ADL x MLDS Fall 2017

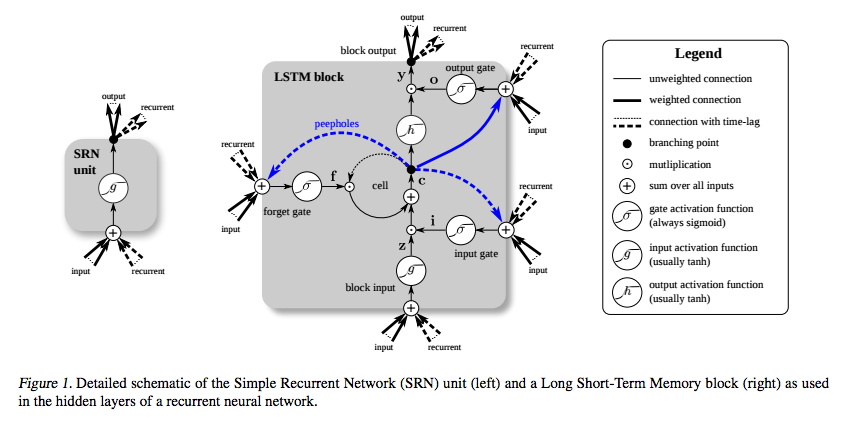
Homework 1 Report

Student: CSIE R06922068 林裕景

# Introduction

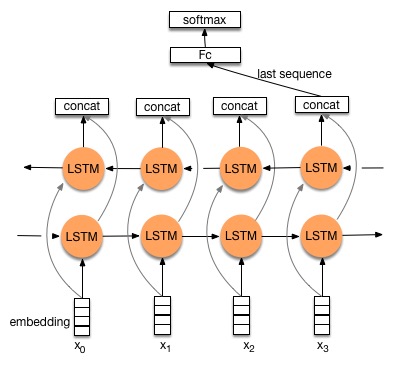
　　這次作業的目的是藉由TIMIT數據集的語音資料，利用深度學習建立能判斷其語音(phone)的模型。助教已經先對原始音檔進行feature extracting by frame，透過moving window方法擷取小段時間內的音訊特徵，提供了mfcc和fbank兩種不同方法取出的features，分別是39維和69維，而每個frame也都有標上標籤表示屬於哪種語音(共48種)。

　　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**  rnn |  |
| **CNN Model**  cnn | **Best Model**  best |

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。