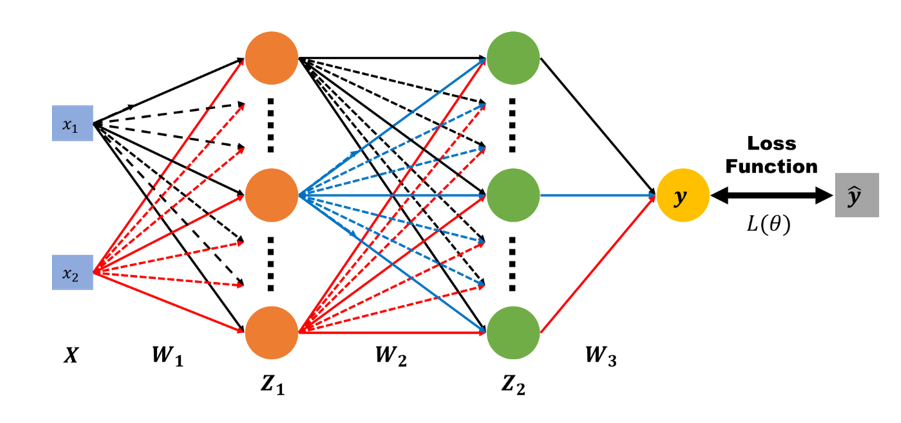
**Lab1: Back-propagation**

311511043 李承翰

1. Introduction

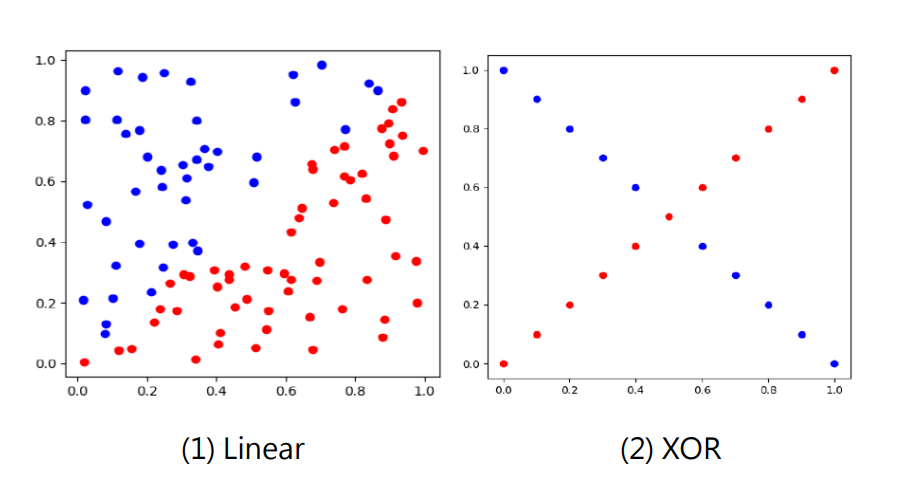


這次lab我們要實作出一個有兩層hidden layers 的 神經網路，並且分別對兩種二元分類問題進行訓練，將分類結果、訓練過程、預測結果等數據可視化，用來探討神經網路實作中的各項細節，包含back-propagation如何實作、如何利用chain-rule 來計算gradient 以及 使用不同optimizer和activation functions會帶來怎麼樣的結果

實作流程:

* 1. 初始化神經網路( 初始化各層權重及設定hyper-parameters)
  2. 生成訓練用資料
  3. Forward propagation 得出 prediction
  4. 計算誤差
  5. Back propagation
  6. 更新權重
  7. 重複直到設定的epoch 或者 loss、accuracy收斂

實驗中使用的input data:

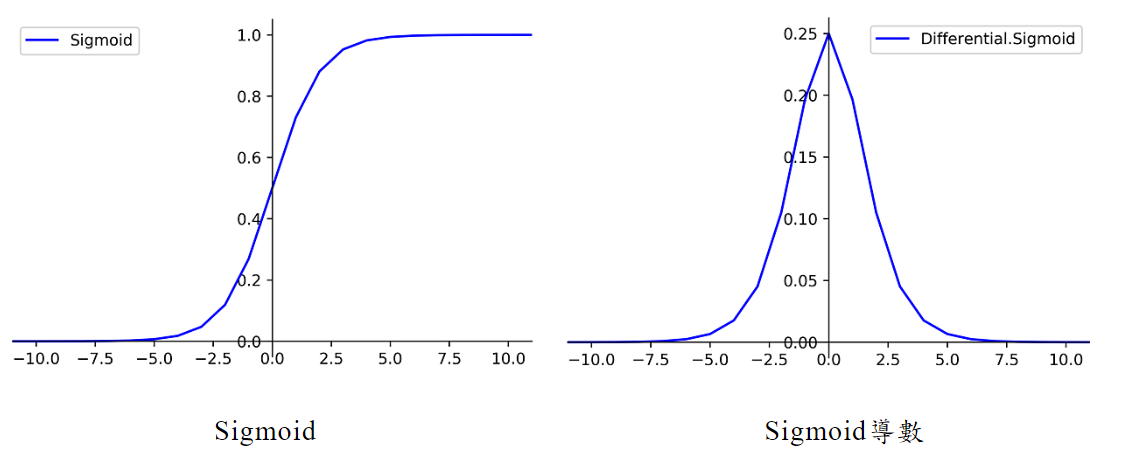


圖中點的顏色分別代表 Y = 0 、 1

1. Experiment setups
2. Sigmoid functions

Sigmoid functions 是深度學習領域開始時最常見的一種activation function，因為他是一個便於求導數的平滑函數

其方程式為: ， 其微分為:

**

從圖中我們可以發現因為此函數的導數最大值只有0.25，所以在實作當中如果有多層隱藏層則可能會有梯度受到多層函數導數的影響而造成梯度消失的問題。不過在此次實驗當中因為我們只有兩層隱藏層，所以相對而言較不會受到這個問題影響。

此外，因為函數當中牽涉到指數函數的運算，因此也比較消耗計算資源。

1. Neural networks

神經網路的特色是利用神經元(節點)來做非線性的特徵轉換，依賴訓練 資料來學習並隨著網路架構的深度提高其精確度。其運作方式為將輸入乘上 權重後，再經由激活函數輸出，一層一層反覆傳遞得到最後的結果。而訓練 的目標就是要出節點和節點之間的關係，也就是隱藏層的權重參數。這次lab由一個輸入層、兩個隱藏層以及一個輸出層來組成。

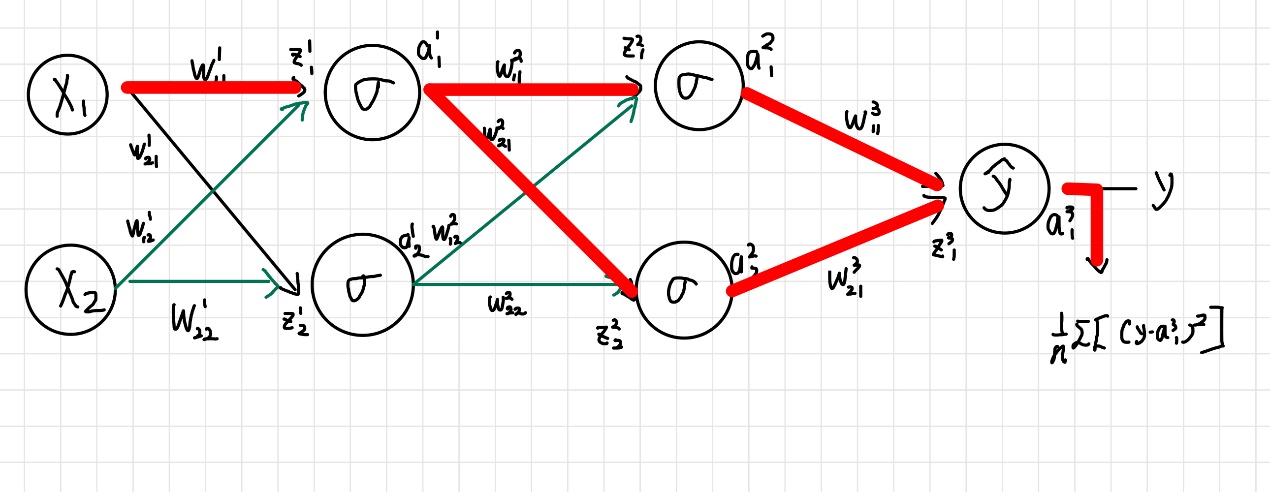
將輸入的input 乘上權重weight之後加總並加上一個bias 得到一個score，使用這個score作為sigmoid function的輸入來得到一個得到output。 從input層開始往下經歷了兩層隱藏層的運算後到達output層， 即為我們的神經網路對於input資料的預測結果，使用預測結果。使用此預測結果與誤差作為一個損失函數的Input，則可得到這一次預測的cost是多少。而在這次實驗當中我們使用MSE來作為我們的損失函數，接著我們就會透過梯度下降的方法來修正我們的權重，如此一來就可以逐步降低loss，使我們的預測結果愈發準確。

1. Backpropagation

在前面我們提到了梯度下降的方法來修正權重，就是引用了back propagation的概念，將誤差值作為回饋的資訊並且透過chain rule我們可以計算出loss對於每一個節點的梯度，接著我們可以算出節點中每一個權重的梯度利用這些梯度來更新我們的節點則可以使得誤差縮小。

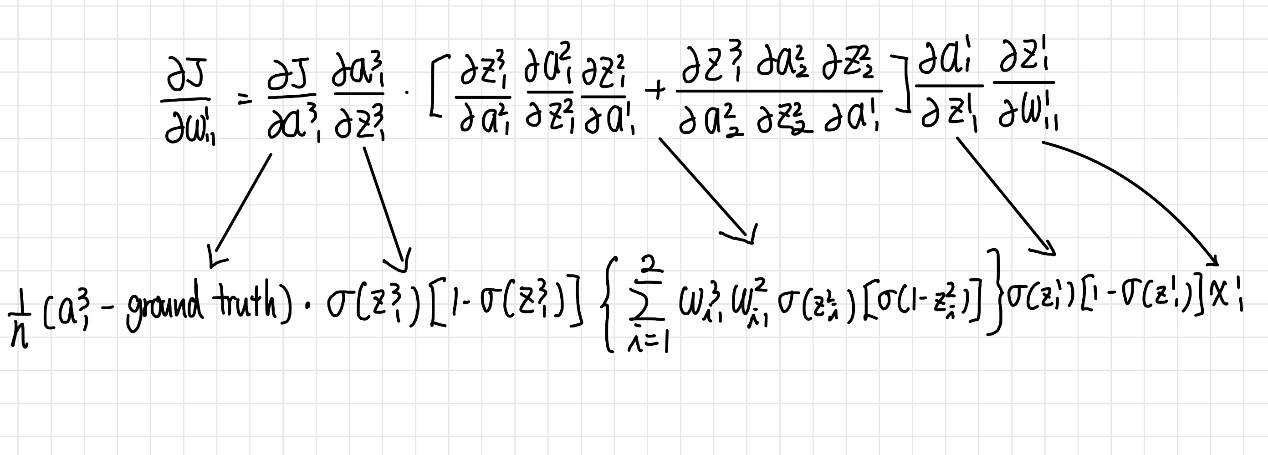
以此次的模型為例:

若要更新(第一層的第一個neuron 到第二層的第一個neuron的weight)

先列出此權重會影響的路徑(紅色路線):

之後列出在路徑中的方程式關係，即可得到 透過chain rule 展開

一張含有 文字 的圖片

自動產生的描述

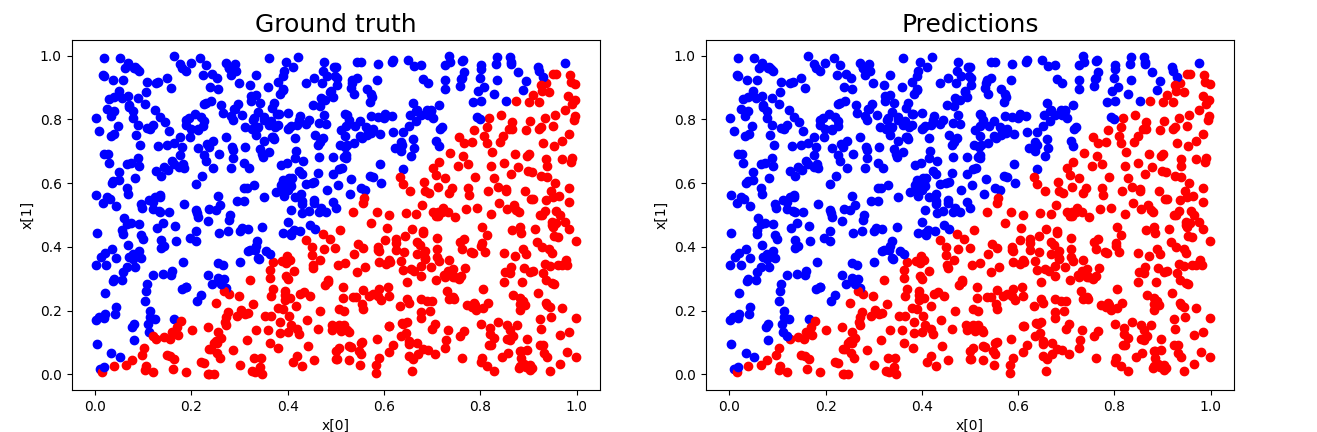
得到了 即可修正 的數值:

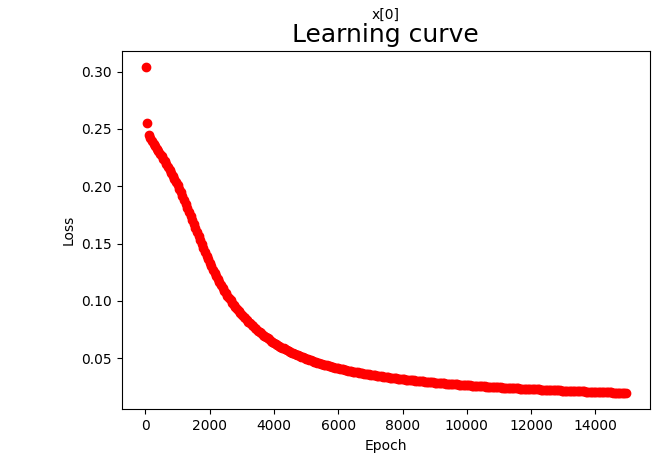
learning rate

以此類推即可修正所有的權重。

1. Result of testing
   1. Linear

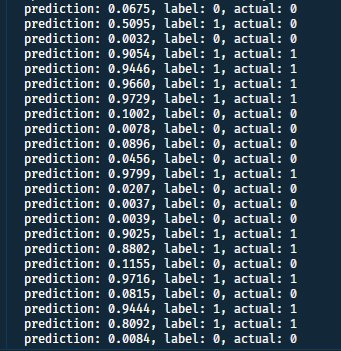
|  |  |  |  |
| --- | --- | --- | --- |
| epoch | Learning rate | Data size | 在隱藏層的neuron數量 |
| 15000 | 0.05 | 1000 | 4 |



一張含有 文字 的圖片

自動產生的描述

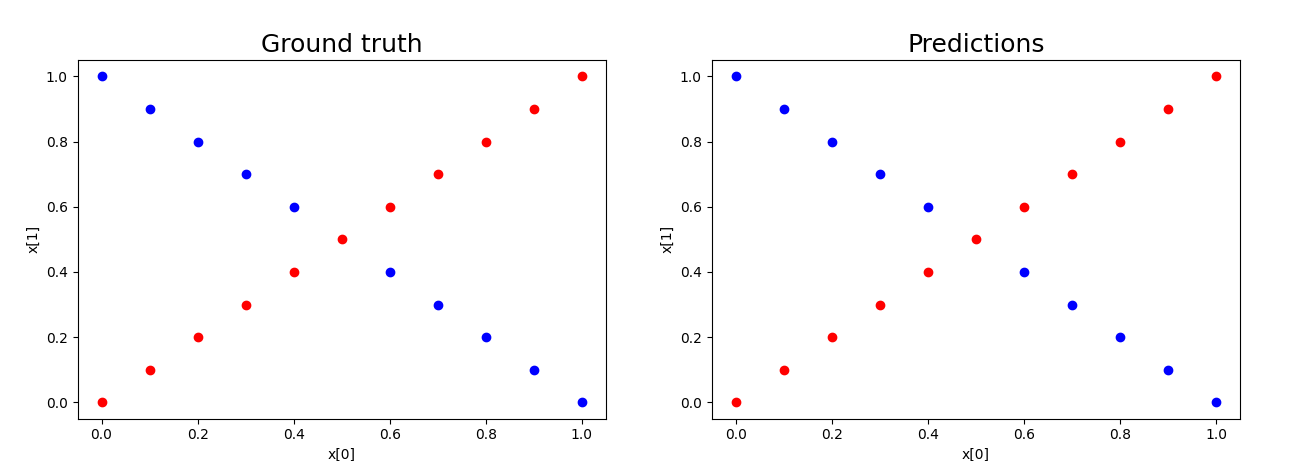
一張含有 文字 的圖片

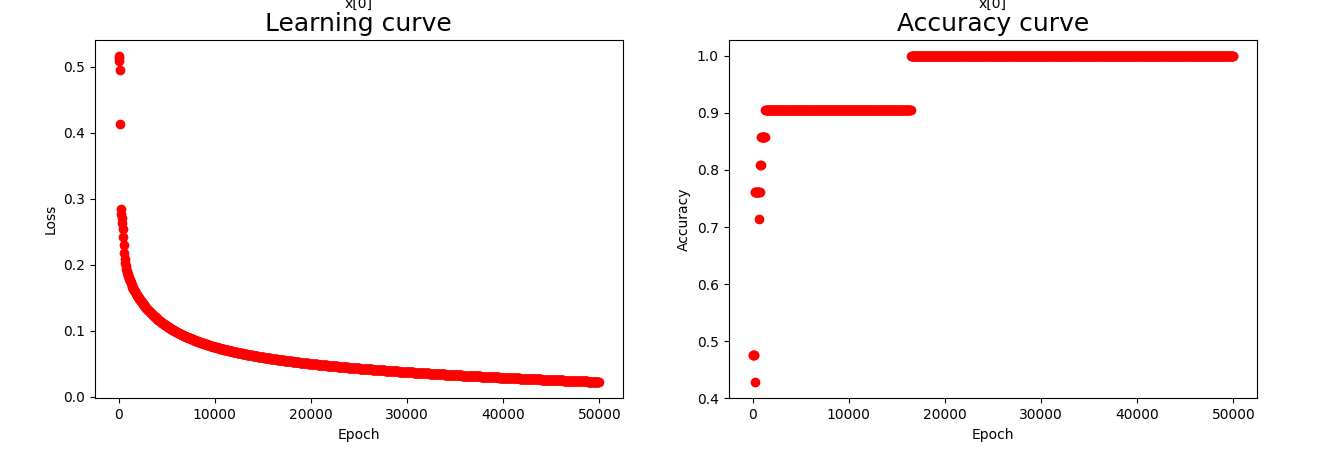
自動產生的描述

因為在我們的prediction當中我們得到的是不會是單純的0或1，所以我在判斷他的label的時候自行設定了一個threshold value，當prediction出來的結果>= 0.5時，則我將他的label設定為1，反之則將其設定為0。

* 1. XOR

|  |  |  |
| --- | --- | --- |
| epoch | Learning rate | 在隱藏層的neuron數量 |
| 30000 | 0.01 | 100 |





一張含有 文字 的圖片

自動產生的描述

1. Discussion
   1. 試試看不同的learning rate 對 Loss curve 的影響

Linear model: epoch為15000、Data amount為1000、隱藏層neuron數量為4

|  |  |  |
| --- | --- | --- |
| Learning rate | 0.05 | 0.5 |
| Learning curve |  |  |

在線性的模型當中我們可以發現無論learning rate大小，整個模型都有機會收斂到一個相對小的loss，不過兩者在收斂的速度上就有了不小的差距。

XOR model: epoch為30000、隱藏層neuron數量為100

|  |  |  |
| --- | --- | --- |
| Learning rate | 0.01 | 0.1 |
| Learning curve |  |  |

在XOR這個較複雜的非線性模型當中，我們可以看見在lr較小的時候，同樣出現了需要比較長的時間才收斂的結果，當我們將lr調大之後，可以看見收斂的速度明顯快了很多

|  |  |  |
| --- | --- | --- |
| Learning rate | 0.01 | 0.1 |
| Accuracy | 1.0000 | 1.0000 |

* 1. 在hidden layers 當中使用不同的neuron amount

Linear model: epoch為15000、Data amount為1000、learning rate = 0.05

|  |  |  |
| --- | --- | --- |
| 隱藏層neuron | 4 | 100 |
| Learning curve |  |  |

從圖中可以看見當隱藏層neuron數量上升，我們的模型用更快的速度收斂了，不過這是指epoch上的速度，實際上在訓練過程當中所需要的時間更長了，而在準確度方面，兩個都很好的到達了accuracy = 1.000的情況

XOR model: model: epoch為30000、learning rate = 0.01

|  |  |  |
| --- | --- | --- |
| 隱藏層neuron | 4 | 100 |
| Learning curve |  |  |

在XOR這個非線性的模型當中，隱藏層中的神經元數量就有很顯著的影響了，當我們使用了數量較大的神經元做隱藏層後，可以看見loss有效的降到了較低的數值，推測是因為在非線性模型當中，需要較高維度的模型才能有效的訓練，而從最後的accuracy當中也可以看見數量為100的模型表現優於數量為4的模型。

|  |  |  |
| --- | --- | --- |
| 隱藏層neuron | 4 | 100 |
| Accuracy | 0.5238 | 1.000 |

* 1. 試著不使用激勵函數

為了要單純找出激勵函數對我們模型的影響，所以這次我採用前面提到的參數，這樣就可以單純的觀察激勵函數究竟會帶來什麼樣的影響

|  |  |
| --- | --- |
| Linear |  |
| XOR |  |

從上面的圖當中我們可以看見，linear data的部分即使少了激勵函數還是可以正確地被判斷出來，而到了XOR data的部分，因為少了non-linear的激勵函數之後，模型就少了non-linear classifier的能力，所以我們的accuracy就變得慘澹了起來。

* 1. 試著使用不同的activation functions

使用XOR 模型來做實驗

|  |  |  |
| --- | --- | --- |
| epoch | Learning rate | 在隱藏層的neuron數量 |
| 30000 | 0.01 | 100 |

1. ReLU

f(x) = max(0,x), f’(x) =

使用ReLU 這個函數的好處是計算、收斂相當快，不過在使用上有一些需要注意的點，例如因為ReLU函數在0的地方不可微分，所以我們需要自己設定x = 0的微分值， 此外在權重的部分不能完全隨機，否則當output剛好都是0或者負值時，這個函數就不會被激發，這樣的情況也被稱為Dead ReLU問題。

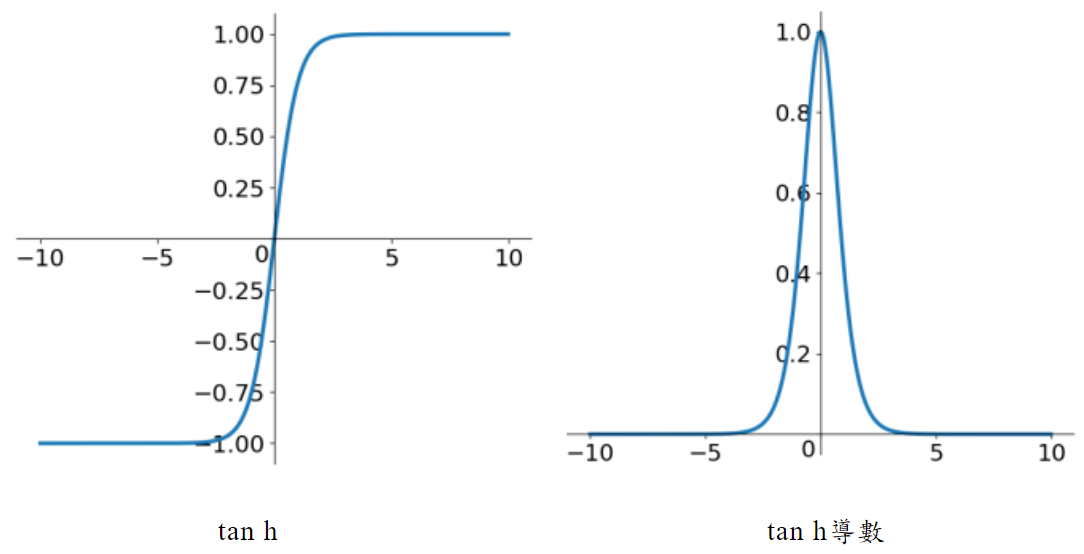
而因為換了激發函數，所以在參數上面我沒辦法直接使用以往的參數，而是重新設計了一組

|  |  |  |  |
| --- | --- | --- | --- |
| epoch | | Learning rate | 在隱藏層的neuron數量 |
| 30000 | | 0.01 | 150 |
| ReLU |  | | |
| sigmoid |  | | |

從兩者的圖表可以看見ReLU的模型收斂的比較快，也收斂到比較低的cost，在訓練上也可以感覺到使用ReLU模型的訓練速度較快。

1. Tanh:

,



從上圖當中我們可以看見tanh函數解決了sigmoid函數中 輸入x = 0輸出卻不為0的情況，而這次實驗我再次換了一組參數來進行比較

|  |  |  |  |
| --- | --- | --- | --- |
| epoch | | Learning rate | 在隱藏層的neuron數量 |
| 30000 | | 0.015 | 100 |
| tanh |  | | |
| sigmoid |  | | |

在導數的峰值變高、解決了在函數在input = 0時函數值不為零的兩個問題後，可以看見我們的loss收斂的比較快、也收斂到了比較低的點

1. Leaky ReLU

f(x) = max(0,x) + min(0.001x,0) , f’(x) =

Leaky ReLU解決了Dead ReLU的問題，而這邊我使用與ReLU同樣的參數來與ReLU、sigmoid進行比較

|  |  |
| --- | --- |
| ReLU |  |
| sigmoid |  |
| Leaky ReLU |  |

在這次實驗當中沒辦法很有效地看出leaky ReLU與ReLU的差別，我推測是因為權重的初始值問題，也因為我沒有碰到dead ReLU問題，所以在這邊看不太出差別

* 1. implement optimizer
     + 1. SGD: SGD其實就是單純的使用梯度作為修正權重用的數值，也就是我們一般在lab中實作的方法
       2. Momentum: 在momentum當中我們引入了一個新的超參數

過去我們在更新權重的時候使用的是

W = W – lr \* gt

B = B – lr \* gt gt代表gradient

而如今我們引入了 之後 將我們更新權重的方式改為

W = W – lr \*

B = B – lr \*

在這一個optimizer當中因為我們將權重的變化量套用了exponential moving average，所以我們可以將以往權重變化時產生的震盪給縮小(可以將，而如此一來我們就可以套用一個更大的learning rate，來達到更好的效果

XOR model，:0.95，epoch: 30000，隱藏層神經元: 150，使用sigmoid函數激發

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Learning rate | 0.01 | | 0.1 | |
| Learning curve |  |  |  |  |
| Learning rate | 0.5 | | 1 | |
| Learning curve |  |  |  |  |

左圖為套用了momentum optimizer之後，右圖則為沒有套用optimizer

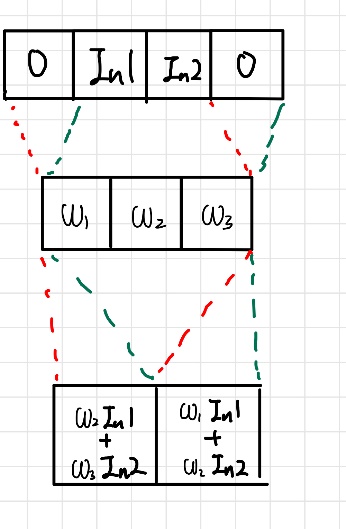
從圖中我們可以看到，雖然從收斂速度上兩者沒有太大的差別，但正如上面所提到的超參數，在模型訓練的過程當中降低了震動，讓學習的曲線變得平滑化。

3. Implement convolutional layers

這次無論是linear還是XOR，都是(X1,X2) 這樣的input，因此convolutional layer的部分我採用的是1D的形式。

首先，我先將本來shape = (1,2) 的input data padding 成(1,4)的data，如此以來我才有移動的空間。而如此一來，我們的Input data的形式變成(0,X1,X2,0)。

接著我設計的kernel size是3。 Kernel的資料是(w1,w2,w3)。

 結合上述兩個資訊，這次我新增的convolutional layer 有以下資訊

1. Input dim = 4
2. Kernel dim = 3
3. Output dim = 2

右圖由上至下分別為:

Input

Kernel

Output

而這次我沒有使用到max pooling。

而forward output的部分就如右上圖所示，所以我們只需要煩惱backward propagation，而因為這次設計我把convolutional layer放在最前面，而且只有一層，所以我們只要套討到 三者即可

一張含有 桌 的圖片

自動產生的描述

得到了每次修正的權重如何計算之後，把他串接到本來的model上面即可

接著就是看看效果，我使用XOR模型來探討這次的結果

使用的參數:

|  |  |  |
| --- | --- | --- |
| epoch | Learning rate | 在隱藏層的neuron數量 |
| 30000 | 0.1 | 150 |
| 有無conv layer | 有 | 無 |
| Learning curve |  |  |

在這次的模型當中神經網路並沒有帶來多大的效益，我覺得有個根本的原因是這次的模型並沒有複雜到需要動用捲積層，而在收斂上的速度也比較慢，這部分我覺得是因為我使用了padding zero的技巧，對於這次的資料而言沒有多大的幫助，反而是增加了資料的複雜度。