Neural Network　Homework1

Problem 3(Back Propagation)

R06922063

使用說明:

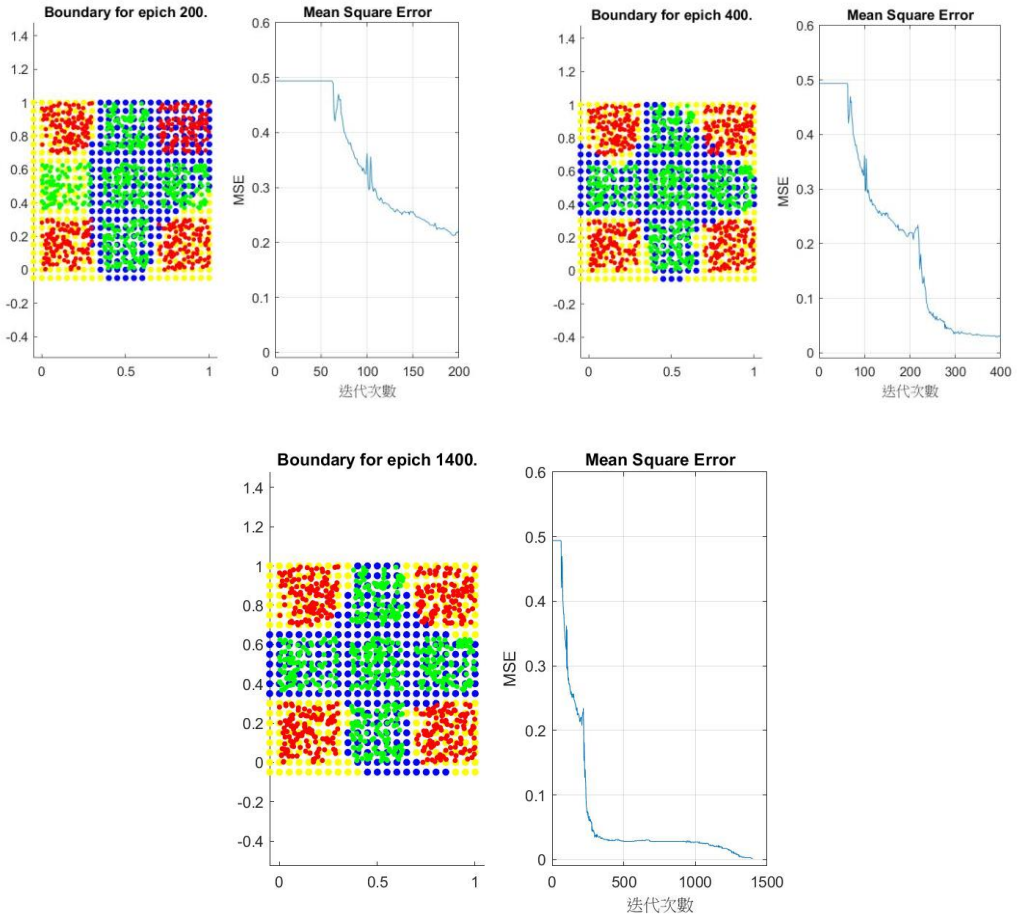
Hw1.m 檔案是我的主程式，裡面依照 section 分成了參數初始化、讀取與 Preprocessing 輸入資料、BP 演算法本體、Visualization 等等部分以及最後的建構 Hidden Tree 的部分。演算法的實作上是參考了維基上的 Back Propagtion 專頁。Code43.mat 和 Code86.mat 是我的程式針對 2-4-3-1 和 2-8-6-1 結構生出的 binary code，用來做為畫 hidden tree 的參考資料(raw data,未視覺化)

基本上整支程式只需要有Matlab的環境就可以跑了。要更改Hidden Layer的架構就直接改code 11行的 `hiddenLayerNodeNum` ，例如 [4　3] 代表第一個 hidden layer 有 4 個 neuron，第二層有三個 neuron。所以要跑 2861 就只要把這個變數改成 [8　6]就行了。程式在跑的過程中，每隔一定數量的 Epoch 就會製作一張當前的 MSE 和 Network 切出來的 decision boundary 示意圖在 hw1_result 資料夾內。(要控制多少個 Epoch 畫一次請設置 23 行的 drawIteration 參數)。另外整支程式跑完後會把建出來的 Hidden Tree(matlab cell)存成 code.mat 檔案。

另外，在%Set the MLP Parameter section 裡面還有很多參數可以自行設置：
Smoothing Parameter:　更新權重時 Momentum 的 alpha 值
LearningRate: 每次更新權重時更新值乘上的係數。
UsingMomentum: 若為 1，則使用 Momentum 更新 BP 權重
UsingLearningRateDecay: 若為 1，則學習率會隨著 Epoch 慢慢降低
UsingRprop: 若為 1，則使用 Resilient Back Propagation (很重要)

Result & Analysis:

首先我們先看 2-4-3-1 在訓練過程中的 Decision Boundary 和 MSE Curve。**(紅色綠色的點是資料點，藍色和黃色則是畫來標示 Decision Boundary 用的，黃色區域代表該區域的所有點都會被判定成紅色的 class，藍色區域代表該區域所有點都會被判定成綠色的 class)**

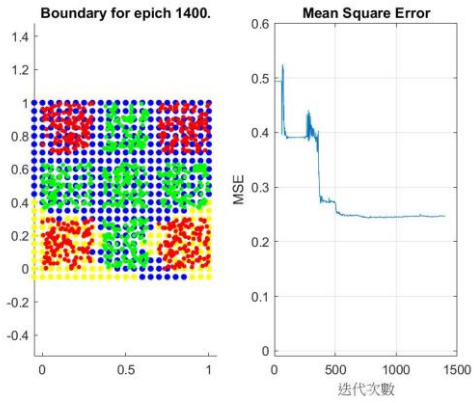


藉由 Decision Boundary Plot(黃色和藍色的點的變化)，我們可以清楚的看到訓練過程中 neuron 的行為。從 MSE CURVE 看來，在第 50 個 epoch 前，其實基本上 mse 是完全沒有下降的，表示 neuron 還找不到任何比較好切分這個十字型資料分布的方法，但是再之後 MSE 就快速下降，在 200 個 Epoch 時，我們已經可以看出 neuron 直接用垂直的 boundary 去切資料。到了 400 個 Epoch 時，neuron 已經大致抓出紅色資料的區分大致在四個角落。到了 1400 Epoch 時，就已經幾乎完美分開紅色跟綠色點了。

另外也很容易觀察到，MSE curve 在基本上下降時都是以很快的速度在下降，直到到達某一個特定的值之後改善速度就明顯下降，MSE Curve 也會趨近於水平線。這通常是因為擬合出來的函數已經可以把大部分的資料集區分開來，但是有一些少數位置跟其他同 label 的 data 不太一樣的少數資料集的存在，導致 NN 無法很好把這些例外資料學出來。通常直到一定數量個 Epoch 之後，Neuron 突然摸索出了一個突破口，MSE Curve 才會繼續下降。

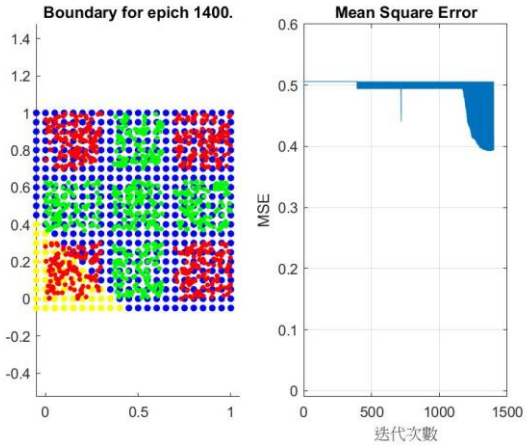
在實作上我觀察到了許多有趣的現象。像是因為一開始的 neuron 的權重是用 Random 出來的，所以有時運氣好只要一千個 Epoch 以內就可以達到完美區分所有資料，有時卻需要到數千個 Epoch 還無法完全區分所有資料。以是我用另一組隨機參數當作初始 Neuron 銓重來 train，可以看到在上面的狀況 1400

epoch 已經幾乎完美分開所以資料，在另一組隨機初始權重的狀況下同樣是 1400 Epoch，紅色資料卻只被區分出了兩塊。



另外，可以發現 Resilient Back Propagation (Rprop)相比於原本的 BP 演算法，在這個 hw1data 資料集上是相當重要的。

Rprop 的權重更新不看偏微分出來的"值"，只依照它的 Sign 來用固定比例的 Learning rate 來調整學習策略，可以有效避免神經元擬合出來的函數 Stuck 在函數某些 Local Minimum 出不去。以上的範例其實都是有使用 Rprop 的例子，我們來看一下沒有使用 Rprop 的例子。可以看到，沒使用 Rprop 的話在 1400 Epoch 時甚至只學到了很粗糙地切分出一塊紅色資料而已，相比於以上兩個範例都差的多。



另外就是關於 Learning Rate 的設置也很重要，如果我們沒有讓 Learning Rate 隨時間慢慢下降，在很多狀況下會發現 NN 最終只會卡在一個偏低的 MSE，但是永遠也下不去 0。這部份是因為 train 到後期，已經可以區分 98%以上的資料之後，若是還有一點點資料需要正確區分，我們可能只需要再微微調整 Neuron 的權重就好，如果 Learning rate 還是跟一開始一樣大會經常沒辦法正確擬合這些微小的參數變化。

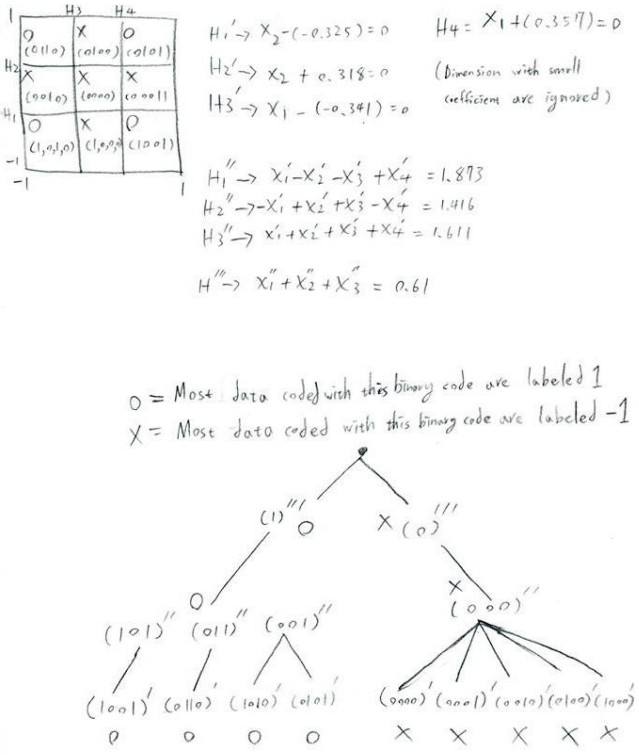
至於 Hidden Tree 的部分，因為要讓 Back Propagation 有辦法 Train 這些

Neurons，所以必須要用 Sigmoid Function(可微分)當作 Activation，而非課程裡所用的 Hard Limit function。在這樣的情況下其實 Neuron 的 Output 的意義不再只是替 input 做 binary coding 了，因為 Neuron 的輸出不再只是 0 或 1 了。

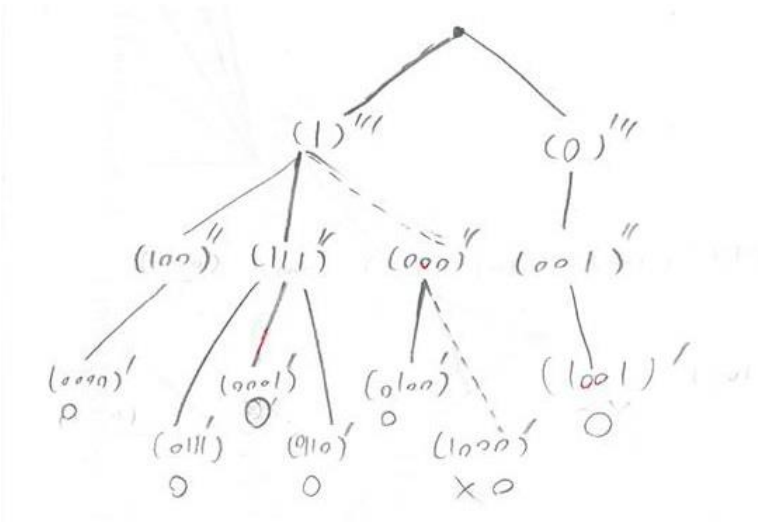
所以為了能夠建出 Hidden tree，我的作法是在 training 的時候使用 Sigmoid function，然後 Train 出 100%可以完美分開兩種 class 資料之後，再重新跑一次 input data，只是這次把 neuron 的 activation function 換成 Hard Limit。這樣子雖然最後輸出的結果會因為 activation function 的改變而有誤差導致正確率下降，但是至少我們可以得到每一層 Hidden Layer 的 Binary coding 來建 Hidden tree。

我這邊會提供兩種 hidden tree，一種是理論上的完美解(藉由直接觀察資料分布情形手算出可以區分開資料的 HyperPlanes)，一種是我自己實際上程式跑出來的 Hidden tree。

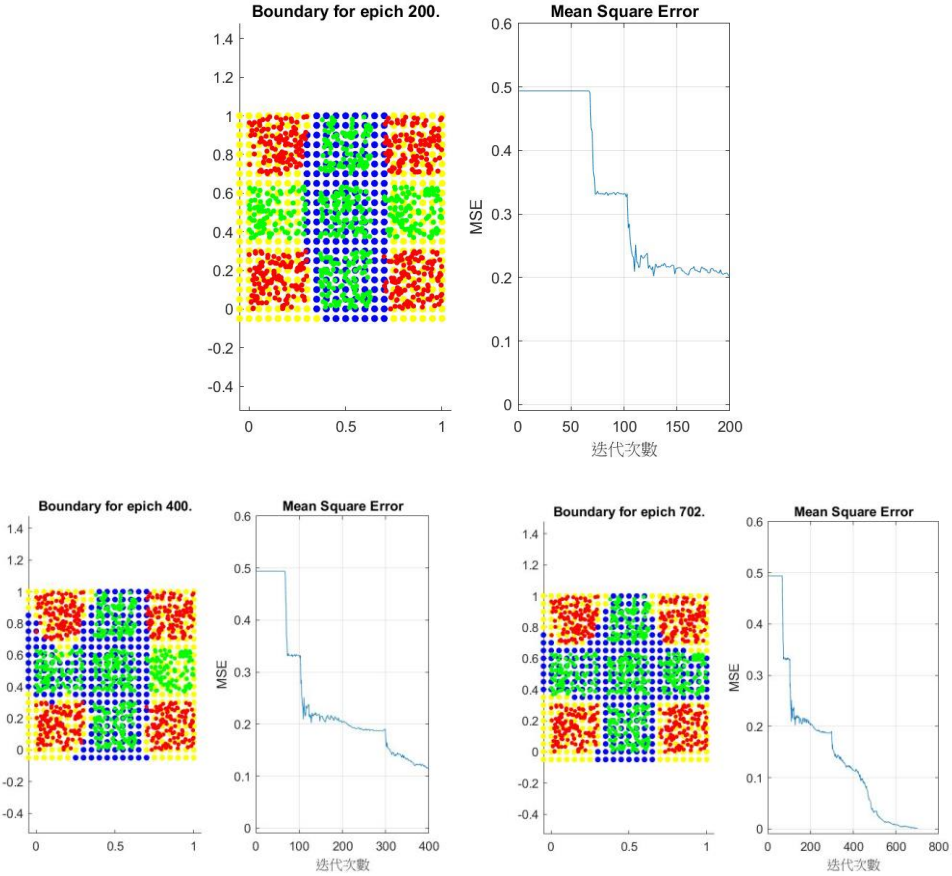
下面這一張是理論上完美解的 Hidden tree(附上 Hyper plane 的方程式)。我們可以看到在理論上最佳解裡面，第二層 Neuron 的 H1 和 H2 是 redundant 的，拿掉也不影響結果。



而下面這個則是我的程式實際上跑出來的 Hidden tree。可以看到雖然我在 training 時用 BP + sigmoid 已達到 100%分辨率，但是為了建 Binary code，在之後餵資料時把 Activation function 換成 hard limit 造成了有些 mixed region 會跑出來。(不過整體的正確率還是可以維持在很高，只是會有一些資料分布出來)，經過觀察，這個 hidden tree 裡面沒有 redundant 的 neuron。



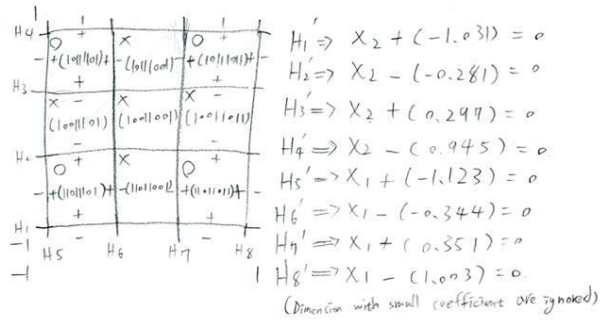
而如果我們把整個 Neural Network 的結構換成 2-8-6-1 的話，我們在 BP training 的過程中可以得到如下的實驗結果：

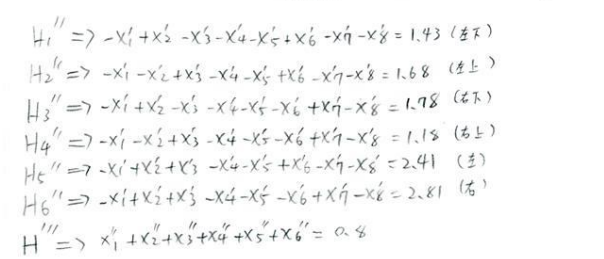


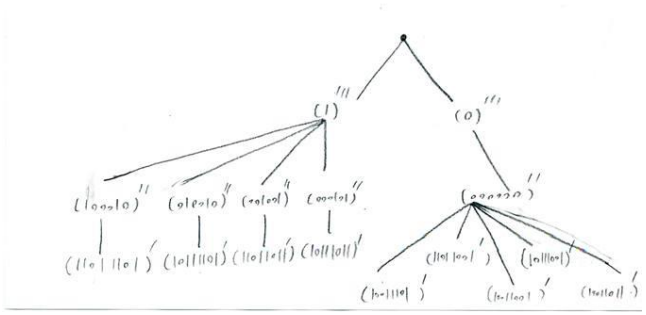
可以發現，相比於 2-4-3-1，2-8-6-1 能在較少的 Epoch 時就找出完美區分出所有 data 的解。在 200 個 Epoch 時就已經可以抓出兩個垂直的 region，到四百和 702 時分別又可以再正確區分出一塊綠色的範圍。

這一部分要歸功於較多的 Neuron 可以比較輕易的擬合比較複雜的函式(像是 hw1data.dat 這種十字型的資料分布)，也可以觀察到 2-8-6-1 不像 2-4-3-1 一樣在 MSE 到達某個瓶頸時會卡很久，而是一直保持著一定程度的速度在下降。

至於 Hidden tree 的部分，跟 2-4-3-1 一樣，我會提供理論上最佳的 hidden tree 跟我自己程式實際上跑出來的。以下是理論上最佳的 hidden tree：







以下則是我的程式實際上跑出來的 Hidden Tree。可以發現其實大部分第二層的 coding 都只有對應到一個第一層的 coding，這可能意味著即使我們不需要第二層，NN 還是有辦法正確區分出這個 DATA SET 中的資料的 Class。
同時也可以看出，在這個 hidden tree 中也沒有 redundant 的 neuron。