機器學習及其深層與結構化　HW1 Report

組名：Software On Demand

一、組員分工：

潘柏丞 B00901087：

1.建立dataset Class讀入資料

2.生成不同維度的data(Ex:39*9)及整理data順序使其一致且連續

3.建立最後輸出的.csv檔

吳孟寰 B00901114：

1. training set & validation set 切割

2.測試各參數實驗實作與實驗結果繪圖

黃騰輝 B00901027：

1. GPU加速矩陣運算toolkit 修改

2. Forward& Backpropagation演算法實作

3. Sigmoid layer及Softmax layer實作

4. 加入Momentum 參數

蔡育倫 B00901158：

1.CUDA與Linux server設置

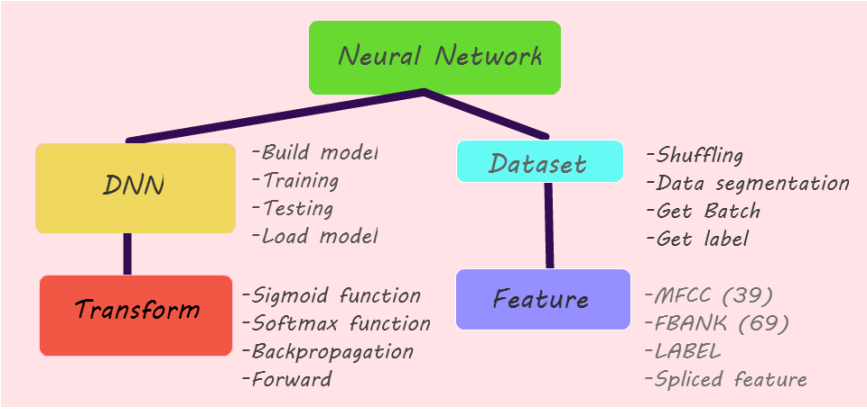
2.I/O for DNN model

3.Training, prediction in DNN level

4.Tuning parameter to reach simple baseline requirement

二、DNN實作：

a. 程式架構：



b. 資料結構及演算法設計：

i. Backpropagation for bias:

在課堂中推導過了backpropagation對weight更新的公式，因此不再贅述。但因為並未推導對更新bias的理論，因此在報告中補上：

令 b_i^l 表示第l 層layer對output neuron i 的bias，則

$$\frac{dC_r}{db_i^l} = \frac{dz_i^l}{db_i^l} \times \frac{dC_r}{dz_i^l}$$

而由課堂中backpropagation推導之公式：

$$\frac{dC_r}{dz_i^l} = \delta_i^l$$

因此bias對error function的貢獻為：

$$\frac{dC_r}{db_i^l} = 1 \times \delta_i^l$$

所以在更新bias值亦可以使用gradient descent。

ii. Softmax function:

本次作業的目標是一個對多個class的辨識，不同於sigmoid輸出對於單一class的辨識，softmax function是輸出對於多種class的可能性：

對於K個class，定義函數：

$$1\{Y = y^i\} = 1 \text{ 若函數內敘述為真；} 1\{Y = y^i\} = 0 \text{ 表示 } Y \neq y^i$$

則可以定義K-1維度的目標函數：

$$L(\bar{y}) = \varphi_1^{1\{Y=y^1\}} \cdot \varphi_2^{1\{Y=y^2\}} \cdots \varphi_K^{1-\sum_{i=1}^{K-1} 1\{Y=y^i\}}$$

其中 φ_i 表示第i個class的機率，經過運算可以得到

$$\varphi_K = 1/\sum_{i=1}^{K} exp(\eta_i)$$

$$\varphi_i = exp(\eta_i)/\sum_{i=1}^{K} exp(\eta_i)$$

配合cross entropy的使用可以用gradient descent的方法來調整參數：

$$E = \sum_{i=1}^{m} 1\{Y = y_i\} logy_i$$

$$\frac{\partial E}{\partial z_i} = \frac{1}{y} y(t_n - y) = t_n - y$$

其中 $t_n = 1\{Y = y_n\}$

iii. DNN class:

此class為整個dnn讀寫檔與演算法執行的控制物件。Object member包含儲存test data與train data的指針，dnn model的架構。方法上則實作了 load/write model, train, predict, backpropagate and compute error等功能。其中training的方式採用mini-batch training，在train函數給訂batch size, decay rate of learning rate, max iteration and validation size。在backpropagation與feedforward函數中，DNN的工作負責傳遞其output至下一級來達成deep learning。

iv. Dataset class:

此class主要處理data的輸出與輸入。Object member包含儲存train、test、label data的動態陣列、map，方法上除了data的前處理外，也實作了dataSegment、getBatch以及將data轉成cuda所用的mat的功能。

c. 資料處理：

i. Splice features in context window:

因為input data是由不同人所說的一段話分割而成，所以data彼此之間具有連續 性，考量有此因素，所以我們會將原來mfcc提供的39維data，將每一個data與其前後四個data結合，建立了39*9=351維的input data。

ii. Get batch:

起初，我們每次的mini-batch都是從training set裡面random獲取，後來也嘗試將整個training set shuffle過後照著順序取。我們發現，兩種batch取法對於accuracy並沒有明顯的差異，而後者可以省掉每次呼叫rand()的時間，因此我們後來改用後者。

d. 實作技巧：

i. 避免overflow：

Softmax函數具有overparameter的性質，因此再取指數前減去向量中最大值，與直接取指數在總和運算上是等值的，但這個步驟可以避免overflow的形情發生。

ii. 合併bias到weight matrix增加運算便利性：
在DNN forward 及 backpropagation演算法中，bias的運算如同是額外的weight對input為1的總和，因此在input最末端補上1，即可將bias合併到weight matrix中方便運算，而在backpropagation時更新bias時也可與wieght同步更新。

iii. learning rate setting：
在training的一開始，learning rate會設置的較大，使得在model的space當中有機會去發現更好的solution，當過了幾個iterations後learning rate可以適當的decay來縮小search的範圍找到該處最好的weights係數。

e. 遭遇之困難及除錯：

i. output layer normalization:
最初的原型並未使用softmax function來造成在training過程中，error rate會突然增加到100%的情況，在我們加入了softmax function後，這個問題也成功解決了。

ii. memory leak:
因為資料量及運算量都不小，因此在開發的過程中，有段時期我們在測試常常會因為記憶體不足而中止程式，因為是多人共同開發，因此花了一段時間確定動態記憶體管理是否正常操作，再經過互相檢查程式碼後成功讓演算中的記憶體穩定，以利程式開發。

iii. bottleneck of program:
雖然使用GPU做運算可以加速矩陣的運算，但在host與device之前的搬移上也花費不少時間，因此在程式的設計中須考量一些運算的順序與變數傳遞過程的型態，避免過多的搬移反而影響整體運行速度。

三、實驗及結果：

※以下實驗進行之預設參數如下：

learning rate = 0.002，momentum = 0.9，decay = 0.9

input dimensions - hidden layer nodes - output dimensions = 39 - 128 - 48

- Momentum對DNN training之影響：

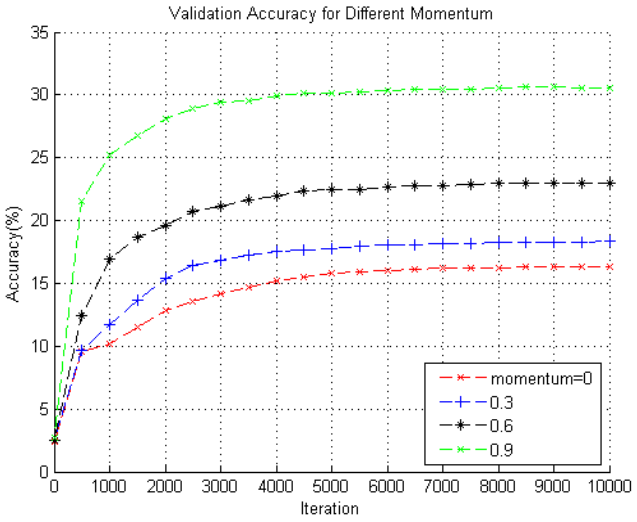
Momentum主要影響在更新DNN weight matrix的過程中，定義如下：

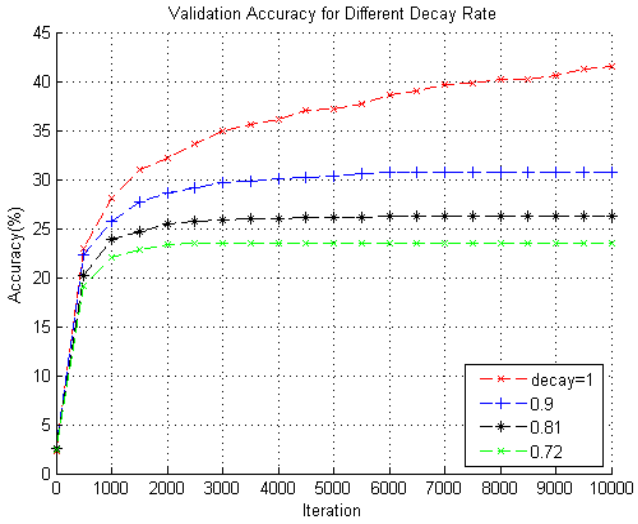
$$\Delta W_n = -\eta\frac{\partial E}{\partial w} + momentum \times \Delta W_{n-1}$$

結果顯示，momentum的使用會讓training初期有更大的機會跳出local minimum，因而得到較佳的表現。

- Decay 參數對DNN training過程之影響：

  我們在程式中使用的decay機制為，每經過200個iteration便會將learning rate乘上設定的Decay 參數，希望達到的目的是，在DNN在training的過程中慢慢減低learning rate使得在model初步建立完成後能夠細部的調整weight matrix，達到更好的表現。實驗結果顯示，在沒有使用decay的狀況下表現較佳，可能原因為，衰減的速度太快，model還沒有完全建立，learning rate就已經過小了。





- 單層hidden layer中node數量對DNN training過程的影響：

  單層當中，可以明顯看到node數量越多，準確度就越高，這顯示了此層hidden layer擷取到了更多的feature。

- Hidden layer層數對DNN training過程的影響：

  從圖中能很明顯看到單層能在相當快的時間內找到minimum，這可能是因為他的error surface較為單調。理論上，雙層與三層的model能train出更好的結果，但是由於iteration次數不夠，沒辦法從此圖看出。