MLDS_HW2　Report

Group name :

城隍府 YEE

Student ID_Name :

B02901030_陳尚甫

B02901071_黃淞楓

B02901074_陳奕禎

B02901144_林呈翰

壹、 組員貢獻

（一）陳尚甫：建立 rnn 的大致架構。將 dnn 的 IO.py 改成可讓 rnn 使用的形式。寫 report。

（二）陳奕禎：建立第一版 rnn，scan 的方式為一個 sequence 一個 step。Writefile 時的 smoothing & trimming。

（三）黃淞楓：建立第二版 rnn，scan 的方式為一層 hidden layer 一個 step。嘗試過用較小的 training data set 去 train，發現 cost 能降很低，但是跑 test data 時會爛掉。

（四）林呈翰：修改 dnnClass，讓 rnnClass inheritance dnnClass。NAG & gradient clipping。引入 github 分享檔案的方法。

（五）大家一起：debug。調整各種參數嘗試 training。

貳、 程式架構

一、　Data structure & algorithm design

(一)　rnn

我們將其分為三個部分：rnnClass.py、rnnTrainFunc.py 以及 rnnMain.py，分別負責處理程式不同層次的部分。

1. rnnClass.py

定義 rnn 這個 class，比較特別的部分是，因為 rnn 和 dnn 的基本架構非常接近，我們在這裡用了繼承（derived class），讓 rnn 繼承 dnn。

2. rnnTrainFunc.py

呼叫 rnnClass.py，並定義 train 和 test 的過程中，所需要用到的 function，如：activation function、step funtion 等。

3. rnnMain.py

定義 rnn 的 main 部分，呼叫 rnnTrainFunc.py，讓程式開始執行。

(二) IO.py

負責處裡 Input 和 Output，我們使用 dictionary 來儲存資料，包括 training data、label、output probability，以方便存取和比較，又因 dictionary 並不像 list 是有序的資料結構，所以我們有將資料的順序再用一個 list 儲存起來。較值得特別提的是，在 writeFile 這個 function 中，我們有做 smoothing，適時的修改 framewise predict 的結果（如 gbg 改成 ggg），來提升預測的結果，在 implement tips 會有較詳細的說明。

二、 Implement tips

(一) Setup

由於 rnn 本身就是 depth learning，簡單起見，我們的 rnn 只有一層 256 個 neuroons 的 hidden layer，activation function 為 ReLU，cost function 為 softmax，並以 identical matrix 來初始化 weight。

我們在 rnnTrainFunc.py 當中，定義了 training 和 testing 這兩個 function，為了在使用時可以較彈性的處理 train 和 test 的過程。如：先在 python 當中 import rnnMain，然後輸入 training(20)的話，就會 train 20 個 epoch，讓我們可以觀察結果來決定要不要繼續 train 下去，train 到結果夠好時，再呼叫 testing 來測試測資結果。

(二) Memory

為了 implement rnn 的 memory cell，我們建立了一個 class：MemoryNetWork，來 maintain 所有 memory cells。它是由 dnn 的 class：IntraNetWork 繼承而來，所以需要先簡單說明 IntraNetWork 的運作方式。

IntraNetWork 所 maintain 的並不是一層 layer，而是兩層 layer 之間的連線關係還有 weight 和 bias，我們將 input 和 output 都視為一層 layer，若 hidden layer 的數量為 L，那麼總共就有 L+2 層 layer，而 IntraNetWork 就會有 L+1 個，在每兩層 layer 之間。

同理，MemoryNetWork 也是 maintain 兩層 layer 之間的關係，在我們的 rnn 裡，有兩個 MemoryNetWork，一個介於 output layer 和 hidden layer 之間，另一個介於 hidden layer 和 input layer 之間。

(三) Scan

由於我們的 rnn 只有一層，所以我們分兩次 scan：第一次 scan 從 input layer 到 hidden layer，第二次 scan 從 hidden layer 到 output。我們定義了兩種 step function：stepReLU 和 stepSoftmax，在第一次 scan 中，輸入 input（x_seq），輸出 hidden layer

的 output（a_seq），且呼叫 stepReLU 為 step funtion；同理，第二次 scan 時，輸入 hidden layer 的 output（a_seq），輸出用 softmax 計算過後的 output（y_seq），則呼叫 stepSoftMax 為 step function。然後利用 y_seq 和 y_hat_seq（label）的結果來計算 cost 和 gradient。

(四) Update

在更新參數的部分，我們定義了一個 update function 來 implement NAG，update() 有兩個 argument：gradient parameter 和 momentum 係數 lambda，計算 movement 的方式即是：movement = lamba * last_movement – gradient * learning rate，所以可以藉由調整 lambda 來控制 momentum 衝刺的程度。

(五) Smoothing

在 write file 的時候，我們使用三種策略做 smoothing：

1. 觀察連續的三個 frames，如果前一個 frame 和後一個 frame 的 phoneme 一樣，就把中間 frame 的 phoneme 取代成前一個 frame 的 phoneme。
2. 觀察連續的三個 frames，如果三個 frames 的 phoneme 都不一樣，就把中間 frame 的 phoneme 取代成前一個 frame 的 phoneme。
3. 觀察連續的四個 frames，如果第二個 frame 和第三個 frame 的 phoneme 一樣，但是和第一個及第四個 frame 的 phoneme 不一樣，就把第二個和第三個 frame 的 phoneme 取代成第一個 frame 的 phoneme。

三、 Bugs & solution

（一）無法更新 Wh

有一段時間我們無法更新 MemoryNetWork 裡面的 weight（Wh），後來發現是 movement 這個 vector 的長度沒有設好，導致 gradient 沒有辦法去更新到 Wh。

（二）Type Error

雖然已經是作業二了，我們仍被 Theano 的 type error 所困擾著，input 吃進來的 data 沒有好好 cast，type 變成 int64 或是 float64，讓 theano（我們預設 floatX = float32）沒有辦法正常吃進 input，例如：input 為 0.9（float64），吃進檔案時變成 1（int64），丟入 theano 時就發生了 type error。儘管這類問題最後都能夠找到原因，但在 print 一堆資料比對哪個環節出錯的過程，耗費了很多心力。

（三）Cost 無法下降

這是最令我們苦惱的 Bug，程式能夠順利執行，但是怎麼樣都 train 不動，cost 也無法下降，我們嘗試了一些方法，都沒有辦法得到好的結果。最後是使用 dnn 的 output 直接做 smoothing 並 trim，得到了一個勉強超過 baseline 的結果。儘管沒有好的成果，我們還是繼續說明所做過的嘗試，以及其結果。

參、 實驗設計及結果

一、 小測資實驗

我們先把 training data（112 萬筆）切成小測資來實驗，得到了還算不錯的結果：

（一）12012 筆 Data、LR = 1E-4、Clip = 10、800epochs

每個 epoch 所需時間大概只需要 10 秒，cost 從一開始的 4.8 最終降低成 0.4 左右，即 softmax 的 possibility 為 0.67，已經接近全部 match。

（二）40093 筆 Data、LR = 1E-4、Clip = 10、400epochs

每個 epoch 所需時間約為 30 秒，cost 從 4.4 降低到 2.23，換算成 softmax possibility 為 0.108，對於長度為 48 的 output 而言，亂猜的機率期望值為 0.0208，表示雖然效果不顯著，但程式確實是有在 train 的。

二、 調整 Learning Rate

在 Clip = 10、lambda= 1.1 的情況下，調整 learning rate 的結果如下：

1 hidden layer(256)								
LR	Cost(ep1)	Gradient(ep1)	Cost(ep2)	Gradient(ep2)	Cost(ep3)	Gradient(ep3)	Cost(epN)	Gradient(epN)
5.00E-04	3.6406	0.4218	3.7035	0.5461	3.7604	0.6167	3.4(N=35)	0.5(N=35)
1.00E-04	3.6342	0.7547	3.6323	0.7152	3.6299	0.6831		
5.00E-05	3.6344	0.8427	3.6228	0.8345	3.6398	0.9364		
1.00E-05	3.6128	1.0804	3.601	1.3091	3.6001	1.544	3.5976(N=10)	1.6821(N=10)
5.00E-06	3.6013	1.2672	3.589	1.3375	3.5895	1.4578		

可以看出 Cost 的幾乎沒有在下降。

聽說因為 rnn 的 cost 不是很平坦就是很陡峭，所以會在某個瞬間大幅下降，但我們曾經嘗試以 learning rate 為 5E-4 train 了 35 個 epoch，cost 穩定的小幅下降到了 3.4，沒有觀察到大幅下降的發生，而且我們在小測資實驗時，cost 從 4.8 降到 0.4 的過程也是穩定的小幅下降。

三、 調整 neuron 的個數及層數

我們懷疑過有沒有可能是因為 hidden layer 的層數太少，沒有辦法使 cost 繼續下降，所以我們做了兩層 hidden layer 的實驗：

2 hidden layer(128x128), LR = 5E-4					
Cost(ep1)	Gradient(ep1)	Cost(ep2)	Gradient(ep2)	Cost(ep3)	Gradient(ep3)
3.6432	0.263	3.6603	0.2762	3.6714	0.2852
2 hidden layer(64x64), LR = 5E-5, eta = 0.5					
Cost(ep1)	Gradient(ep1)	Cost(ep2)	Gradient(ep2)	Cost(ep3)	Gradient(ep3)
3.6441	0.6798	3.6326	0.5784	3.6269	0.5493

同樣的，結果也沒有顯著的改善。