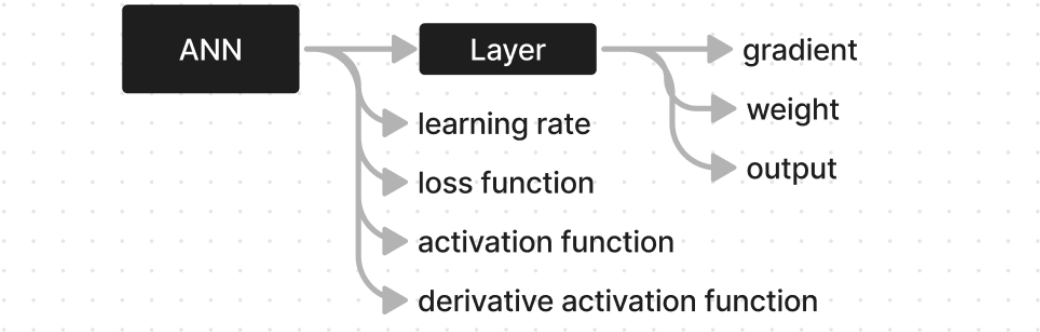
Deep Learning Lab1 Student ID: 312554014 Name: 羅翊軒

1. Introduction



The picture above is the structure of my simple neural network, which datatype is class. It includes some members to store information of a network, such as layer, learning rate, loss function, and so on. One of the most important things is that information of each layer in the network is also stored in datatype class, and it owns some crucial data like weight, gradient, etc. Overall, the whole neural network is a class, and each hidden layer or output layer in the network is a class as well.

2. Experiment Setups

A. Sigmoid functions

The default activation function of the neural network is sigmoid function. It will map every variable to [0, 1], and the derivative of sigmoid function has no negative value.

```
# sigmoid activation function
def sigmoid(self, x):
    return 1.0/(1.0 + np.exp(-x))
def derivative_sigmoid(self, x):
    return np.multiply(x, 1.0 - x)
```

B. Neural network

```
model = ANN([2, 3, 3, 1], 0.1, "without")
```

Passing three parameters to creating a neural network object, and it represents neurons of each layer, learning rate, and activation function, respectively. Datatype of the first parameter is list, and it can determine not only the neuron numbers of each layer but also the layer numbers. For example, [2, 3, 3, 1] means that there are 2 values for input layer, 3 neurons for hidden layer 1, 3 neurons for hidden layer 2, and 1 value for output. For second and third parameter represent learning rate = 0.1 and use sigmoid activation.

C. Backpropagation

```
self.inputs = np.array([xi])
yhat = self.forwardPass([xi])
error = yhat - [yi]
self.backwardPass(error)
self.update_weights()
loss = self._lossFunction(xi, yi)
pred_y = 1.0 if yhat > 0.5 else 0.0
acc = self.calculate_acc(yi, pred_y)
print("[ Epoch{} / Step{} ] pred_y:{}, actual_y:{}, loss:{:.6f}, acc:{}".format(i+1, j+1, yhat.flatten(), [yi], loss, acc))
```

For each input, it will pass input into network and calculate predict y. After y is calculated, then measures the error between predict y and actual y, and compute each gradient and update the weights.

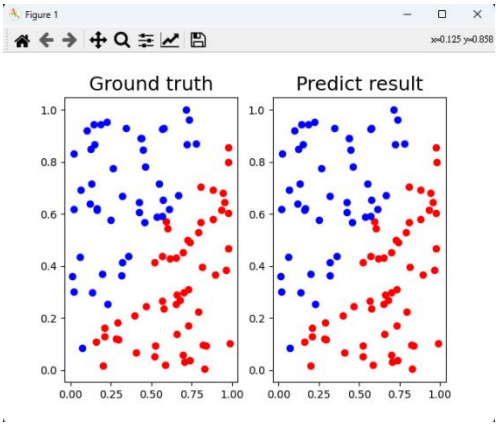
3. Results of your testing

```
train_meta = np.array(model.train(x, y, 2000))
pred_y, acc = model.test(x, y)
```

```
[ Epoch2000 / Step89 ] pred_y:[0.99998727], actual_y:[array([1])], loss:0.000000, acc:1.0
[ Epoch2000 / Step90 ] pred_y:[0.00030098], actual_y:[array([0])], loss:0.000000, acc:1.0
[ Epoch2000 / Step91 ] pred_y:[0.99998599], actual_y:[array([1])], loss:0.000000, acc:1.0
[ Epoch2000 / Step92 ] pred_y:[0.00034677], actual_y:[array([0])], loss:0.000000, acc:1.0
[ Epoch2000 / Step93 ] pred_y:[0.00035859], actual_y:[array([0])], loss:0.000000, acc:1.0
[ Epoch2000 / Step94 ] pred_y:[0.99656763], actual_y:[array([1])], loss:0.000012, acc:1.0
[ Epoch2000 / Step95 ] pred_y:[0.0003245], actual_y:[array([0])], loss:0.000000, acc:1.0
[ Epoch2000 / Step96 ] pred_y:[0.00056484], actual_y:[array([0])], loss:0.000000, acc:1.0
[ Epoch2000 / Step97 ] pred_y:[0.999988], actual_y:[array([1])], loss:0.000000, acc:1.0
[ Epoch2000 / Step98 ] pred_y:[0.00041942], actual_y:[array([0])], loss:0.000000, acc:1.0
[ Epoch2000 / Step99 ] pred_y:[0.96521025], actual_y:[array([1])], loss:0.001205, acc:1.0
[ Epoch2000 / Step100 ] pred_y:[0.99998434], actual_y:[array([1])], loss:0.000000, acc:1.0
Accuracy: 100.0%
```

(Picture above is a part of result for execution)

A. Screenshot and comparison figure



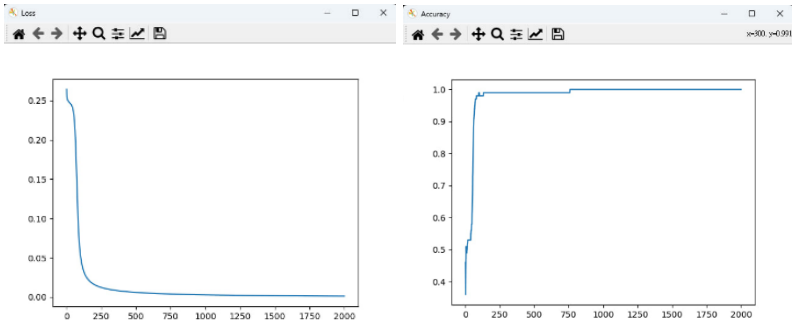
Picture on the left-hand side is comparison figure of ground truth and predict result. After training for 2000 epochs by linear dataset, dataset is correctly classified.

B. Show the accuracy of your prediction

```
[ Epoch2000 / Step98 ] pred_y:[0.00041942], actual_y:[array([0])], loss:0.000000, acc:1.0
[ Epoch2000 / Step99 ] pred_y:[0.96521025], actual_y:[array([1])], loss:0.001205, acc:1.0
[ Epoch2000 / Step100 ] pred_y:[0.99998434], actual_y:[array([1])], loss:0.000000, acc:1.0
Accuracy: 100.0%
```

After 2000-epoch training, the accuracy for testing the linear dataset is 100%.

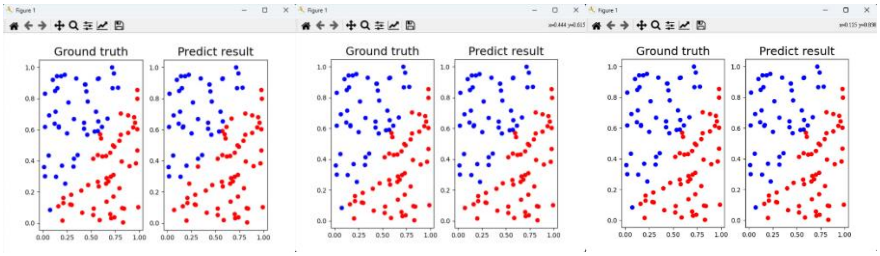
C. Learning curve (loss, epoch curve)



The picture on the leaf-hand side is the loss curve by each epoch.

The neural network converges after about 1000th epoch.

D. Anything you want to present



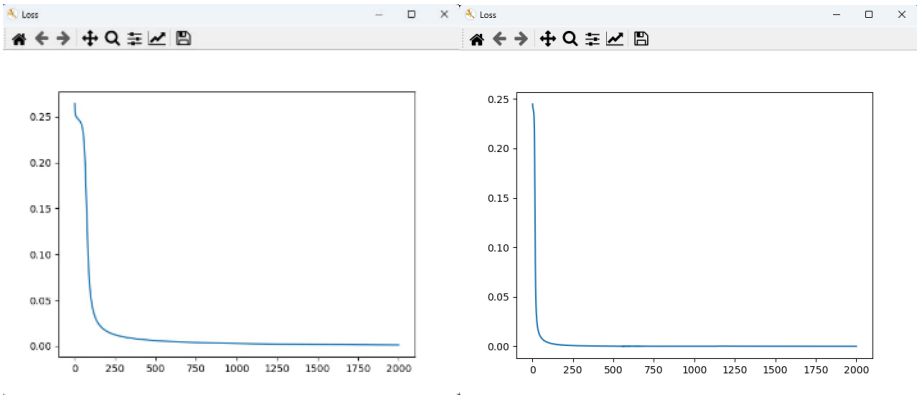
70 epochs
Acc 94%

100 epochs
Acc 99%

2000 epochs
Acc 100%

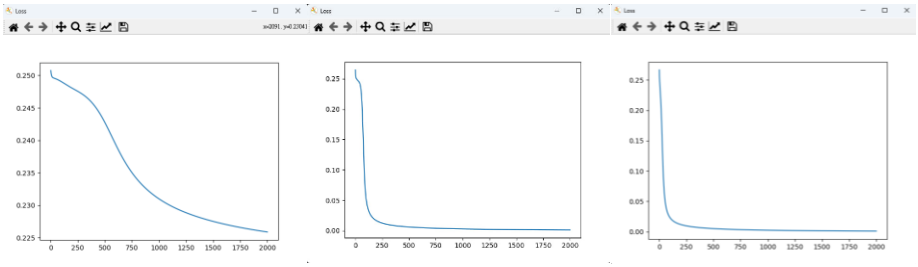
4. Discussion

A. Try different learning rates



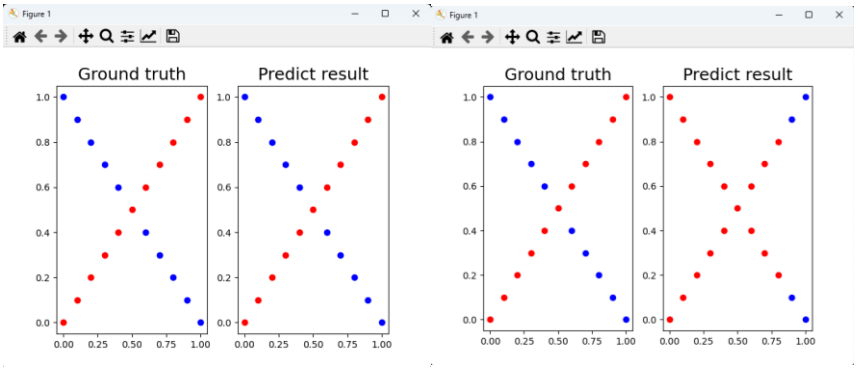
Picture on the left-hand side has learning rate = 0.1, and compares to the picture on the right-hand side which has learning rate = 0.5, the picture on the right-hand side converges earlier.

B. Try different numbers of hidden units



Pictures from left to right above have neuron numbers [2,1,1,1], [2,3,3,1], [2,6,6,1], respectively. If the number of neurons is not enough, then underfitting may occurs.

C. Try without activation functions



Picture on the left-hand side with sigmoid function, and picture on the right-hand side with no activation function. Neural network with activation function can deal with complex data, and which without activation can only solve linear problem.