NCTU Spring 2020 Deep Learning and Practice

Lab #1

0616015 劉姿利

Table of Content

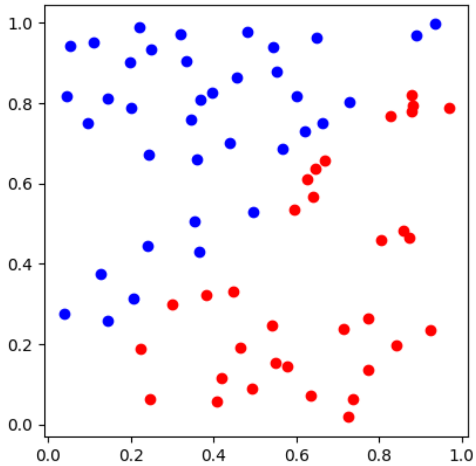
1 Introduction

1.1 Task

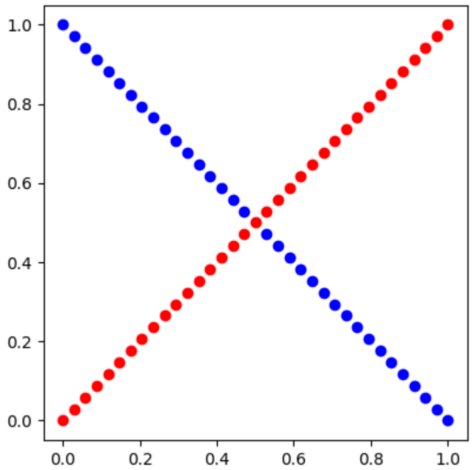
使用 numpy 實作一個 neural network，求 gradient 的部份要使用 back-propagation。並使用 XOR 和 Linear 的資料進行測試。

1.2 Default Datasets

Linear



XOR



2 Experimental Setup

2.1 Sigmoid Functions

Sigmoid

```python
def sigmoid(x):
   return 1 / (1 + np.exp(-x))
```

Derivative of sigmoid

```python
def der_sigmoid(y):
   return y * (1 - y)
```

其中 y 為 sigmoid layer 的 output

2.2 Neural Network

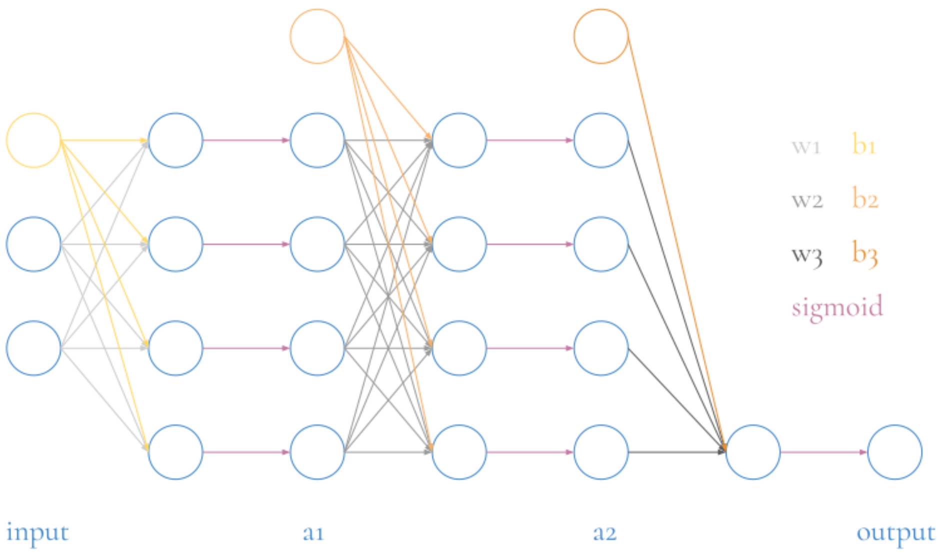
Network initialization code

```python
class SimpleNet:
   def __init__(self, hidden_size, num_step=2000, print_interval=100, lr=0.2):
       self.num_step = num_step
       self.print_interval = print_interval

       # Model parameters initialization
       input_size = 2
       output_size = 1
       self.lr = lr
       self.mo = 0.9
       self.w1 = np.random.randn(input_size, hidden_size)
       self.w2 = np.random.randn(hidden_size, hidden_size)
       self.w3 = np.random.randn(hidden_size, output_size)
       self.b1 = np.zeros((1, hidden_size))
       self.b2 = np.zeros((1, hidden_size))
       self.b3 = np.zeros((1, output_size))

       # Model parameters for momentum
       self.v_w1 = np.zeros((input_size, hidden_size) )
       self.v_w2 = np.zeros((hidden_size, hidden_size))
       self.v_w3 = np.zeros((hidden_size, output_size))
       self.v_b1 = np.zeros((1, hidden_size))
       self.v_b2 = np.zeros((1, hidden_size))
       self.v_b3 = np.zeros((1, hidden_size))
```

Network forwarding code

```python
class SimpleNet:
    def forward(self, inputs):
        self.input = inputs
        self.a1     = sigmoid(np.dot(self.input, self.w1) + self.b1)
        self.a2     = sigmoid(np.dot(self.a1, self.w2) + self.b2)
        output      = sigmoid(np.dot(self.a2, self.w3) + self.b3)
        return output
```

Network figure

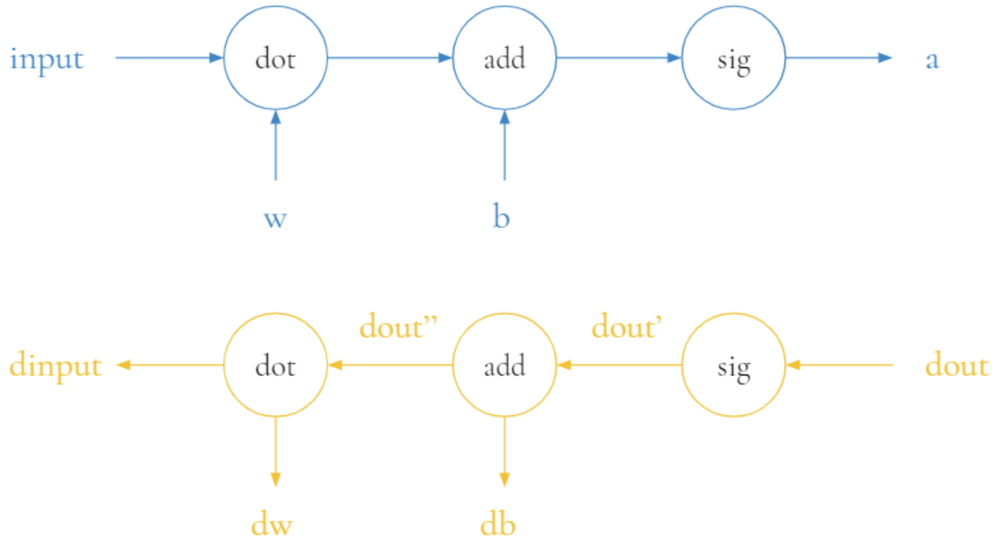


2.3 Back-propagation

Back-propagation code

```
class SimpleNet:
      def backward(self):

            # w3
            dout    = self.error
            dout    = np.multiply(dout, der_sigmoid(self.output))
            grad_w3 = np.dot(self.a2.T, dout)
            grad_b3 = np.sum(dout, axis=0)

            # w2
            dout    = np.dot(dout, self.w3.T)
            dout    = np.multiply(dout, der_sigmoid(self.a2))
            grad_w2 = np.dot(self.a1.T, dout)
            grad_b2 = np.sum(dout, axis=0)

            # w1
            dout    = np.dot(dout, self.w2.T)
            dout    = np.multiply(dout, der_sigmoid(self.a1))
            grad_w1 = np.dot(self.input.T, dout)
            grad_b1 = np.sum(dout, axis=0)

            # momentum
            self.v_w1 = self.mo * self.v_w1 + self.lr * grad_w1
            self.v_w2 = self.mo * self.v_w2 + self.lr * grad_w2
            self.v_w3 = self.mo * self.v_w3 + self.lr * grad_w3

            self.v_b1 = self.mo * self.v_b1 + self.lr * grad_b1
            self.v_b2 = self.mo * self.v_b2 + self.lr * grad_b2
            self.v_b3 = self.mo * self.v_b3 + self.lr * grad_b3

            # updating
            self.w1 -= self.v_w1
            self.w2 -= self.v_w2
            self.w3 -= self.v_w3

            self.b1 -= self.v_b1
            self.b2 -= self.v_b2
            self.b3 -= self.v_b3

            return
```

Back-propagation of one layer



dout' = np.multiply(dout, der_sigmoid(a))

dout" = dout'

db = np.sum(dout', axis=0)

dw = np.dot(input.T, dout")

dinput = np.dot(dout", w.T)

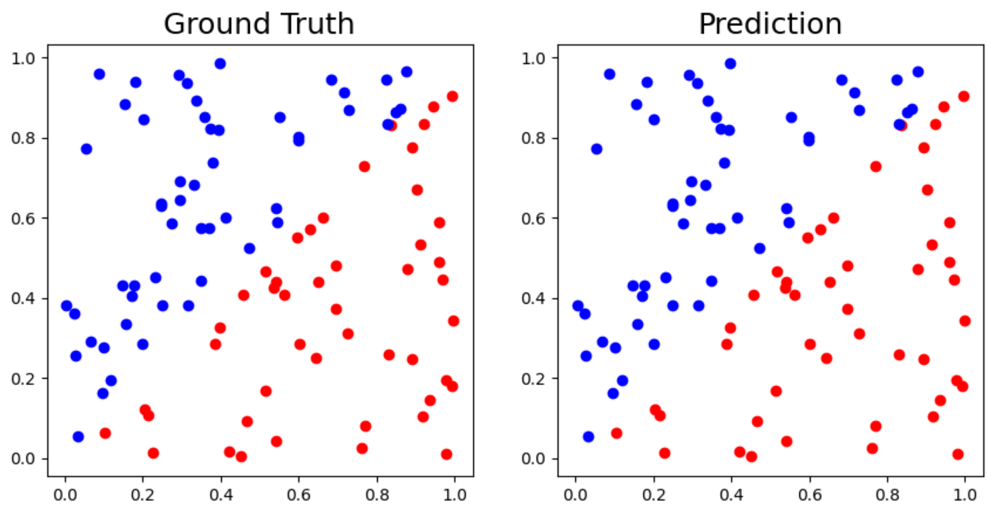
Updating

parameter -= learning_rate * der_parameter

3 Experimental Results

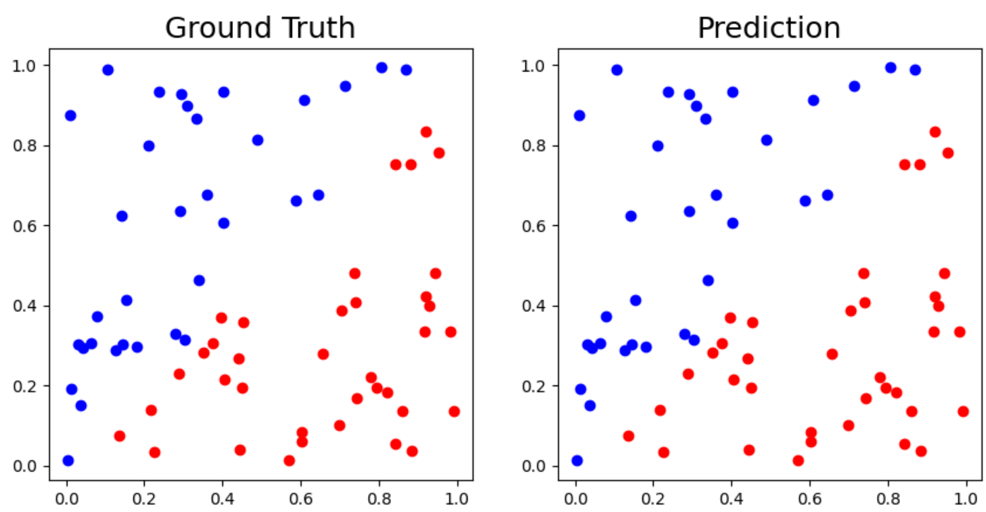
3.1 Screenshots and Comparison Figures

Linear - training set



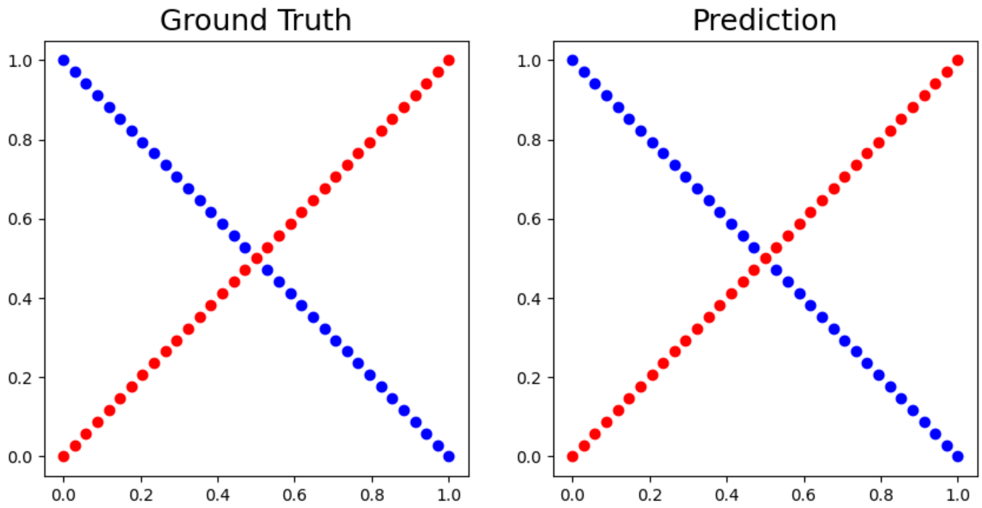
# of points = 100
hidden_size = 4
learning_rate = 0.1
epoch = 100
accuracy = 100%

Linear - testing set



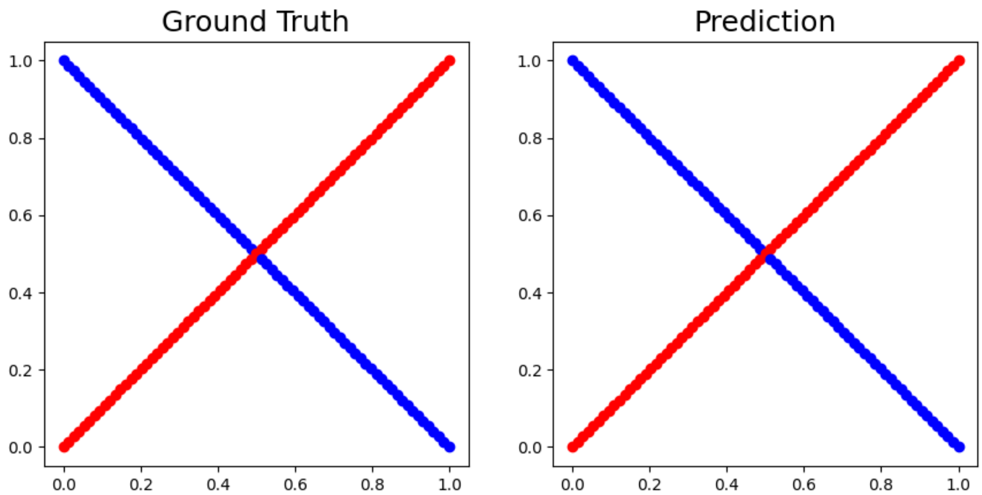
accuracy = 100%

XOR - training set



# of points = 100
hidden_size = 20
learning_rate = 0.1
epoch = 300
accuracy = 100%

XOR - testing set



# of points = 150
accuracy = 100%

3.2 Results with and without Momentum

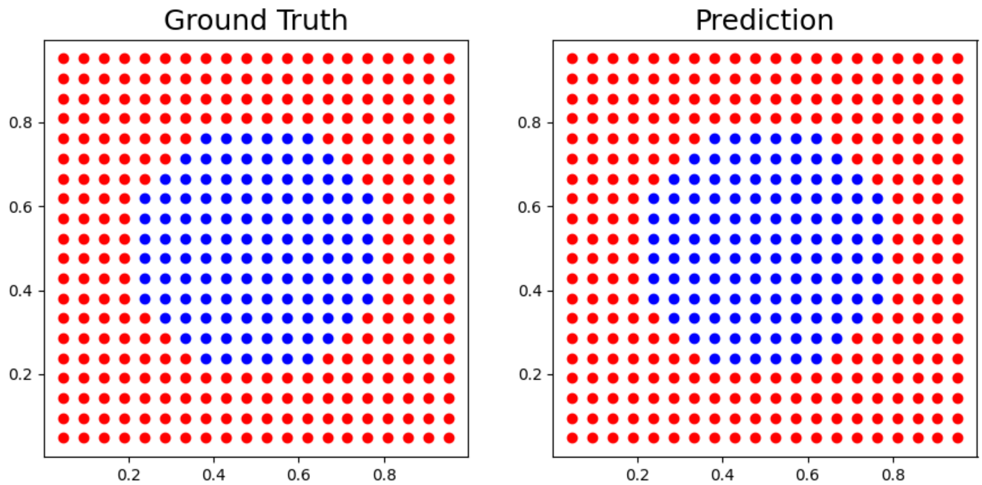
of epoches to achieve accuracy 100%

	Linear	XOR
with momentum	100	300
without momentum	100	3700

4 Discussion and Extra Experiments

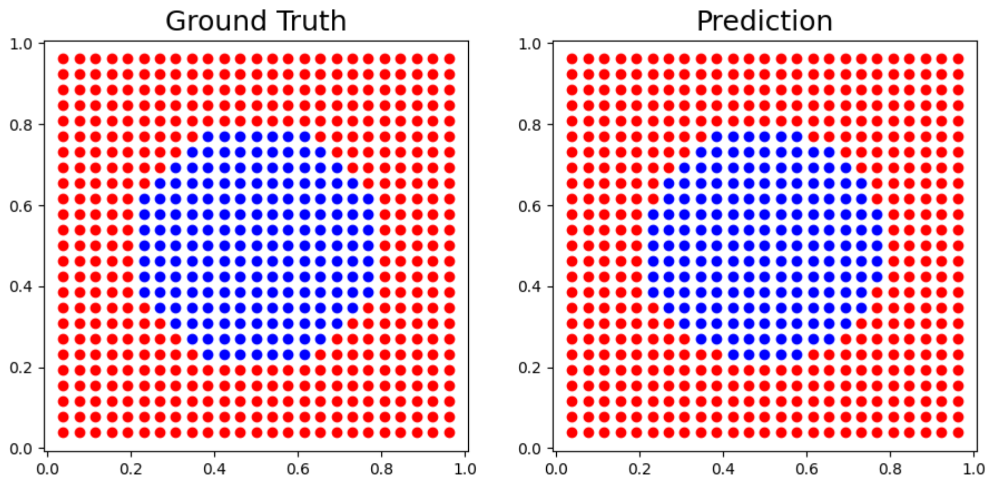
4.1 Extra Experiment - Circle

Circle - training set



# of points = 400
hidden_size = 20
learning_rate = 0.1
epoch = 1000
accuracy = 100%

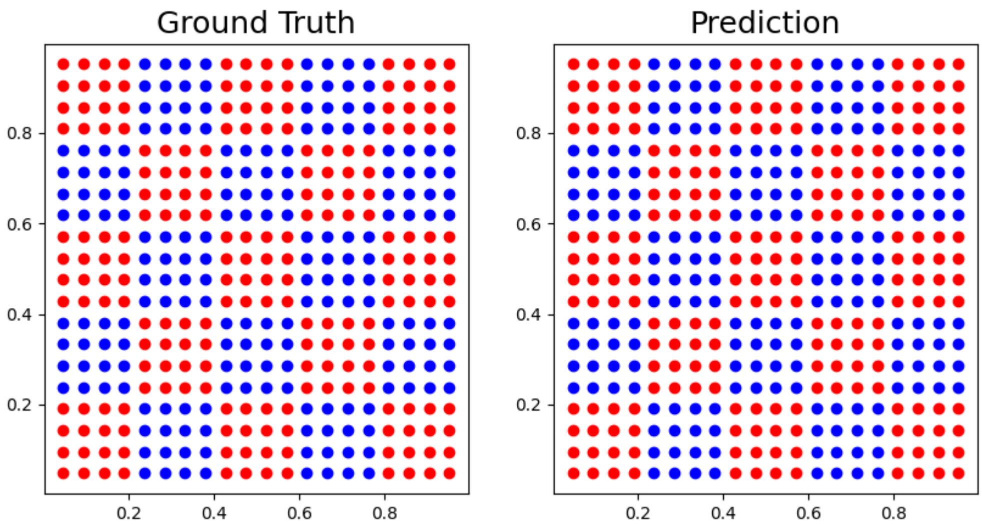
Circle - testing set



# of points = 625
accuracy = 99.04%

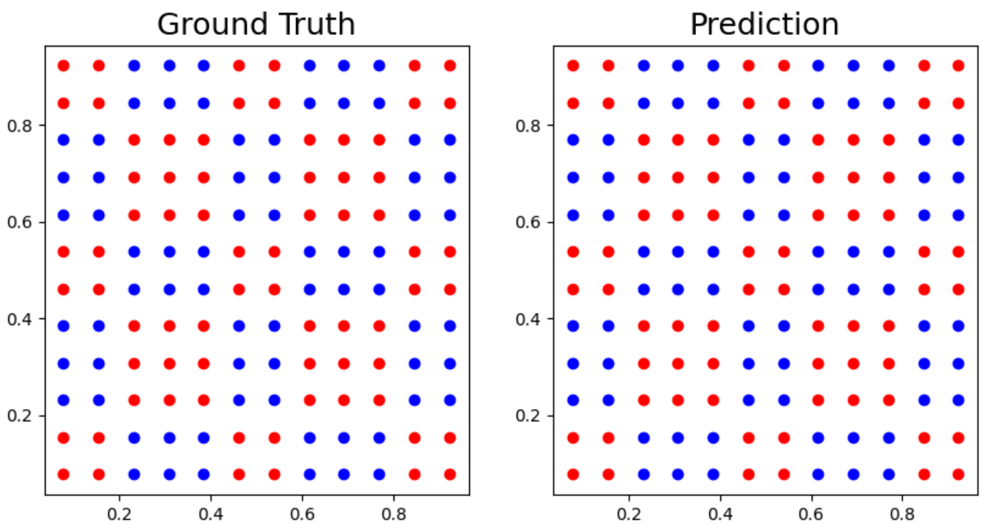
4.2 Extra Experiment - Grid

Grid - training set



# of points = 400
hidden_size = 20
learning_rate = 0.2
epoch = 5000
accuracy = 100%

Grid - testing set



# of points = 144
accuracy = 100%

4.3 Discussion

在這個 lab 裡面我的第一個版本是很簡單的 back-propagation，只有求 gradient 然後更新，但是這樣子 training 的效率非常低，所以加了 momentum，加了之後有好一點（見 section 3.2 Results with and without Momentum），但如果想要更好的話覺得可以往調整 output 格式還有換 loss function 的方向嘗試。