Lab1-Backpropagation

張千祐
Department of Computer Science
National Yang Ming Chiao Tung University
qianyou.cs11@nycu.edu.tw

1 Introduction

在這次 lab 中，目標是希望不藉由 python 中現有的 pytorch 或 tensorflow 等框架，只用 numpy 和其他標準套件，實作出簡單的 neural network 以及相關功能，包含 forward propagation 及 back propagation 等，希望能夠分類出簡單的線性和 XOR 的資料，報告呈現方式會包含視覺化結果、繪製學習曲線，並討論在不同 activation function 及 optimizer 時的表現結果。

2 Experiment setups

2.1 Sigmoid functions

Sigmoid function 在神經網路當中扮演 activation function 的角色，目的在提供非線性的轉換，使得神經網路具有擬合非線性資料的能力，用 sigmoid 作爲 activation function 的優點包含 sigmoid 在任意實數上都存在微分值，以及接近收斂時能夠穩定的收斂到最佳解或局部最佳解，缺點則是計算花費相對高。

$$\sigma(x) = \frac{1}{1+e^{-x}}$$
$$\sigma'(x) = \sigma(x)\cdot(1-\sigma(x))$$

2.2 Neural network

Neural network	
Input dim	2
Hidden layer	2
Hidden dim	[10, 10]
Output dim	1

Table 1: Architecture of the neural network

2.3 Backpropagation

我們知道每一層 hidden layer 皆是進行如下運算

$$X' = XW_i$$
$$Z_i = \sigma(X')$$

因此若要找出對的方向更新 weights，使得 loss 最小，我們必須計算出 $\frac{\partial L}{\partial W_i}$，根據 chain rule，我們知道

$$\frac{\partial L}{\partial W_i} = \frac{\partial X'}{\partial W_i}\frac{\partial Z_i}{\partial X'}\frac{\partial L}{\partial Z_i}$$

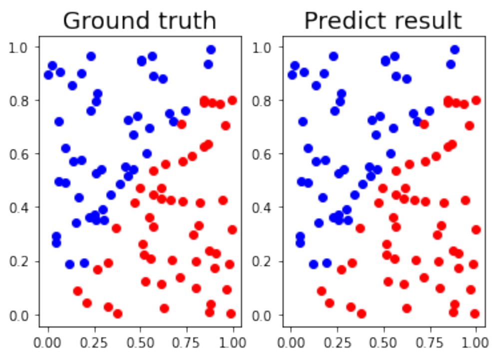
若 X 爲上一層 hidden layer 的 output，也就是

$$X = \sigma(KW_{i-1})$$

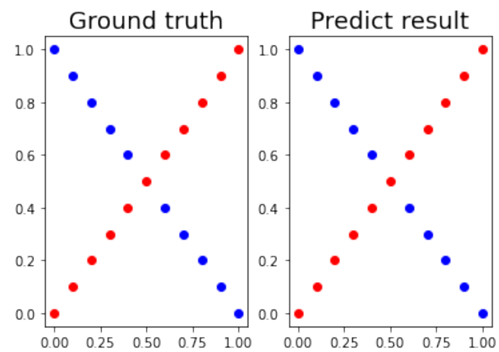
那麼在計算 $\frac{\partial L}{\partial W_i}$ 的同時，也必須把 $\frac{\partial L}{\partial X}$ 的資訊往前傳遞，以利於計算 $\frac{\partial L}{\partial W_{i-1}}$，因此 gradient 傳遞方向是由後面的 layer 往前面的 layer 傳遞，這整個過程稱爲 backpropagation。

3 Result

3.1 Screenshot and comparison figure



(a) Linear data



(b) Xor data

Figure 1: Screenshot of comparison result

3.2 Accuracy

3.2.1 Linear data

```
Case 0  | Ground truth: 1 | Prediction: 0.90731
Case 1  | Ground truth: 0 | Prediction: 0.15723
Case 2  | Ground truth: 1 | Prediction: 0.98613
Case 3  | Ground truth: 0 | Prediction: 0.01071
Case 4  | Ground truth: 0 | Prediction: 0.04441
Case 5  | Ground truth: 1 | Prediction: 0.98300
Case 6  | Ground truth: 0 | Prediction: 0.03666
Case 7  | Ground truth: 0 | Prediction: 0.17310
Case 8  | Ground truth: 0 | Prediction: 0.03529
Case 9  | Ground truth: 0 | Prediction: 0.15278
Case 10 | Ground truth: 0 | Prediction: 0.09943
Case 11 | Ground truth: 1 | Prediction: 0.54573
Case 12 | Ground truth: 0 | Prediction: 0.48933
Case 13 | Ground truth: 1 | Prediction: 0.85052
Case 14 | Ground truth: 1 | Prediction: 0.84826
Case 15 | Ground truth: 1 | Prediction: 0.95503
Case 16 | Ground truth: 1 | Prediction: 0.82661
Case 17 | Ground truth: 0 | Prediction: 0.01406
Case 18 | Ground truth: 1 | Prediction: 0.98347
Case 19 | Ground truth: 1 | Prediction: 0.96104
Case 20 | Ground truth: 0 | Prediction: 0.11160
Case 21 | Ground truth: 1 | Prediction: 0.98362
Case 22 | Ground truth: 1 | Prediction: 0.98280
Case 23 | Ground truth: 0 | Prediction: 0.01203
Case 24 | Ground truth: 0 | Prediction: 0.01573
Case 25 | Ground truth: 1 | Prediction: 0.98615
Case 26 | Ground truth: 0 | Prediction: 0.15594
Case 27 | Ground truth: 0 | Prediction: 0.05772
Case 28 | Ground truth: 1 | Prediction: 0.96074
```

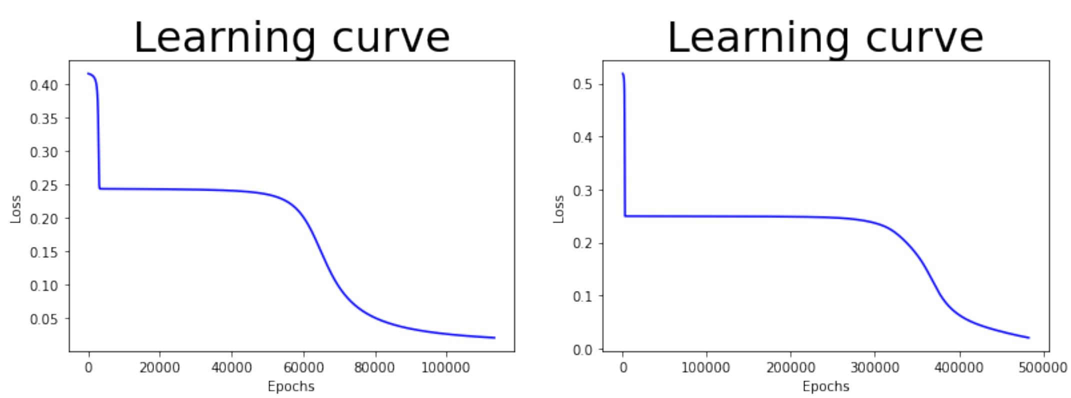
```
Case 29 | Ground truth: 0 | Prediction: 0.01033
Case 30 | Ground truth: 0 | Prediction: 0.01396
Case 31 | Ground truth: 0 | Prediction: 0.14409
Case 32 | Ground truth: 1 | Prediction: 0.98437
Case 33 | Ground truth: 1 | Prediction: 0.96486
Case 34 | Ground truth: 1 | Prediction: 0.98239
Case 35 | Ground truth: 1 | Prediction: 0.88327
Case 36 | Ground truth: 0 | Prediction: 0.02484
Case 37 | Ground truth: 0 | Prediction: 0.01118
Case 38 | Ground truth: 0 | Prediction: 0.04993
Case 39 | Ground truth: 1 | Prediction: 0.95840
Case 40 | Ground truth: 0 | Prediction: 0.01590
Case 41 | Ground truth: 1 | Prediction: 0.98657
Case 42 | Ground truth: 1 | Prediction: 0.98603
Case 43 | Ground truth: 1 | Prediction: 0.98769
Case 44 | Ground truth: 0 | Prediction: 0.05439
Case 45 | Ground truth: 1 | Prediction: 0.98815
Case 46 | Ground truth: 1 | Prediction: 0.98789
Case 47 | Ground truth: 0 | Prediction: 0.46322
Case 48 | Ground truth: 0 | Prediction: 0.05547
Case 49 | Ground truth: 0 | Prediction: 0.13987
Case 50 | Ground truth: 0 | Prediction: 0.05737
Case 51 | Ground truth: 1 | Prediction: 0.97522
Case 52 | Ground truth: 1 | Prediction: 0.97388
Case 53 | Ground truth: 1 | Prediction: 0.98567
Case 54 | Ground truth: 1 | Prediction: 0.98615
Case 55 | Ground truth: 1 | Prediction: 0.98734
Case 56 | Ground truth: 1 | Prediction: 0.98325
Case 57 | Ground truth: 1 | Prediction: 0.98112
Case 58 | Ground truth: 0 | Prediction: 0.01737
Case 59 | Ground truth: 1 | Prediction: 0.98736
Case 60 | Ground truth: 1 | Prediction: 0.98617
Case 61 | Ground truth: 0 | Prediction: 0.01130
Case 62 | Ground truth: 1 | Prediction: 0.98719
Case 63 | Ground truth: 0 | Prediction: 0.01026
Case 64 | Ground truth: 0 | Prediction: 0.02591
Case 65 | Ground truth: 1 | Prediction: 0.94710
Case 66 | Ground truth: 1 | Prediction: 0.98599
Case 67 | Ground truth: 1 | Prediction: 0.98255
Case 68 | Ground truth: 0 | Prediction: 0.30132
Case 69 | Ground truth: 1 | Prediction: 0.98789
Case 70 | Ground truth: 0 | Prediction: 0.08066
Case 71 | Ground truth: 1 | Prediction: 0.68137
Case 72 | Ground truth: 1 | Prediction: 0.98701
Case 73 | Ground truth: 0 | Prediction: 0.32869
Case 74 | Ground truth: 1 | Prediction: 0.55379
Case 75 | Ground truth: 1 | Prediction: 0.98476
Case 76 | Ground truth: 1 | Prediction: 0.98745
Case 77 | Ground truth: 1 | Prediction: 0.88091
Case 78 | Ground truth: 1 | Prediction: 0.98454
Case 79 | Ground truth: 0 | Prediction: 0.26657
Case 80 | Ground truth: 0 | Prediction: 0.01191
Case 81 | Ground truth: 1 | Prediction: 0.98585
Case 82 | Ground truth: 1 | Prediction: 0.98758
Case 83 | Ground truth: 0 | Prediction: 0.03468
Case 84 | Ground truth: 0 | Prediction: 0.21898
Case 85 | Ground truth: 1 | Prediction: 0.92139
Case 86 | Ground truth: 1 | Prediction: 0.80771
Case 87 | Ground truth: 1 | Prediction: 0.92797
Case 88 | Ground truth: 1 | Prediction: 0.92768
Case 89 | Ground truth: 0 | Prediction: 0.02432
Case 90 | Ground truth: 0 | Prediction: 0.12366
Case 91 | Ground truth: 1 | Prediction: 0.98371
Case 92 | Ground truth: 1 | Prediction: 0.92868
Case 93 | Ground truth: 1 | Prediction: 0.73306
```

```
Case 94 | Ground truth: 1 | Prediction: 0.98411
Case 95 | Ground truth: 1 | Prediction: 0.98657
Case 96 | Ground truth: 1 | Prediction: 0.97990
Case 97 | Ground truth: 1 | Prediction: 0.77679
Case 98 | Ground truth: 1 | Prediction: 0.55551
Case 99 | Ground truth: 0 | Prediction: 0.02557
accuracy: 100.00%
```

3.2.2 Xor data

```
Case 0  | Ground truth: 0 | Prediction: 0.00089
Case 1  | Ground truth: 1 | Prediction: 0.99890
Case 2  | Ground truth: 0 | Prediction: 0.00475
Case 3  | Ground truth: 1 | Prediction: 0.99835
Case 4  | Ground truth: 0 | Prediction: 0.03320
Case 5  | Ground truth: 1 | Prediction: 0.99612
Case 6  | Ground truth: 0 | Prediction: 0.13486
Case 7  | Ground truth: 1 | Prediction: 0.97475
Case 8  | Ground truth: 0 | Prediction: 0.24227
Case 9  | Ground truth: 1 | Prediction: 0.64739
Case 10 | Ground truth: 0 | Prediction: 0.25026
Case 11 | Ground truth: 0 | Prediction: 0.18425
Case 12 | Ground truth: 1 | Prediction: 0.67892
Case 13 | Ground truth: 0 | Prediction: 0.10929
Case 14 | Ground truth: 1 | Prediction: 0.97784
Case 15 | Ground truth: 0 | Prediction: 0.05872
Case 16 | Ground truth: 1 | Prediction: 0.99339
Case 17 | Ground truth: 0 | Prediction: 0.03154
Case 18 | Ground truth: 1 | Prediction: 0.99430
Case 19 | Ground truth: 0 | Prediction: 0.01792
Case 20 | Ground truth: 1 | Prediction: 0.99301
accuracy: 100.00%
```

3.3 Learning curve



(a) Linear data

(b) Xor data

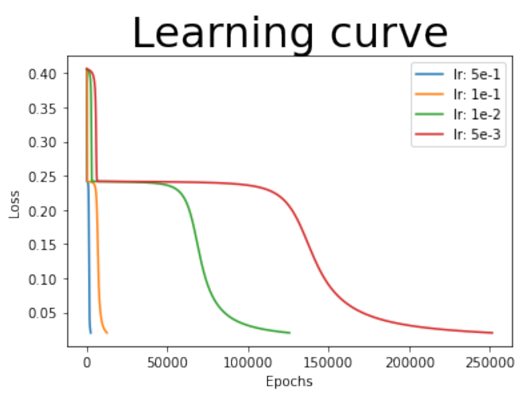
Figure 2: Learning curve

4 Discussion

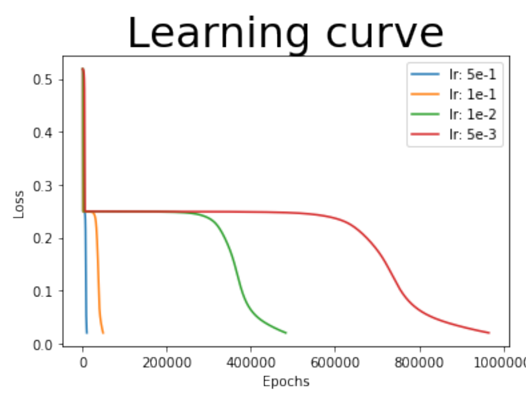
這邊我試圖從不同 hyperparameters 的設定，來比較各種設定下的表現情形

4.1 Different learning rate

從 Figure 3可以看出在四種 lr 的設定下，越大的 lr 收斂的速度越快，推測是因爲預測資料的複雜度都相對簡單，loss function 是相對平滑的函數，因此在大一點的 lr 設定下表現更好。若從資料類型來看，也可以看出 linear data 的收斂速度都比 xor data 來得更快。



(a) Linear data



(b) Xor data

Figure 3: Learning curve in different lr

4.2 Different numbers of hidden units

在固定 $lr = 5e-1$ 的情況下，我把最大 epochs 數設定爲 1e+6，且設定提前結束訓練的門檻爲 2e-2，之後試著改變 hidden units 來看收斂速度與準確度的表現，如 Table 2，可以看出在兩種 data 類型都是在 hidden units=10 時收斂最快，其中在 hidden unit 爲 2 時，xor data 在 model 達到最大 epochs 仍尚未收斂，使得無法全部預測正確。

Hidden units	Linear data		Xor data	
	# of epochs	Accuracy	# of epochs	Accuracy
2	4715	100.00%	1000000	85.71%
5	2765	100.00%	11621	100.00%
10	2516	100.00%	9658	100.00%

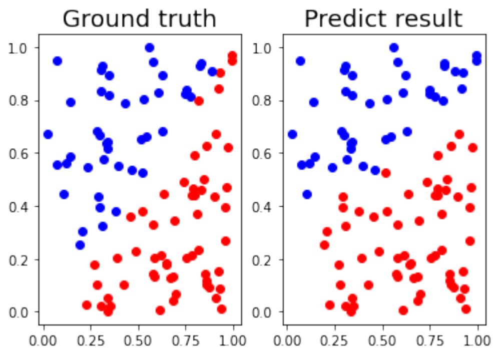
Table 2: Training epochs and accuracy in different hidden units

4.3 Without activation function

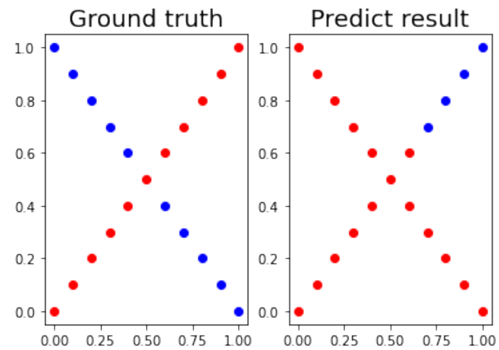
在這部分我固定 $lr = 1e-3$，想要比較若不使用 activation function 會有什麼結果，而結果也如同學到的那樣，如 Figure 4，在兩種 data 類型下的表現都不如有使用 activation function 的結果。因爲不使用 activation function 的因素，使得 model 只具有模擬線性資料的能力，在 linear data 中，因爲產生的訓練資料有一部份很接近分隔線，因此若只用線性來做分類，很容易因爲一點誤差導致無法全部預測正確；在 xor data 中，因爲資料本身就不是線性的資料，因此沒有 activation function 的 model 在這裡預測的能力來的比 linear data 更差。

5 Extra

這部分主要列出並說明了我嘗試實作不同 optimizer 和 activation function



(a) Accuracy: 88.00%



(b) Accuracy: 33.33%

Figure 4: Comparison without activation function

5.1 Implement different optimizers

在 optimizer 的部分，除了最平常的 gradient descent(GD)，我另外實作了同樣常被使用的 adam optimizer，他結合了 Momentum 和 AdaGrad 兩種方法，使得能夠得到更合適的 learning rate，讓收斂速度更快，從實作中可以看到 adam 需要更多 hyperparameters，如 β_1, β_2 等。

```
class optim:
    def __init__(self, parameters, args):
        self.optimizer = args.optimizer
        self.layers = parameters
        if self.optimizer == 'gd':
            self.lr = args.lr
        elif self.optimizer == 'adam':
            self.beta1 = args.beta1
            self.beta2 = args.beta2
            self.eps = args.eps
            self.lr = args.lr
            self.m = [0 for i in range(len(self.layers))]
            self.v = [0 for i in range(len(self.layers))]
            self.t = 1

    def step(self):
        for i in range(len(self.layers)):
            if self.optimizer == 'gd':
                self.layers[i].w -= self.layers[i].grad * self.lr
            elif self.optimizer == 'adam':
                self.m[i] = (self.beta1 * self.m[i] + (1 - self.beta1) * self.layers[i].grad)
                self.v[i] = (self.beta2 * self.v[i] + (1 - self.beta2) * (self.layers[i].grad**2))
                m_hat = self.m[i] / (1-self.beta1**self.t)
                v_hat = self.v[i] / (1-self.beta2**self.t)
                self.layers[i].w -= self.lr * (m_hat / (self.eps + np.sqrt(v_hat)))
        if self.optimizer == 'adam':
            self.t += 1
```

5.2 Implement different activation functions

這部分除了 sigmoid，我另外實作了 leakyReLU 和 tanh，同樣地，也必須一同求出它們各自的微分，如下

```
import numpy as np

```

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

def derivative_sigmoid(x):
    return np.multiply(x, 1.0-x)

def leakyrelu(x, alpha):
    return np.maximum(alpha*x, x)

def derivative_leakyrelu(x, alpha):
    return np.where(x>0, 1, alpha)

def tanh(x):
    return (np.exp(x) - np.exp(-x)) / (np.exp(x) + np.exp(-x))

def derivative_tanh(x):
    return 1 - x**2
```