Report
0616237 廖俊崴

Data structure:

```
class neuron_node:
    def __init__(self):
        self.weights=[]
        self.bias=1.0
        self.a_z=None
        self.input=[]
        self.output=-1.0
        self.weight_deriv=[]
        self.sig_deriv=0.0
        self.errfun_deriv=0.0
        self.forward_coefficient=[]
```

我自定義了一個 structure neuron_node 來實作 nruron 我的 neuron_network 在 code 叫 Network 它包含了 2 個叫 layer 的 list，第一層 Layer 有 2 個 neurons，第二層有 1 個 neuron，它除了 weight 和 bias 還儲存了一些東西，像是現在的 input 跟現在 sigmoid_deriv 的值是多少。

Forward propagation:

```
def forward_pass(network):

    for index1 in range(len(network)):
        for ind2 in range(len(network[index1])):
            for i in range(len(network[index1][ind2].input)):
                network[index1][ind2].forward_coefficient[i]=network[index1][ind2].input[i]
            network[index1][ind2].forward_coefficient[-1]=1

    return
```

主要動作把 weight function 對各係數的微分記錄在 forwar_cefficient 這個 neuron 的 list 裡。

Sigmoid function:

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

def sigmoid_deriv(x):
    return sigmoid(x)*(1.0-sigmoid(x))
```

參考講義，基本我沒動到它。

Back propagation:

```
def backward_pass(network,label):

    network[1][0].errfun_deriv=network[1][0].output-label

    network[0][0].errfun_deriv =network[1][0].errfun_deriv*network[1][0].weights[0]*network[1][0].sig_deriv
    network[0][1].errfun_deriv =network[1][0].errfun_deriv*network[1][0].weights[1]*network[1][0].sig_deriv

    return
```

從後面開始算每個 neuron 對 error function 的微分，並記錄在 errfun_deriv 裡。

Update weights:

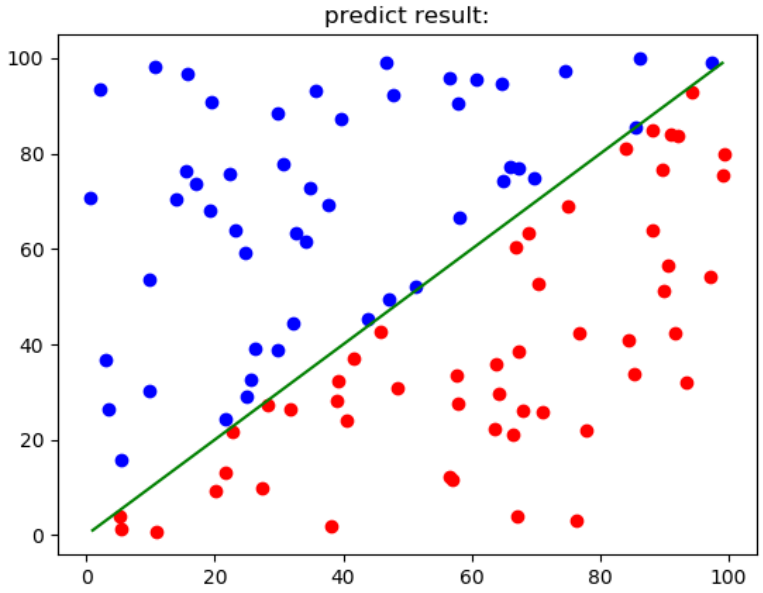
```
    weight_deriv_add: weight_deriv_add
def weight_deriv_add(network):

    for i in range(len(network)):
        for j in range(len(network[i])):

            for k in range(len(network[i][j].weights)):

                network[i][j].weight_deriv[k] += network[i][j].input[k]*network[i][j].errfun_deriv*network[i][j].sig_deriv

            network[i][j].weight_deriv[-1] += network[i][j].errfun_deriv*network[i][j].sig_deriv

    return

def update_weight(network,learn_rate):

    for i in range(len(network)):
        for j in range(len(network[i])):
            for k in range(len(network[i][j].weights)):
                network[i][j].weights[k] -= learn_rate*network[i][j].weight_deriv[k]
                #print(network[i][j].weights[k])
            network[i][j].bias -= learn_rate*network[i][j].weight_deriv[-1]

    return
```

Weight_deriv_add 會再跑資料迴圈時，讓每個 neuron 的 weight_derivate 加上這次針對這筆資料計算出來的 gradient。

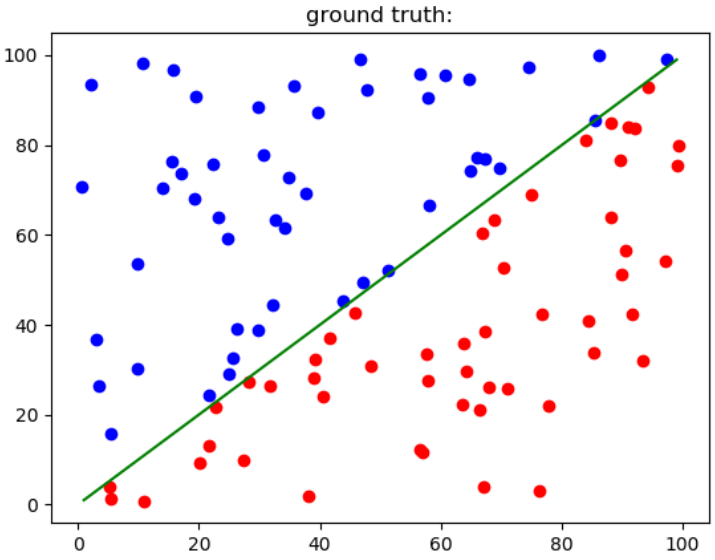
而下面的 update_weight，它是在跑完資料迴圈呼叫，是真正更新每個 neuron 的 weights 跟 bias，剪掉學習率乘上我們已經算好的 gradient。

predictions and ground truth:

prediction:



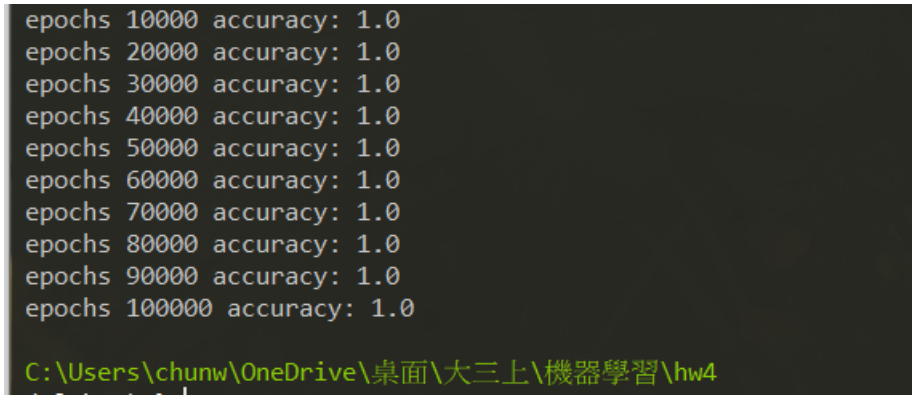
Ground truth:



Loss

```
epochs  10000  loss:  0.004187077821104303
epochs  20000  loss:  0.0020304228831181563
epochs  30000  loss:  0.001339037130869147
epochs  40000  loss:  0.000998608936746764
epochs  50000  loss:  0.0007960793374332553
epochs  60000  loss:  0.0006617960218547813
epochs  70000  loss:  0.0005662475733234758
epochs  80000  loss:  0.0004947913772050133
epochs  90000  loss:  0.0004393379727024538
epochs  100000  loss:  0.00039505435738677755
```

accuracy.


epochs 10000 accuracy: 1.0
epochs 20000 accuracy: 1.0
epochs 30000 accuracy: 1.0
epochs 40000 accuracy: 1.0
epochs 50000 accuracy: 1.0
epochs 60000 accuracy: 1.0
epochs 70000 accuracy: 1.0
epochs 80000 accuracy: 1.0
epochs 90000 accuracy: 1.0
epochs 100000 accuracy: 1.0
C:\Users\chunw\OneDrive\桌面\大三上\機器學習\hw4