深度學習與其結構化應用

組別名稱：Software On Demand

組員及學號：

黃騰輝 B00901027

潘柏丞 B00901087

蔡育倫 B00901158

吳孟寰 B00901114

組員分工：

黃騰輝：

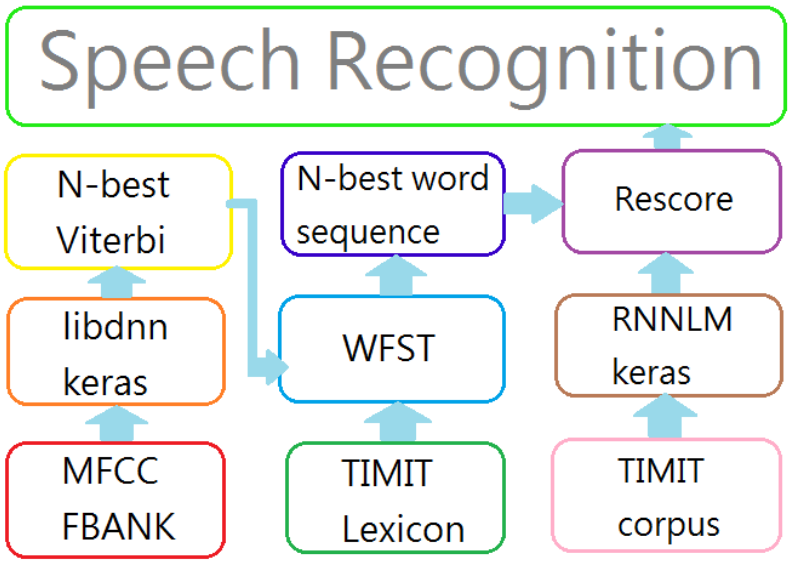
1. N-best Viterbi algorithm 實作產生 phone sequence
2. 撰寫資料處理工具
3. StochHmm toolkit 之應用

潘柏丞： WFST 改寫,相關 script 工具的撰寫

蔡育倫：libdnn, keras and caffe toolkits 來做 DNN 模型的訓練

吳孟寰：rnnlm simple RNN nbest rescoring, keras deep LSTM nbest rescoring

一、程式流程圖：



二、演算法及使用技巧：

1. N-best Viterbi algorithm:

不同於 Viterbi algorithm，每一次的 state 轉換要考慮到前一個 state 的 S*N 種可能，並排序取前 N 個可能當作下一個 state 的轉換，運算複雜度為 Viterbi algorithm 的 $O(N^2 logN)$倍，即 $O(T \times S^2 \times N^2 logN)$。**在我們設計的語音辨識流程中，這步驟是以 DNN 預測的後算機率以 Maximum Likelihood 的數學模型得到最可能的 phone sequence.**

2. DNN state/phone 預測模型，訓練中使用之技巧：

由於語音的特性具有連續性，因此適當地將輸入 feature 做堆疊在 DNN 的訓練上有相對較好的表現。在輸出端即使是相同的 phone label 也會有因為接於不同的 label 前後而

有不同的特性，因此採用作業一給的 1943 的 label 來訓練，再轉回 48 的 phone 的 trimming，其表現會比只用 48 或是 39 個 label 來得好。但當兩者維度皆放大時所需的記憶體量與花的時間亦增多，在機器可負荷的情況下要做適當的調整。在輸出端的訓練加上約莫 0.4 的 dropout 比率可以突破原本什麼都沒有所無法突破的正確率的上限，tradeoff 為所花的時間會增加。

3. Language Model 字詞分數模型，訓練中使用之技巧：

在最後 rescoring 的步驟，資料的前處理顯然是最重要的步驟。大部分的步驟都依照作業三，不過由於助教的 lexicon map 以及轉中文 label 的 map 裡有些特殊符號需要特別處理，少了這些步驟會導致不少 OOV 的產生。另外，為自己的 training corpus 選擇最適當的 layer size 是相當重要的，考量到訓練速度影響，corpus 的大小必須受限制，開太大的 node 只會讓 model 的複雜度大到 corpus 無法負荷，而讓 model overfit。以下的實驗結果會說明這個部分。

三、實驗及結果：

1. 比較不同的 label 與 input 格式對於準確度的影響

在實驗中我們跑了 39 維 feature 與 351 維(39 加上前四後四)對上 48 個 label，後者的表現因為有考慮連續性比前者好。而在這個實驗中以 48 作為 label 的原因為記憶體與 toolkit 的限制，因為 libdnn 中的 ReLU 功能有問題在沒有 RBM 建模型的情況下其層數無法大於五層且輸出的 label 無法使用 1943，我們的訓練的過程 activation 函數選擇 sigmoid 並加上 0.3 的 dropout 比率。後來換了 toolkit 之後解決的 memory 的問題改用 69 與 345(69 加上前二後二)並以 1943 為 label 進行訓練，在 Final 的 kaggle 上分數可突破 simple baseline。其中 69 的表現比 39 好是 HW1 所實驗出來，而 69 與 351 的差別在結果尚不明顯，可能原因與 model 的複雜度有關，因為記憶體限制而沒有用更複雜的網路所以使得兩者的表現差不多。

在替換為 keras 並解決 memory 問題後，皆用 345 與 1943 為資料的形式，時間限制一個 epoch 約為兩個小時，層數設定為三層皆為 2048 的隱藏層加上 ReLU 與 0.3 的 dropout 比率，其表現皆優於之前的設定。

2. N-best Viterbi 對於一對一狀態及多對一狀態的表現差異

在我們的語音辨識系統中，嘗試了不同類型的 DNN 預測之後算機率(posterior)，首先我們直接預測 feature 對應到的 48 個 label，並且將每一個 label 當作一個 state，以 Viterbi 演算法計算出狀態轉換序列，相當於 Phone 序列。另外我們將作業一的 1943 個 state 對應到 48 個 label，以不同的 DNN model 預測一特徵向量對應到的 state 後算機率，同樣可得到 Phone 序列，經過刪減後即可比較語音辨識的優劣。

epoch	48 phones treated as states DNN (351-2048-2048-2048-48)	epoch	1943 states DNN (69-2048-2048-2048-1943)
30	7.27664	3	6.92897
50	7.19439	10	6.67477

(單位: kaggle 錯誤分數)

3. 考慮 monophone 及 biphone，於語音辨識上的優劣

如前文所述，我們嘗試過將 Viterbi 演算法中計算的 state 一對一對應到 phone 所以在計算 state transition 的機率時，等同於考慮前一個 phone 出現的條件下，下一個 phone 為 48 個 phone 內，有最大 likelihood 的，我們將這樣的 phone transition 視為 monophone，因此，如果考慮了預測出最大的前兩個 phone 後再計算下一個 phone 為何者理論上會比 monophone 表現更加。只是更重要的問題是，能夠提升多少的表現。

Phones	N-best Viterbi	N-best WFST	Rescoring	Score
monophone	100	1000	RNN	7.22991
biphone	100	1000	RNN	7.11985

4. WFST 的 nbest 數量對於後面 Rescoring 的影響

我們利用 WFST 來解出所有可能的語句組合，但是我們並不會每次都用所有的可能組合來 rescore，而是給定一個 n 作為 n-best 數字上限，選出最短的 n 條路徑作為可能的組合，理論上使用越大的 n 越好，但隨著 n 的數字增加，WFST 所需的執行時間也更久，因此，我們對同一組 phone sequence，用不同的 n-best 跑 WFST，最後再經由 rnnlm，想找出時間及準確度的權衡下，最好的 n。

以下為使用同一組 1-best Viterbi 獲得的 phone sequence，經由不同 n-best 的 WFST 的運行時間與準確度。

N-best WFST	N-best WFST	Rescoring	Time(sec)	Score
1	1000	RNN	97	7.22097
10	1000	RNN	149	7.04120
100	1000	RNN	632	6.97522
1000	1000	RNN	6416	6.94382

由上面表格可以看出隨著 n-bet 的數量增加，運行時間也不停地增加，其實對於分數影響沒有非常大，像是 100 跟 1000 的分數只有 0.03 的差距，但是運作時間差了快 10 倍，所以想要快速得到分數不差的結果可以採取 100-best WFST，但若是想要獲得更高的分數，就要採取 1000-best 甚至更多的 WFST，但相對的，也需要多花不少的時間。

5. Language Model Rescoring

在這個步驟，我們一共嘗試了兩種不同的 language model，rnnlm 和 keras，前者是一個和 srilm(n-gram language model)整合的 simple rnn 模型，後者則是一個用 python 寫成的 Deep Learning toolkit，可以很彈性的用來整合不同功能的 layer。前處理方面，RNNLM 使用的是作業三的語料庫外加 TIMIT 的 training sentences。LSTM 由於運算速度慢，只從作業三裡挑出長度比較適合的 4 萬句話加上 TIMIT。

在 rnnlm 裡面，我們用的是一般的單層 RNN，透過調整不同的 hidden layer node 來尋找最佳的 language model。測試的結果如下：

hidden layer node size	Score
50	7.00749
100	7.01124
200	6.98502

由表可知，hidden layer node 數量越多，可以萃取的隱藏特徵會越多，也越能 rescore 出好的句子，但是這個影響因素在我們的實驗裡並不明顯。我們認為原因在於 validation set 與 testing set 中間存在了大量的 bias，導致 validation error 能不停下降，但 testing set 早就已經被 overfit 了。

由於 RNN 表現有限，我們後來選用 keras 來 train LSTM 的語言模型。LSTM 可以在不同的層中間加上 dropout 以及 pooling 的功能，因此應用起來特別靈活，變數也相對比較多。不過由於在 linux 上跑會有 memory error，而 mac 則有一種切換 swap 的型式可以容納大量的虛擬記憶體，因此我們只能在 mac 上面用 cpu 運作，速度很有限，因此以下只討論每層 node 數量對於 rescoring 表現的影響。以下的 model 都是用同樣的 WFST 結果進行 rescoring，並且 model 是由兩層 LSTM 以及一層 fully-connected layer 組成。

hidden layer node size	Score
50	6.62172
100	6.63296
200	6.64299

原先，我們預期隨著 node size 變大，model 的 performance 應該要變好，但是這個現象不只不明顯，甚至呈現著些微相反的趨勢。我們推測，這是因為我們只使用 TIMIT 的 3696 句話當作 training corpus，這樣的大小根本不足以負荷整個 model 的複雜度。從 training 的過程也能看出來，在前 10 個 epoch 裡 training 以及 validation 的準確度都極低，只有 loss 穩定下降，到了第 12 個 epoch，準確度突然上升到 2%，並在下一個 epoch 就迅速下降；model 只在一個 epoch 之間就達到 overfit，代表 corpus 的大小可能不夠，所有的 model parameters 都被 fit 到 corpus 上，根本就沒什麼學習效果，也因此 model 不會隨著 node size 變大而變好。不過相較於 simple RNN，LSTM 的確有較好的 baseline performance，這個部分值得我們用 keras implement simple RNN 來進行更公平的比較。

6. PostProcessing

我們從 language model 找到最佳可能的 word sequence 之後，發現裡面含有許多像是 phoneme 的東西，因此我們做了一點 post-processing 將不是 word 的東西刪掉，這樣的結果大約可以進步 0.04 左右。