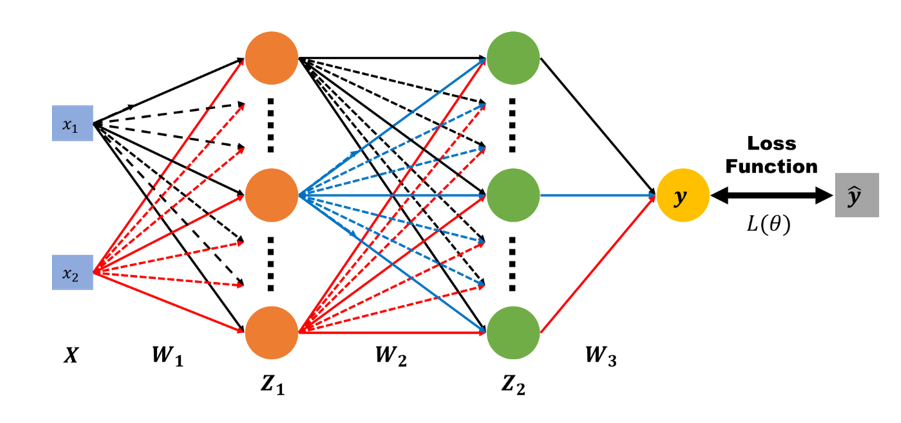
**Lab1: Back-propagation**

**系級:智能系統 學號:312581006 姓名:張宸瑋**

1. Introduction

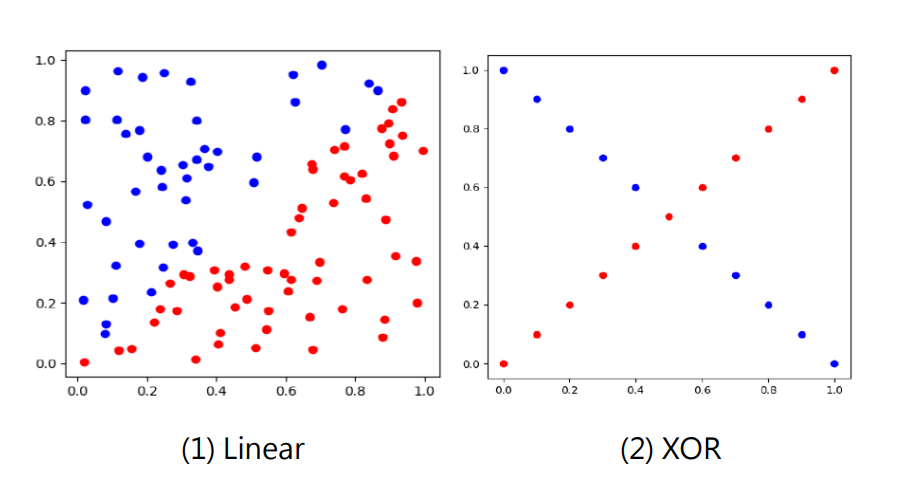


這次lab要實作出一個有兩層hidden layers 的 神經網路，並且分別對兩種二元分類問題進行訓練，將分類結果、訓練過程、預測結果等數據可視化，用來探討神經網路實作中的各項細節，包含back-propagation如何實作、如何利用chain-rule 來計算gradient 以及 使用不同optimizer和activation functions會帶來怎麼樣的結果

實作流程:

* + 1. 生成訓練用資料
    2. 初始化神經網路(初始化各層權重)
    3. 前向傳播（Forward Propagation）
    4. 計算損失函數
    5. 反向傳播（Backpropagation）
    6. 參數更新
    7. 重複訓練

實驗中使用的input data:



圖中點的顏色分別代表 Y = 0 or 1

1. Experiment setups
2. Sigmoid functions

Sigmoid 函數（也稱為 Logistic 函數）是一個常見的激勵函數，在深度學習和機器學習中廣泛使用。它將輸入值映射到介於0和1之間的範圍，通常用於處理二元分類問題或產生概率值。

|  |
| --- |
| Activation Functions for Deep Learning | by Mehmet Toprak | Medium |
| Sigmoid Function示意圖 |
|  |
| 程式實作部分 |

1. Neural networks

這次的實驗的神經網路架構主要是兩層hidden layer，並且每一層hidden layer都必須包含一個轉換以及激勵函數，透過兩層hidden layer(neuron數量為4)的運算後，到達output layer，透過運算出來的值，搭配我這次使用作為損失函數的MSE，來評估預測值以及實際值的誤差。

|  |
| --- |
|  |
| 神經網路架構 |
|  |
| 神經網路程式碼架構 |

1. Backpropagation

反向傳播（Backpropagation）是一種用於訓練神經網絡的優化算法，用來改善當參數量多的時候，傳統的gradient descent會遇到計算複雜的問題，反向傳播通過計算損失函數對於權重的梯度，逐步調整權重以最小化損失，使得預測輸出逼近真實標籤

|  |
| --- |
|  |
| 反向傳播程式碼架構 |

1. Result of testing
   1. Linear

|  |  |
| --- | --- |
|  |  |
| Predictions and Ground truth | Learning curve |
|  |  |
| Training loss | Testing prediction |
|  | |
| Accuracy of model’s prediction | |

* 1. XOR

|  |  |
| --- | --- |
|  |  |
| Predictions and Ground truth | Learning curve |
|  |  |
| Training loss | Testing prediction |
|  | |
| Accuracy of model’s prediction | |

討論:

在訓練線性資料集中，loss的收斂會來得比較快，而且在預設值的部分，線性資料集比較能預測接近0或1的答案。

1. Discussion
   1. 調整成不同的learning

Liner:

|  |
| --- |
|  |
|  |

XOR:

|  |
| --- |
|  |
|  |

討論:

這裡可以很明顯的看到，learning rate的調整是很重要的，如果learnging rate太大的話，可能會一直在最佳解附近徘徊，導致損失函數的曲線看起來一直震動，而如果學習率太小的話，曲線的收斂速度就會很慢，因為每一次的變動都很小，因此適當的學習綠是很重要的這樣才能透過適當的調整，並且更新參數。

* 1. 調整hidden layers 神經元個數

Liner

|  |
| --- |
|  |
|  |

XOR

|  |
| --- |
|  |
|  |

討論:

而在可以看到，當神經元個數增加的時候，很明顯的曲線的收斂速度比較快，而在XOR資料集中，可以看到神經元較少時，因為模型的表示能力會受到限制，因此對於這種非線性資料的表現會比較差，而因為神經元較多，模型的表示能力會增加，因此在這種非線性的資料集中，表現會較好，但是並不是神經元越大就越好，因為越大就代表著參數量會增加，運算速度可能會變慢，因此選擇合適的神經元是很重要的。

* 1. 激勵函數比較

Liner

|  |
| --- |
|  |
|  |

XOR:

|  |
| --- |
|  |
|  |

討論:

在這裡我們可以看到，在有沒有激勵函數的情況下，在線性資料集的表現上都差不多，我認為這是因為線性資料集中，只需要簡單的線性組合就能夠表示了，所以在表現上看起來都差不多，甚至還更快收斂，但是在非線性的資料集中，表現就非常差，因此適當的加入激勵函數，才能讓模型有更強大的表現能力。

1. Extra
   1. 實現不同優化器

Liner

|  |
| --- |
|  |
|  |

XOR

|  |
| --- |
|  |
|  |

討論:

這裡我是用了三種不同的優化器來做比較，第一個是傳統的Gradient Descent，第二個是Stochastic Gradient Descent，第三個是Min-Batch Stochastic Gradient Descent，而後面兩種算法都是為了改善傳統Gradient Descent每次更新時，都是使用全部的樣本來做梯度更新，在一些比較大的數據集中，這樣的做法會導致效能降低，而且引入大量的資料集也有可能造成內存不足的問題，而透過SGD每次更新時只會使用一個樣本來做更新，這樣做讓計算成本降低，但也很有可能會因為它的隨機性，導致它在最佳解中一直震盪，因此實現了第三種優化器算法MSGD，這是計算成本以及模型效能的折衷方案，使用小批次數據的梯度來更新模型參數。它介於 GD 和 SGD 之間，每次迭代中使用部分訓練樣本，這保持了部分樣本的隨機性，又減輕了計算成本。

# 在數據圖中，我們也可以看到，相對於gd來說，sgd跟msgd都更容易出現震盪，但是通常在傳統gd的作法，比較容易收斂到局部最佳解。

* 1. 實現不同激勵函數

Liner

|  |
| --- |
|  |
|  |

XOR

|  |
| --- |
|  |
|  |

討論:

在這裡我們可以看到sigmoid，在這兩個資料集中，表現最差，因為它的輸出範圍有限，但這裡比較讓我意外的是tahh的表現居然跟ReLU以及leeky ReLU差不是到太多，不太確定是不是權重初始化的問題，但這裡還是可以看到正常來說，ReLU跟leeky ReLU的表現還是會比較好，因為他們比較不會遭遇到梯度消失的問題，但在實驗的時候，發生一個插曲，訓練時，少數幾次會造成權重不是很明顯地在更新，應該說，可以當成沒更新，在查了資料才發現，這叫ReLU dead，這種情況通常發生在神經元的輸入小於或等於零的情況下，因為 ReLU 函數在輸入小於等於零時會返回零，導致神經元的權重不會更新，也無法再被激活，因此可能要再學習率跟權重初始化的設置中多加注意。