ADL x MLDS Fall 2017

Homework 1 Report

Student: CSIE R06922068 林裕景

Introduction

這次作業的目的是藉由 TIMIT 數據集的語音資料，利用深度學習建立能判斷其語音(phone)的模型。助教已經先對原始音檔進行 feature extracting by frame，透過 moving window 方法擷取小段時間內的音訊特徵，提供了 mfcc 和 fbank 兩種不同方法取出的 features，分別是 39 維和 69 維，而每個 frame 也都有標上標籤表示屬於哪種語音(共 48 種)。

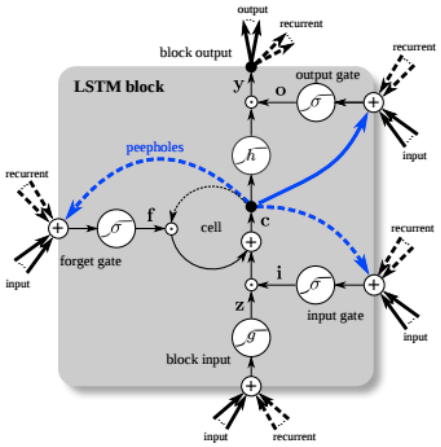
由於語音資料具有時間上的關聯性，因此使用 RNN 類型的神經網路應該會有不錯的效果，其中我選擇的普遍被廣泛使用的 LSTM，許多研究指出 LSTM 的 gate 機制能使它有效的記住或遺忘不定時間長度的數值，在預測時間序列資料有顯著的效果。而在進入 RNN 前使用 CNN，從原始資料的特徵間關聯彼此，雖然是助教要求的一種模型，但我也覺得很值得嘗試。在經過 LSTM 後，接 39 維度的 Dense 層加上 softmax activiation function，以輸出分類的結果。在實驗的期間，我也有嘗試在 LSTM 和最後的 Dense 間增加幾層 Dense，提高從時間序列學出的特徵之非線性關係，期望能找出隱藏的關聯。

最後，由於需要大量測試不同模型的組合，以及調整參數已找到最佳的配置，我選擇使用 TensorFlow+Keras 作為主要的工具進行實驗。

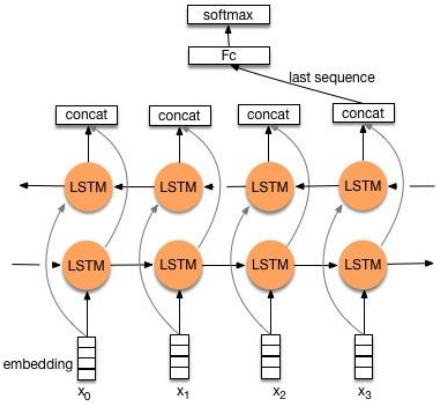
Model Description

作業中共設計了三個模型，第一個是單純的 RNN，第二是先 CNN+RNN，最後是我試過各種排列組合後找出最好的模型。

在我設計的三個模型中，我所使用的 RNN 都是 LSTM (如右上圖)，有著較好的能力去記憶不定時間間隔的數值，我認為對於語音資料的訓練應該會有幫助，比方說會有某個音出現後，過兩三個音節後通常會出另一個音的情況。

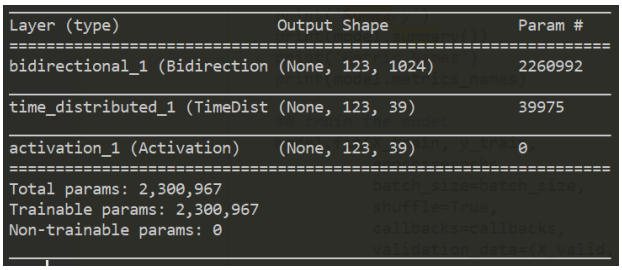


而在每個 LSTM 外，我還包了 Bidirectional Layer (如右下圖)在它的外面，藉由在時間序列上雙向的 propagation，能使 neurons 接收到時間前和時間後的訊息，比起一般只往前傳遞的 LSTM，我認為雙向能有更好的效果。為了方便表示，以下都會用 Bi-LSTM 代表 Bidirectional LSTM Layer。



而三個模型的架構如下圖：

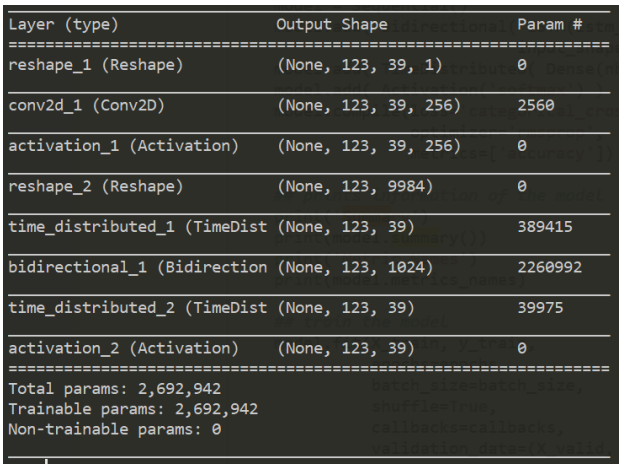
RNN Model



Layer (type)	Output Shape	Param #
bidirectional_1 (Bidirection	(None, 123, 1024)	2260992
time_distributed_1 (TimeDist	(None, 123, 39)	39975
activation_1 (Activation)	(None, 123, 39)	0

Total params: 2,300,967
Trainable params: 2,300,967
Non-trainable params: 0

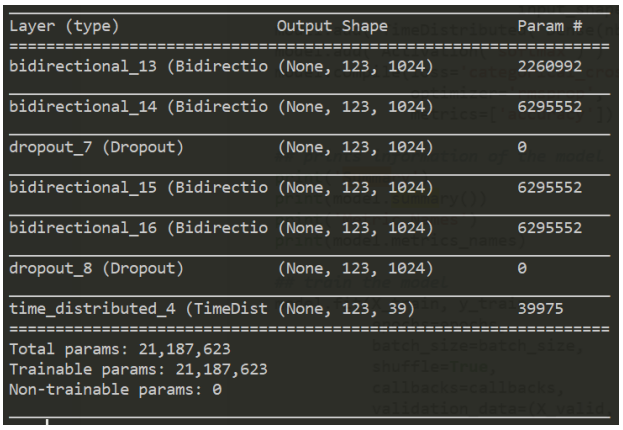
CNN Model



Layer (type)	Output Shape	Param #
reshape_1 (Reshape)	(None, 123, 39, 1)	0
conv2d_1 (Conv2D)	(None, 123, 39, 256)	2560
activation_1 (Activation)	(None, 123, 39, 256)	0
reshape_2 (Reshape)	(None, 123, 9984)	0
time_distributed_1 (TimeDist	(None, 123, 39)	389415
bidirectional_1 (Bidirection	(None, 123, 1024)	2260992
time_distributed_2 (TimeDist	(None, 123, 39)	39975
activation_2 (Activation)	(None, 123, 39)	0

Total params: 2,692,942
Trainable params: 2,692,942
Non-trainable params: 0

Best Model



Layer (type)	Output Shape	Param #
bidirectional_13 (Bidirectio	(None, 123, 1024)	2260992
bidirectional_14 (Bidirectio	(None, 123, 1024)	6295552
dropout_7 (Dropout)	(None, 123, 1024)	0
bidirectional_15 (Bidirectio	(None, 123, 1024)	6295552
bidirectional_16 (Bidirectio	(None, 123, 1024)	6295552
dropout_8 (Dropout)	(None, 123, 1024)	0
time_distributed_4 (TimeDist	(None, 123, 39)	39975

Total params: 21,187,623
Trainable params: 21,187,623
Non-trainable params: 0

詳細的建構原理在下部分說明。

Model Implementation

實作模型的部分可以分為預處理、訓練、和預測與後處理，訓練三個模型用的 code 大部分都是相同的，只有中間訓練時給予的資料和模型不一樣。

Preprocessing

對於訓練和測試資料我都用一樣的方式進行資料預處理。

首先將必要的資料讀取進來，包含了：train_mfcc、test_mfcc、labels、48to39 map、phone char map。我使用 pandas 套件做最初的資料處理，前述的資料都先以 data frame 的形式儲存，而由於 training data 和 labels 內的順序不同，我將兩個依據 ID merge 起來，接著從 data frame 中拿出 X_train、X_test、y_train 和 ID_test (以便預測時依據 testing data 的順序輸出)，再把兩個 map 轉換成 dictionary。最後是用 sklearn 的 LabelBinarizer，對字串型態的 y_train 做 one hot encoding，之後才能讓模型來分類。此外原本的 y_train 有 48 種 labels，Kaggle 上的資料只有 39 種 phones，所以我用 phone char map 把 y_train 轉換成 39 種 label，最後會形成 39 維度的 one hot 向量。

Training

首先，訓練時我先將 training data reshape 成適合的 size，因為 Bi-LSTM 要求設定一個 timestep，使得這段序列中的 data 都會有前後關聯性的傳遞，這部分三個模型都是一樣的。我的作法是，將所有的 data 串接再一起，然後每 123 個分為一組，最後不夠 123 的從資料頭補。

第一個 RNN 模型很一般的接了 Bi-LSTM 和 Dense(39)，選擇 Bi-LSTM 的原因如前面所述，普遍被認為效果較好。

第二個 CNN 則是基於 RNN 模型，前面加上一個 CNN，我的方式是讓核心維 3x3 的 CNN 跑過原始的 data，讓原本時間先後以及前後 frame 的資料互相關聯，然後產生出過多的維度後（因為我設 CNN 維 256 個 node），flatten 回一個維度，再用 Dense 壓縮成跟原本一樣的 39 維。

最後，我的 best model，則是第一種的加強版，放上 4 層的 Bi-LSTM，較深層的 RNN 能使模型具備較大的非線性，而中間的 2 層 Dropout 則提供了 generalize 的能力，每次並非看到全部的資料，才不容易 overfitting。這個模型看起來雖然很精簡，但我其實嘗試過許多更複雜的架構，但結果都沒有比較好，所以最後才又選擇這個訓練起來不會太久，效果也好的模型，至於失敗的經驗會在下個部分提到。

Predicting and Proprocessing

預測的第一步當然是先讀取訓練好的模型進來，對測試資料做上面的預處理，如果有需要的話 reshape testing data 為設定好的 n_timesteps。再來就可以用模型預測 testing data 的 labels 了，得到結果 y_predict，這時候的每個 result 都是 softmax 後的 39 維向量，最大的值表示其預測出的類別。

第二部分是對 y_predict 做後處理，以輸出成我們所希望的格式(.csv)。首先將 y_predict 做 reshape 和 truncat(去除 padding)，找出每個 39 維向量中的最大值和其索引值，然後 one hot 化 39 維向量，使最大值為 1 其他則是 0，再用預處理的 LabelBinarizer 將它們反向轉換回對應的 phones labels，再轉成其對應的 abcd 等字元。接下來，我對每個結果做 thresholding，把原本最大值小於某個門檻的都去掉，在這裡我預設為 0.7。刪掉所有不想要的數值後，我把屬於同一個人和句子的字元接起來，去除頭尾的 L（也就是 sil）和壓縮連續相同的字元成一個，最後就能寫成 csv 檔了。

How did I improve the performance of my models?

一開始的 RNN 模型預測結果不盡理想，雖然可以訓練到 93%左右的精確度，但 validating 的精確度卻只有 80%。而前面加上 CNN 後的模型，後來發現結果沒有相差多少，在 Kaggle 上的評分只進步不到 0.5 分。

後來借鏡之前在 HW0 做 Fashion MNIST 時的模型，那時我拿到滿高的準確度是用 Conv2D*2+Dropout+Conv2D*2+Dropout 這樣的架構，模仿一樣的方式，我用兩層 Bi-LSTM 接 Dropout 再兩層 Bi-LSTM 接 Dropout，也就是我的 best model 的架構，經測試後發現準確度上升許多，而訓練到最後的 training 和 validating 的準確度為 92%和 95%。然後我嘗試了很多不同的參數，比如不同的 LSTM hidden numbers、activation functions、validating rate、batch size、dropout rate 等等，放著機器訓練各種模型好幾天，但是都沒有顯著的進步，我才發現這些參數應該是調整的最後一步。幸好其中 timesteps 還是比較有影響力，我最後的模型都設 123，看似隨便打的數字，實際上比起 32、64、144 等長度，123 做出的結果是最好的（100~130 都差不多好）。

另一個重大躍進的是，我加入了 thresholding 機制，在後處理的時候把最大值小於某個門檻的結果都直接去掉，雖然在訓練時的 training 和 validating 準確度都是 92%和 95%左右，但是 Kaggle 上的分數卻大大躍進了 3 分。我認為這是因為評分機制使用 edit distance 加上移除了連續的相同字元，若有一點雜訊參雜在一個序列中，對最後的評分會有很大的影響，比方說應是 aaaa 的序列被預測為 aaba，最後比對結果時會是 a 比對 aba，一下就差了兩個 edit。

最後我也試驗了不同的結構的模型，2 層 Bi-LSTM 的效果比 best model 差，6 層 Bi-LSTM 的則是差不多，但是訓練時間是原本的 1.5 倍左右(700 秒)，我也嘗試 CNN 用不同的方式接，用 1 顆 Conv2D，或是用 512 顆再 Dense 回 39 維向量接 Bi-LSTM，都難以使結果更好。而我也是過在中間的 RNN 結束到 softmax 的 Dense 之間放一兩層 Dense，看看能不能學到多點非線性的關係，但是結果也沒有比較好。

此外，我在試不同 activation function 時，犯了一個有趣的錯誤，就是把 relu 和 selu 拿來做為 Bi-LSTM 的 activation，當時發現到不管怎麼訓練，每個 epoch 的準確度都是 0.1974，也有 0.0415 的，從第一個到最後都不變，以為是模型哪裡接錯了，結果發現是這類型的 activation function 不適用於 RNN，因為 RNN 會迭代無數次，relu 和 selu 在輸入大於零時是線性的，會被放大到無限大而 dominate 整個結果，造成相當嚴重的預測偏差，因此 RNN 只能用 tanh(-1~1)或 sigmoid(0~1)這類有限範圍的函式，或者要對 relu 和 selu 設置 maximum 才能進行有效的訓練。

Experiment Settings and Results

我的實驗環境是：

- CPU：Intel Xeon 處理器 E3-1230 v3
- RAM：8G
- GPU：GeForce GTX 980
- OS：CentOS Linux 7

使用的工具為 Python 3.6.2 with TensorFlow 1.3.0 and Keras 2.0.7。

將預測結果放上 Kaggle 測試的分數為：

- RNN 模型：
  private<10.81204>、public<11.00564>
- CNN 模型：
  private<10.53493>、public<10.96610>
- Best Model：
  private<7.73975>、public<7.77401>

我的 RNN 模型和 CNN 模型每個 epoch 約花費 200 秒，而 best model 則需要 450 秒左右，每次做一個完整 round 需要至少一個半小時，相當花時間，因此我設定了 CheckPoint callback function 讓每次 epoch 結束時，把 model 儲存下來，以便我最後找出最好的 epoch 數。實驗結果發現：對於我的 best model，在 9~13 epochs 時就趨近上限，validating 準確度不再升高，此時預測的準確度最高，繼續訓練下去只是讓 model overfitting 而已，反而準確度會下降。而這個數值對於簡單或複雜的模型都不一樣，越簡單的模型越快到最佳狀態，RNN 模型和 CNN 模型差不多在 6~8；越複雜的則越久，試過使用 6 層 Bi-LSTM 的模型花了 13 個 epoch 才飽和。

透過這次的實驗學到很多關於深度學習應用的東西，做出一個好的模型需要考慮的地方太多了，希望之後會越來越熟悉這些不同模型的性質，也很期待前幾名的同學來分享他們是如何做出準確的模型的。