ADLxMLDS HW1: Sequence Labeling

學號：R06922030　系級：　資工碩一　姓名：傅敏桓

I.　模型描述與前處理

本次作業使用 Keras 2.0 來建立神經網路模型，訓練時從 3969 筆資料分出最後 10% (370 筆) 作為驗證集，取每個幀 (frame) 的 MFCC、FBank 的串連向量 (dim=108) 為特徵向量。每筆資料取固定時間長度 400 幀，並對長度不符的資料做前方補 0 或截短的處理；對於訓練集的標籤也做同樣處理，給補 0 的部分新的分類標籤，將訓練標籤的 49 個類別用 49 維的 one-hot 向量來表示。前處理完成後得到大小為(3969, 400, 108)的訓練資料以及大小為(3969, 400, 49)的訓練標籤。批大小 (batch_size) 統一設定為 64。

1. RNN 模型

Keras 的 RNN 模型需要三個軸度的輸入：批大小 (batch_size)、時間步數 (timestpes) 及特徵維度。首先讓輸入通過依時間步數輸出的全連接層 (使用 TimeDistributed 包裝器)，通過一層全連接層後再通過依時間步數輸出 (return_sequences=True) 的遞迴層，接著通過幾層全連接層、適度套用 Dropout 後通過 softmax 函數得到大小為(batch_size, 400, 49)的輸出向量。詳細的模型結構如下左圖所示。為了避免梯度消失或爆炸的問題，除了輸出使用 softmax 函數、輸出前一層使用 sigmoid 函數作為激發函數外，其餘層皆使用 ReLU 函數。訓練時透過 Adam 進行參數更新。模型結構如左下圖。

2. CNN + RNN 模型

我們期望 CNN 能掌握資料整體的局部特徵。考慮到卷積核在時間維度上橫移 (同個卷積核考慮不同時間範圍) 可能效果有限，使用的是 1D 卷積層，在原有的模型上直接在遞迴層後面多加入一層 1D 卷積層做成了新的模型如右下圖。這裡使用的卷積核大小是 5，也就是同時考慮該時間點與前後各兩個幀的特徵。邊界的地方就補 0 讓輸出的時間步和輸入相同。這樣得到的模型在 Kaggle 分數上比原先的模型表現略好一些。在其他實驗中加入兩層卷積層效果似乎可以更好。兩種模型的結果比較會在 III 討論。

Layer (type)	Output Shape	Param #
time_distributed_1 (TimeDist	(None, 400, 512)	35840
dropout_1 (Dropout)	(None, 400, 512)	0
dense_2 (Dense)	(None, 400, 256)	131328
dropout_2 (Dropout)	(None, 400, 256)	0
simple_rnn_1 (SimpleRNN)	(None, 400, 256)	131328
dense_3 (Dense)	(None, 400, 128)	32896
dropout_3 (Dropout)	(None, 400, 128)	0
dense_4 (Dense)	(None, 400, 128)	16512
dense_5 (Dense)	(None, 400, 128)	16512
dropout_4 (Dropout)	(None, 400, 128)	0
dense_6 (Dense)	(None, 400, 64)	8256
dense_7 (Dense)	(None, 400, 64)	4160
dense_8 (Dense)	(None, 400, 49)	3185

RNN

Layer (type)	Output Shape	Param #
time_distributed_1 (TimeDist	(None, 400, 512)	35840
dropout_1 (Dropout)	(None, 400, 512)	0
conv1d_1 (Conv1D)	(None, 400, 256)	655616
dense_2 (Dense)	(None, 400, 256)	65792
dropout_2 (Dropout)	(None, 400, 256)	0
simple_rnn_1 (SimpleRNN)	(None, 400, 256)	131328
dense_3 (Dense)	(None, 400, 128)	32896
dropout_3 (Dropout)	(None, 400, 128)	0
dense_4 (Dense)	(None, 400, 128)	16512
dense_5 (Dense)	(None, 400, 128)	16512
dropout_4 (Dropout)	(None, 400, 128)	0
dense_6 (Dense)	(None, 400, 64)	8256
dense_7 (Dense)	(None, 400, 64)	4160
dense_8 (Dense)	(None, 400, 49)	3185

RNN + CNN

II. 嘗試改善模型的方法

改善模型所做的嘗試與改變

1. 最初嘗試的模型

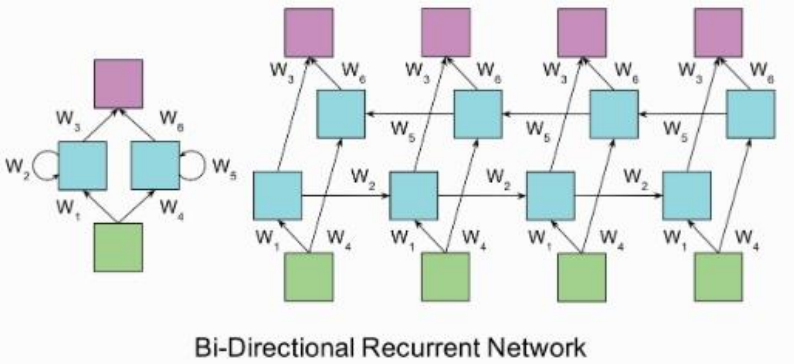
   最初是先將資料透過固定大小的滑動窗取每個時間步以及前後的特徵去和對應的標籤對應成一筆資料，如此每筆資料都只有一個對應標籤，資料總數就是音標（phone）的總數，然後把整個訓練資料集的句子全部連接後展開。然而這樣的模型並沒有太好的表現，可能是因為把句子連起來之後模型在句末句首處學到錯誤的時間性特徵所致，後來才改成使用整句截短或補 0 的方法並調整模型。

2. 使用屏蔽（Masking）層：

   屏蔽層可以用在 RNN 類的神經網路上，讓模型忽略特徵全部為 0 的時間步。會嘗試加入屏蔽層是因為按照上述的資料前處理方式會讓部分資料的開頭補 0。由於這次實驗中加入屏蔽層結果似乎沒有太大的進步，且屏蔽層無法和卷積層共用，所以最後沒有採用。

3. 使用雙向 RNN 模型：

   單向 RNN 的問題是對於某時刻的資料我們只能利用該時刻之前的訊息，但是在分類問題上也有可能會使用到未來的訊息。雙向 RNN 在任意時刻都保持兩個隱藏層，一個記錄由左至右的訊息傳遞，另一個記錄由右至左的訊息傳遞，希望這樣的架構可以利用更多的訊息而有更好的表現。雙向 RNN 可以透過 Keras 提供的 Bidirectional 包裝器來實作。



(圖片取自 https://feisky.xyz/machine-learning/rnn/)

4. 使用 GRU 替代普通的 RNN 層：實驗發現，在 RNN 的變形中，雙向的 GRU 表現最好。
5. 使用批標準化層(BatchNormalization)：實驗發現，對批進行標準化後結果有些微的進步。

其他在作業中使用的技巧

1. 改變修整輸出序列的方式：

   本次作業是以編輯距離作為誤差計算方式，想到微調修整輸出序列的方法可能可以改善結果。觀察訓練集的標記不難發現標記大多是 3 個以上連續出現的，也就是說輸出序列中單獨出現的標記極有可能是預測誤差，於是把連續出現不超過 3 個的標記直接丟棄，在結果上有顯著的進步，同樣模型的誤差最多可以差到約 15.08 分。(29.22->14.14)

2. 改變替換音標標記的時機：

剛開始是直接把標記對應到 39 種再下去訓練，後來改在最後才做標記對應的操作，同樣模型上後者在 Kaggle 分數上有約 1.11 分的進步(14.14-> 13.03)。推測可能的原因是：原先訓練集中有不同標記的音標之間特徵一定有所差異才會被分到不同類別，硬把這些有差異的音標歸到同一類可能會讓分類變得不太準確。

3. 使用的特徵對結果的影響

分別比較只使用 MFCC 特徵、只使用 FBank 特徵和同時使用兩種特徵串接 (MFCC + Fbank) 在同樣的模型上實驗的結果，表現是 MFCC + Fbank > Fbank > MFCC，最後使用 MFCC + Fbank 特徵訓練模型。

III. 實驗設定與結果

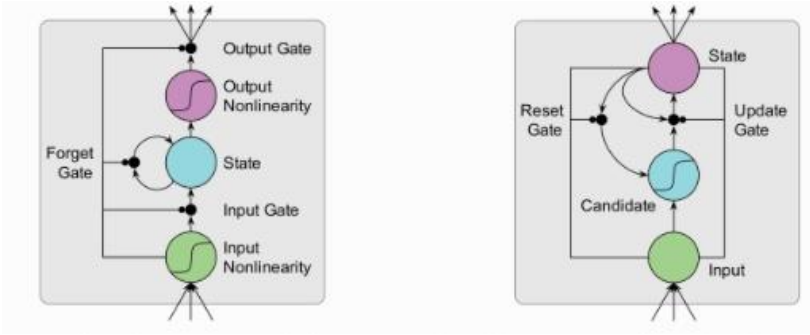
1. 前處理：以 I 所述之方法得到大小為 (3969, 400, 108) 的訓練資料和 (3969, 400, 49) 的訓練標籤。
2. 比較 RNN 與 CNN 之差異

此處使用的模型即 I 所述之 RNN 模型與 CNN 模型。將原有的 RNN 模型架構加上一層 1D 卷積層後，發現新模型的表現比原先更好，誤差分數可以降低約 1 分左右，而訓練模型的花費時間並沒有增加太多。推測進步的原因可能是加了卷積層後讓模型更能掌握資料整體的時間性局部特徵。兩個模型訓練 100 個 epochs 後在 Kaggle 上的得分如下：

	Public Score	Private Score
RNN	12.27118	12.91325
CNN + RNN	10.85875	11.15421

3. 比較各模型之間的差異

在這次實驗中分別加入單層的雙向 RNN (SimpleRNN)、雙向 GRU 以及雙向 LSTM，發現在 RNN + CNN 的模型架構下 GRU 的表現最好，而且兩層卷積層的效果比一層的略好。GRU 是 LSTM 的一種變形，參數比 LSTM 少而普遍被認為可以達到和 LSTM 差不多的效果，而這類遞迴層可以較有效地處理和預測時間序列中間隔和延遲較長的重要事件。以下是 LSTM 和 GRU 神經元的比較。



(圖片取自 https://feisky.xyz/machine-learning/rnn/)

下表是各模型的實驗結果比較，實驗設定都是以上述之 RNN 模型為基礎，每個模型各跑 100 至 200 個訓練 epochs。另外，本來預期 LSTM 會有最好的結果，但加入卷積層後的 LSTM 不知道為什麼模型訓練不起來，再加上模型的訓練時間過長，最後就只有列出 RNN 和 GRU 的比較。

	Public Score	Private Score
RNN	12.27118	12.91325
CNN + RNN	10.85875	11.15421
CNN + 雙向 RNN	10.63276	11.10120
CNN + 雙向 GRU	9.97740	10.08674
兩層 CNN + 雙向 GRU	**8.68351**	**9.04096**

文末附上最終採用的模型。使用單層雙向 GRU、兩層卷積核大小為 9 的卷積層，前後各通過數層全連接層，訓練約 200 個迭代，在 Kaggle 上得到的分數是 (Public, Private) = (8.98554, 9.04096)。

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed_1 (TimeDist (None, 400, 512)          55808
_________________________________________________________________
dropout_1 (Dropout)          (None, 400, 512)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 400, 512)          2359808   <== CNN
_________________________________________________________________
dense_2 (Dense)              (None, 400, 512)          262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 400, 512)          0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 400, 512)          2359808   <== CNN
_________________________________________________________________
dense_3 (Dense)              (None, 400, 512)          262656
_________________________________________________________________
dropout_3 (Dropout)          (None, 400, 512)          0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 400, 512)          1181184   <== GRU
_________________________________________________________________
dense_4 (Dense)              (None, 400, 256)          131328
_________________________________________________________________
dropout_4 (Dropout)          (None, 400, 256)          0
_________________________________________________________________
dense_5 (Dense)              (None, 400, 128)          32896
_________________________________________________________________
dense_6 (Dense)              (None, 400, 128)          16512
_________________________________________________________________
dropout_5 (Dropout)          (None, 400, 128)          0
_________________________________________________________________
dense_7 (Dense)              (None, 400, 64)           8256
_________________________________________________________________
dense_8 (Dense)              (None, 400, 64)           4160
_________________________________________________________________
dense_9 (Dense)              (None, 400, 49)           3185
=================================================================
```

參考資料

https://feisky.xyz/machine-learning/rnn/