788]]  [0.371]]
```

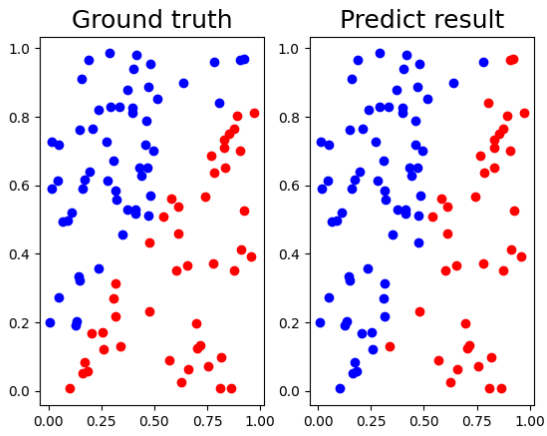
```
epoch 689 loss : 0.2462248245832326
epoch 690 loss : 0.24517705980795174
epoch 691 loss : 0.24413642103306288
epoch 692 loss : 0.24309173471901346
epoch 693 loss : 0.24205484859614992
epoch 694 loss : 0.24101341658849537
epoch 695 loss : 0.23998034627187498
epoch 696 loss : 0.2389423383540188
epoch 697 loss : 0.2379131502160636
epoch 698 loss : 0.2368787312201362
epoch 699 loss : 0.23585349381134033
```

```
epoch 689 loss : 0.6755091846455316
epoch 690 loss : 0.6754727279470312
epoch 691 loss : 0.6754362712386675
epoch 692 loss : 0.6753998145100173
epoch 693 loss : 0.675363357750778
epoch 694 loss : 0.6753269009507648
epoch 695 loss : 0.6752904440999065
epoch 696 loss : 0.6752539871882443
epoch 697 loss : 0.6752175302059265
epoch 698 loss : 0.6751810731432064
epoch 699 loss : 0.67514461599044
```

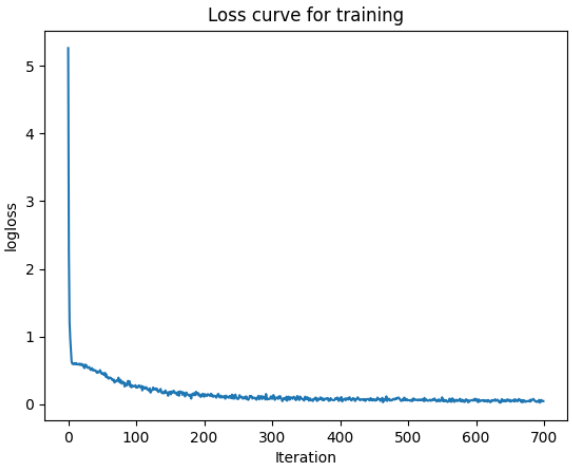
Linear:

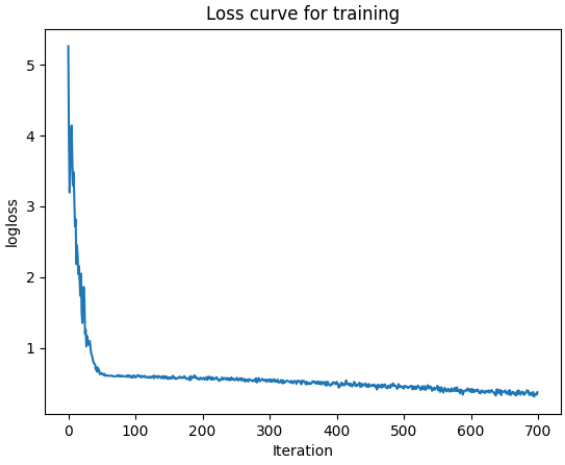


accuracy is 100



accuracy is 86



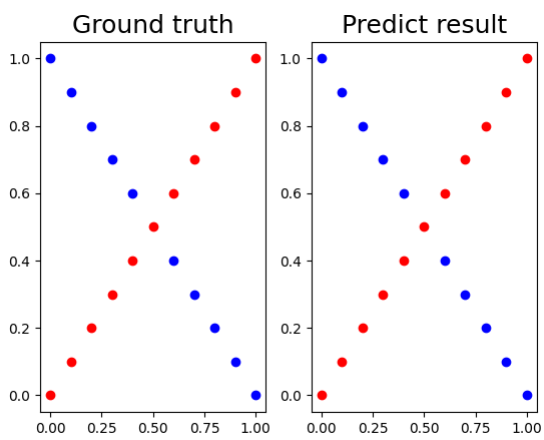


```
[0.997]
[0.   ] [0.114]
[0.679]
[0.971] [0.335]
[0.611] [0.507]
[0.   ] [0.647]
[0.84 ] [0.901]
[0.002] [0.597]
[1.   ] [0.098]
[1.   ] [0.536]
[0.997] [0.49 ]
[1.   ] [0.929]
[0.014] [0.387]
[0.169] [0.794]
[0.998] [0.382]
[0.014] [0.881]
[0.005] [0.324]
[0.177] [0.478]]
[1.   ]
[0.004]
[1.   ]
[0.001]
[1.   ]
[0.016]
[0.215]]
```

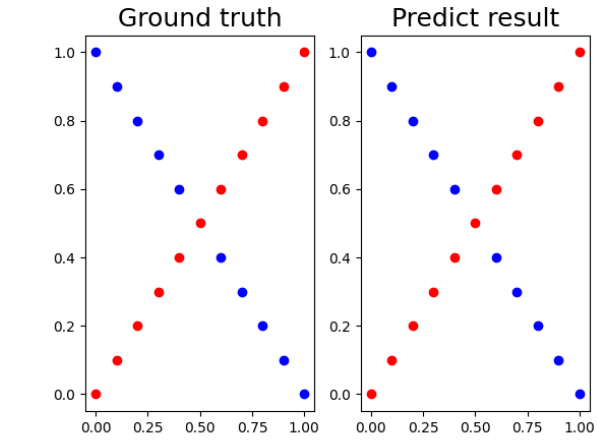
```
epoch 686 loss : 0.054519815889939836
epoch 687 loss : 0.03946056901835811
epoch 688 loss : 0.04280865511881737
epoch 689 loss : 0.03262623299434872
epoch 690 loss : 0.034708071922369514
epoch 691 loss : 0.03264219635664099
epoch 692 loss : 0.05560056886403581
epoch 693 loss : 0.06182715332610179
epoch 694 loss : 0.020950840620454655
epoch 695 loss : 0.04361696244291182
epoch 696 loss : 0.06029565742538862
epoch 697 loss : 0.046484029640714644
epoch 698 loss : 0.04127385695060533
epoch 699 loss : 0.04423107770643497
epoch 689 loss : 0.3639885517251373
epoch 690 loss : 0.3556023363504343
epoch 691 loss : 0.32962735723258474
epoch 692 loss : 0.3772534453161206
epoch 693 loss : 0.372905259723417
epoch 694 loss : 0.3120415427551752
epoch 695 loss : 0.3385518249849669
epoch 696 loss : 0.34545821690081185
epoch 697 loss : 0.3549486145209989
epoch 698 loss : 0.3392183541288813
epoch 699 loss : 0.37499919585645003
```

[2, 4, 4, 1] hidden units layout v.s [2, 1, 1, 1] hidden units layout

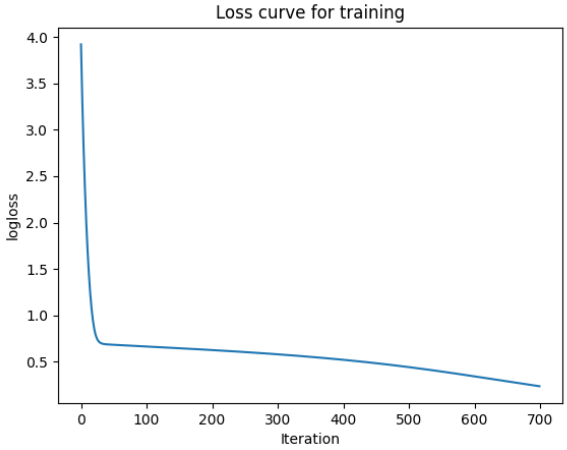
xor:

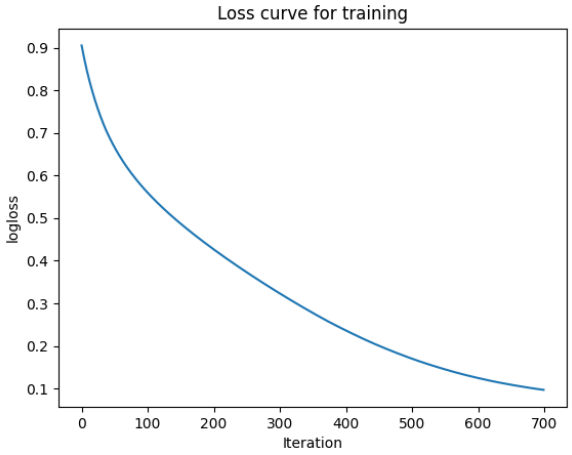


accuracy is 100



accuracy is 100



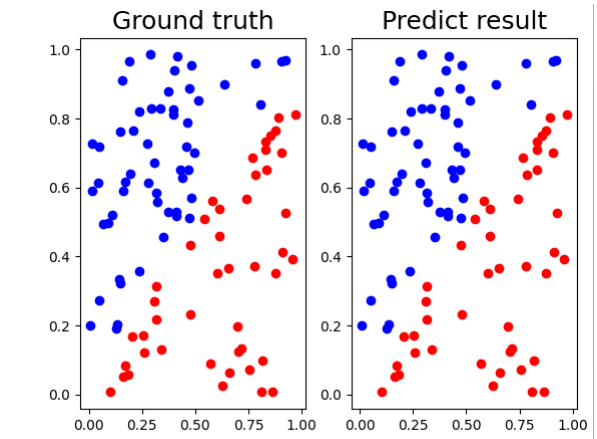



```
[[0.39 ]  [[0.025]
 [0.823]   [0.863]
 [0.348]   [0.03 ]
 [0.82 ]   [0.868]
 [0.308]   [0.035]
 [0.816]   [0.873]
 [0.27 ]   [0.041]
 [0.813]   [0.877]
 [0.234]   [0.047]
 [0.809]   [0.878]
 [0.202]   [0.054]
 [0.172]   [0.062]
 [0.802]   [0.873]
 [0.145]   [0.069]
 [0.798]   [0.87 ]
 [0.121]   [0.077]
 [0.795]   [0.868]
 [0.1  ]   [0.086]
 [0.791]   [0.866]
 [0.087]   [0.095]
 [0.788]]  [0.865]]
```

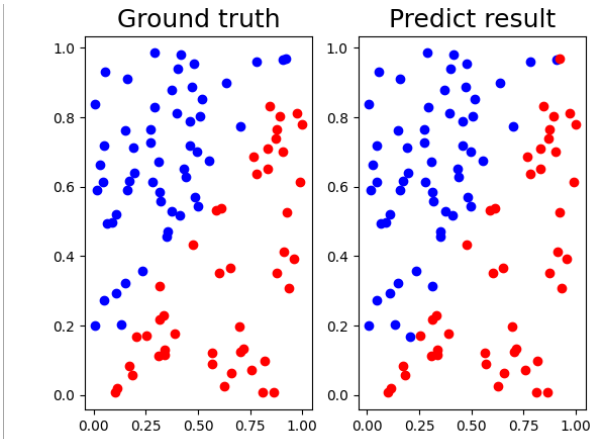
```
epoch 689 loss : 0.2462248245832326
epoch 690 loss : 0.24517705980795174
epoch 691 loss : 0.24413642103306288
epoch 692 loss : 0.24309173471901346
epoch 693 loss : 0.24205484859614992
epoch 694 loss : 0.24101341658849537
epoch 695 loss : 0.23998034627187498
epoch 696 loss : 0.2389423383540188
epoch 697 loss : 0.2379131502160636
epoch 698 loss : 0.2368787312201362
epoch 699 loss : 0.23585349381134033
```

```
epoch 689 loss : 0.09913133608853518
epoch 690 loss : 0.0989063784116856
epoch 691 loss : 0.09868268885190101
epoch 692 loss : 0.09846990402999885
epoch 693 loss : 0.09826632577277987
epoch 694 loss : 0.09804600246227907
epoch 695 loss : 0.09784078638704656
epoch 696 loss : 0.09763374599727848
epoch 697 loss : 0.09743533134147064
epoch 698 loss : 0.09723601424569916
epoch 699 loss : 0.09702077340819475
```

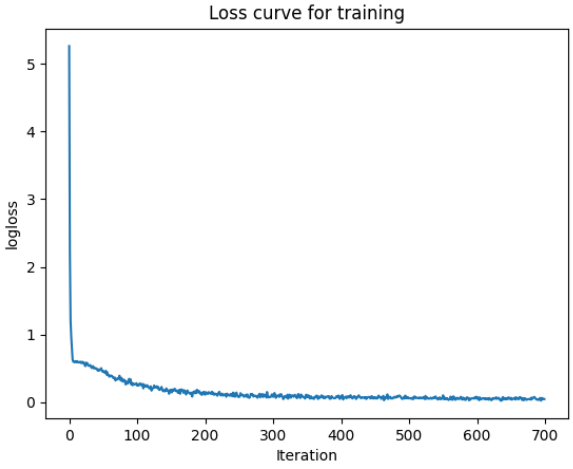
Linear:

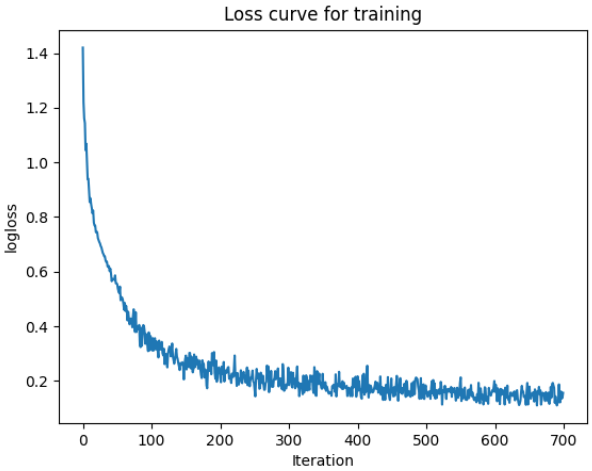


accuracy is 100



accuracy is 97





```
[0.997]
[0.   ] [0.645]
[0.679] [0.   ]
[0.971]
[0.611] [0.854]
[0.   ] [0.854]
[0.84 ] [0.   ]
[0.002]
[1.   ] [0.493]
[1.   ] [0.193]
[0.997]
[1.   ] [0.853]
[0.014] [0.   ]
[0.169]
[0.998] [0.854]
[0.014] [0.855]
[0.005] [0.854]
[0.177] [0.854]
[1.   ] [0.854]
[0.004]
[1.   ] [0.854]
[0.001] [0.   ]
[1.   ] [0.854]
[0.016]
[0.215]] [0.   ]]
```

```
epoch 686 loss : 0.054519815889939836
epoch 687 loss : 0.03946056901835811
epoch 688 loss : 0.04280865511881737
epoch 689 loss : 0.03262623299434872
epoch 690 loss : 0.034708071922369514
epoch 691 loss : 0.03264219635664099
epoch 692 loss : 0.05560056886403581
epoch 693 loss : 0.06182715332610179
epoch 694 loss : 0.020950840620454655
epoch 695 loss : 0.04361696244291182
epoch 696 loss : 0.06029565742538862
epoch 697 loss : 0.046484029640714644
epoch 698 loss : 0.04127385695060533
epoch 699 loss : 0.04423107770643497
epoch 689 loss : 0.1243729070609671
epoch 690 loss : 0.10980638803808423
epoch 691 loss : 0.12078117229348032
epoch 692 loss : 0.12198194436887805
epoch 693 loss : 0.18696233018919592
epoch 694 loss : 0.12146768822464302
epoch 695 loss : 0.12175732144876227
epoch 696 loss : 0.16090197431265826
epoch 697 loss : 0.1351860016293751
epoch 698 loss : 0.13802619971243282
epoch 699 loss : 0.15599850833197468
```

Discussion:

A. Try different learning rates

I tried learning rates of 0.001 and 0.0001 with 700 epoch. From the results we can clearly see that learning rates of 0.0001 has not converged yet, thus the performance is worse.

B. Try different number of hidden units

The [2, 1, 1, 1] layout converged much more smoothly the that of [2, 4, 4, 1] layout, and has a much lower loss in the initial stages of training.

Maybe because it is easier for smaller numbers of units to learn easy features such as those in the dataset of this Lab.

C. Try without activation functions

I trained the network with sigmoid functions to squash the value in 0~1 and calculate loss, then remove it when predicting the results. The results are a bit weird as the theory goes that no amount of perceptrons are able to solve the xor problem due to it's linearity. But even without a sigmoid function it's still able to solve the xor problem perfectly.

D. Update after different amounts of data

Originally, I only update the network after the entire dataset (normal gradient descent). Then I tried to update the network after a single sample data (close to stochastic gradient descent). And the results showed that updating after a single sample performs worse than updating after the entire dataset. Because updating only based on one sample is just like stochastic gradient descent, the direction isn't necessarily optimal. Furthermore, in a task as simple as this, there is no point using stochastic gradient descent.

Extra:

A. Relu

```
def relu(self,Z):
    return np.maximum(0,Z)
```

```
def dRelu(self, x):
    x[x<=0] = 0
    x[x>0] = 1
    return x
```

I implemented relu function in my network but the difference is minimal.