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。

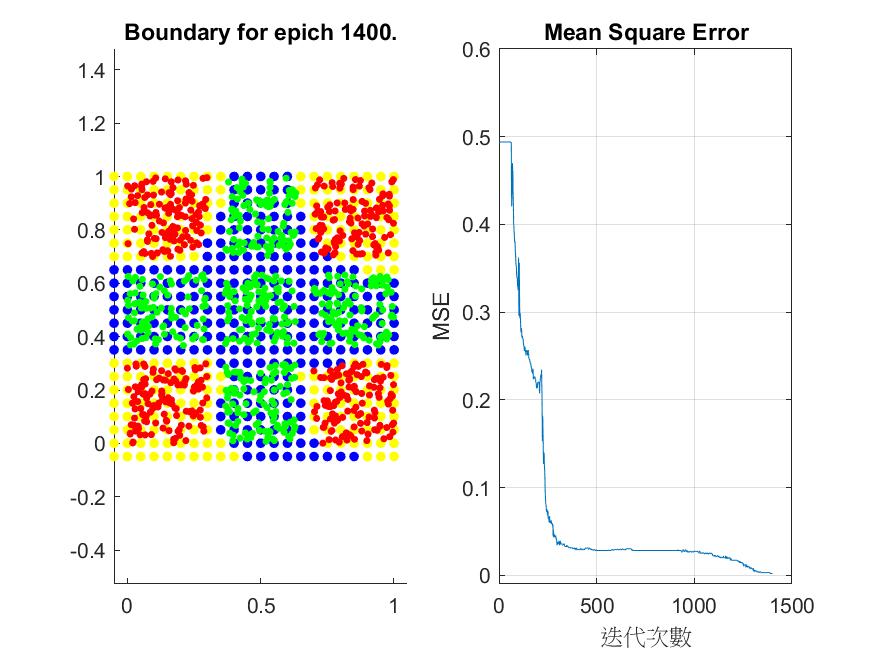
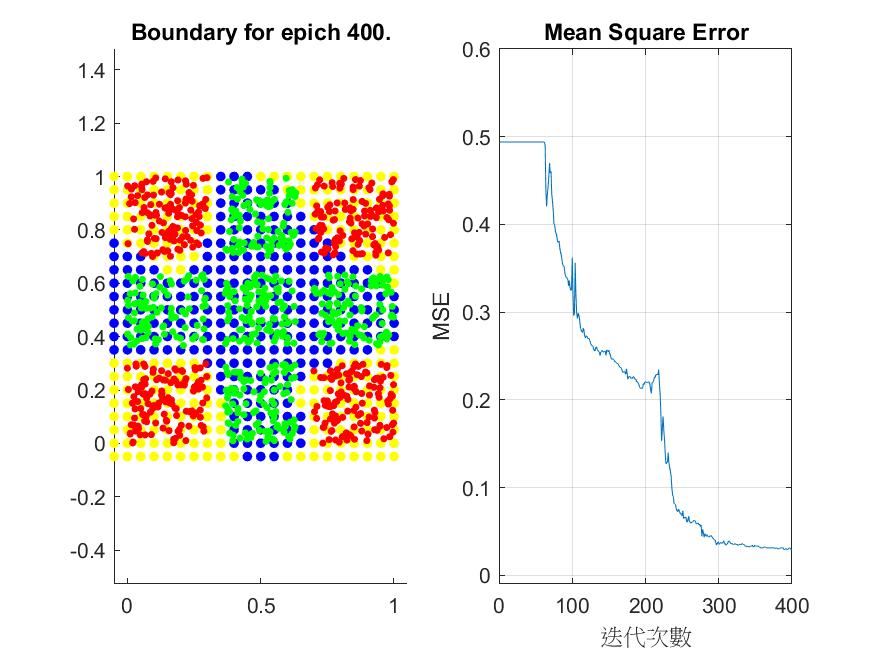
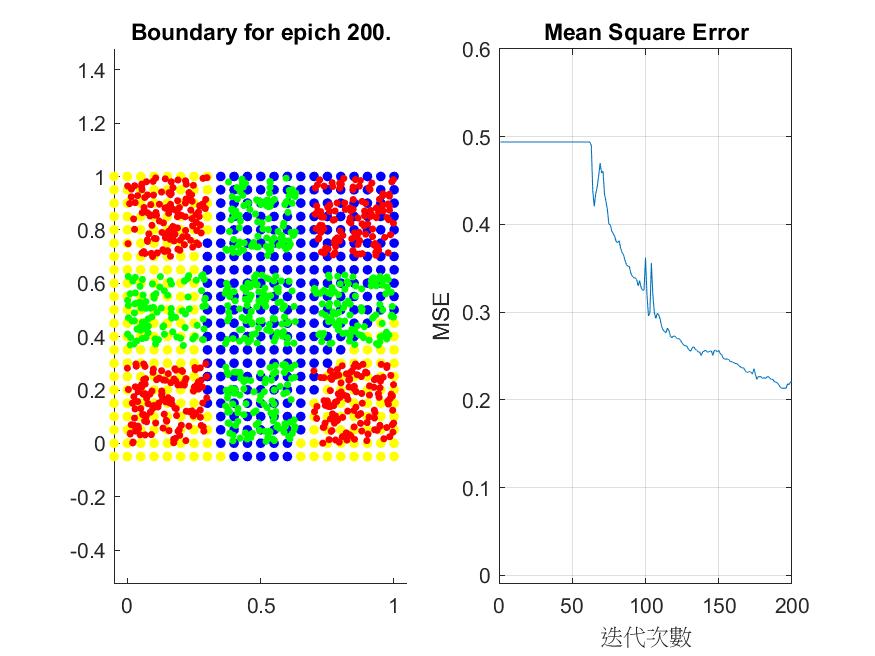
UsingMomentum: 若為1，則使用Momentum更新BP權重

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:

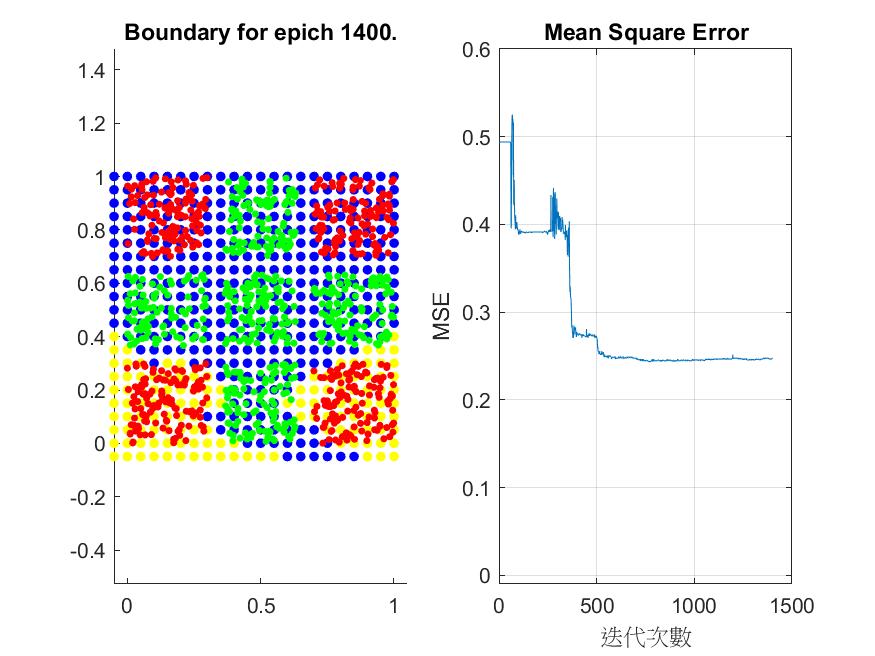
首先我們先看2-4-3-1在訓練過程中的Decision Boundary和MSE Curve。**(紅色綠色的點是資料點，藍色和黃色則是畫來標示Decision Boundary用的，黃色區域代表該區域的所有點都會被判定成紅色的class，藍色區域代表該區域所有點都會被判定成綠色的class)**



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。

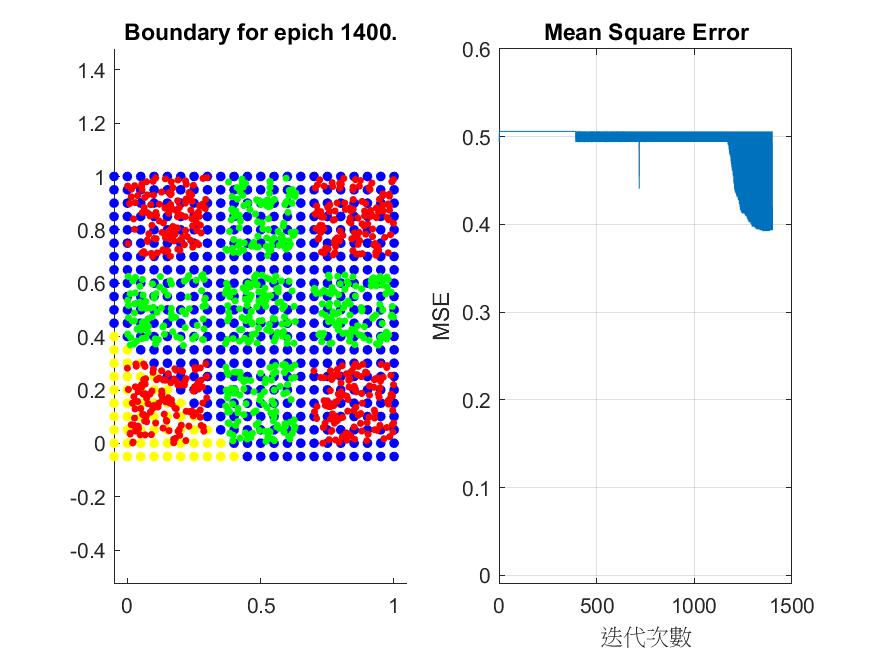
另外也很容易觀察到，MSE curve在基本上下降時都是以很快的速度在下降，直到到達某一個特定的值之後改善速度就明顯下降，MSE Curve也會趨近於水平線。這通常是因為擬合出來的函數已經可以把大部分的資料集區分開來，但是有一些少數位置跟其他同label的data不太一樣的少數資料集的存在，導致NN無法很好把這些例外資料學出來。通常直到一定數量個Epoch之後，Neuron突然摸索出了一個突破口，MSE Curve才會繼續下降。

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。



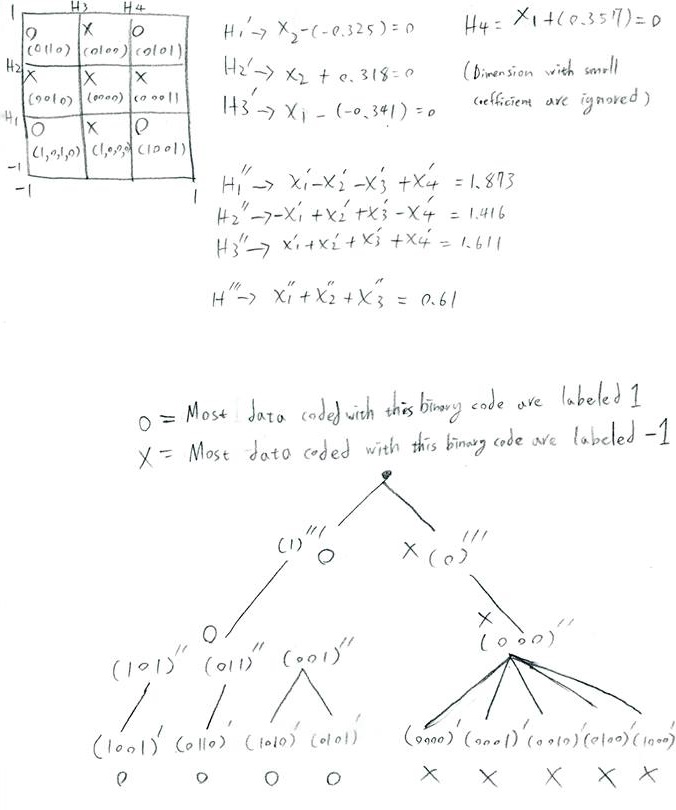
另外就是關於Learning Rate的設置也很重要，如果我們沒有讓Learning Rate隨時間慢慢下降，在很多狀況下會發現NN最終只會卡在一個偏低的MSE，但是永遠也下不去0。這部份是因為train到後期，已經可以區分98%以上的資料之後，若是還有一點點資料需要正確區分，我們可能只需要再微微調整Neuron的權重就好，如果Learning rate還是跟一開始一樣大會經常沒辦法正確擬合這些微小的參數變化。

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。

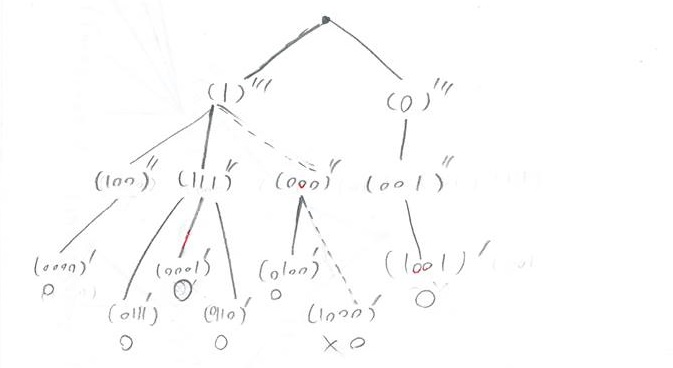
所以為了能夠建出Hidden tree，我的作法是在training的時候使用Sigmoid function，然後Train出100%可以完美分開兩種class資料之後，再重新跑一次input data，只是這次把neuron的activation function換成Hard Limit。這樣子雖然最後輸出的結果會因為activation function的改變而有誤差導致正確率下降，但是至少我們可以得到每一層Hidden Layer的Binary coding來建Hidden tree。

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。

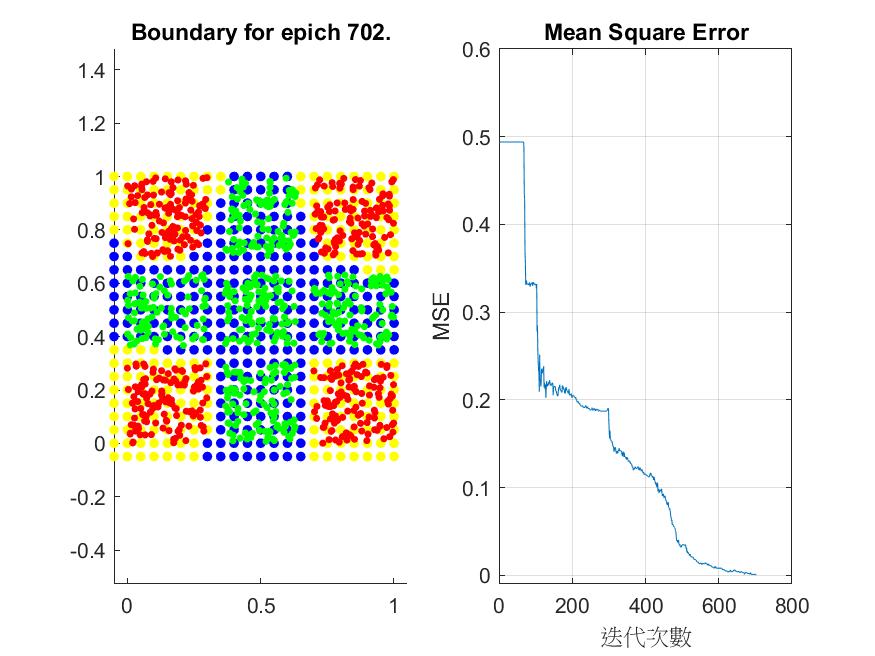
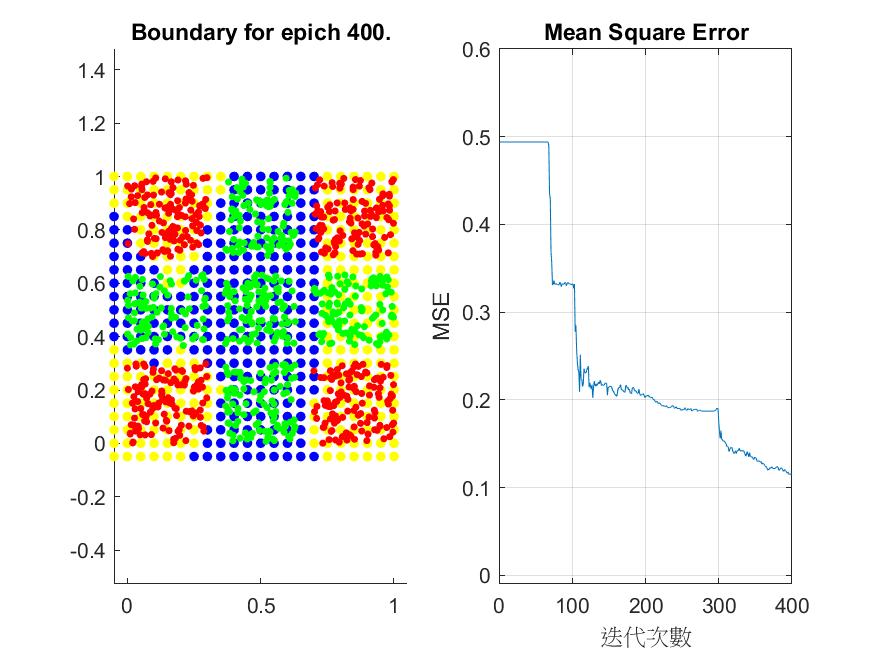
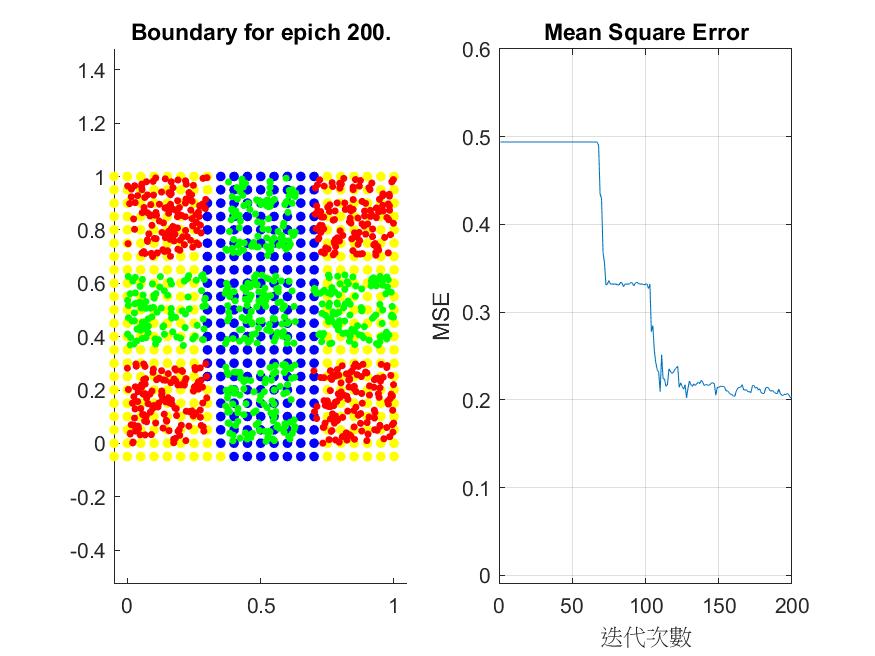
下面這一張是理論上完美解的Hidden tree(附上Hyper plane的方程式)。我們可以看到在理論上最佳解裡面，第二層Neuron的H1和H2是redundant的，拿掉也不影響結果。



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。



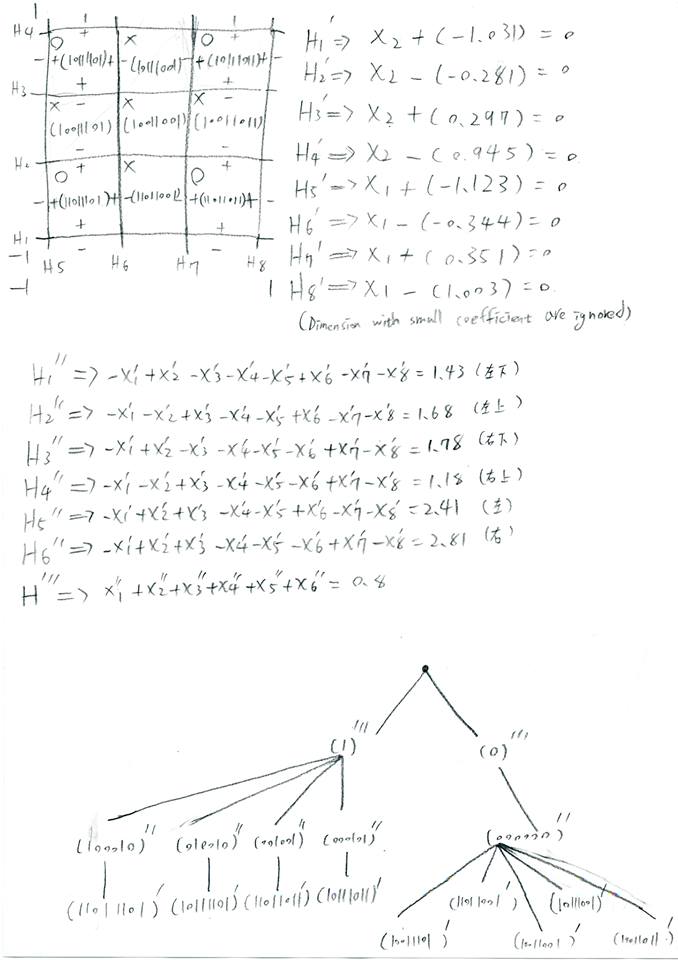
而如果我們把整個Neural Network的結構換成2-8-6-1的話，我們在BP training的過程中可以得到如下的實驗結果：



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。

