HW2 TIMIT Report

學號:R06942074
系級: 電信所碩一
姓名:李宇哲

Model description:

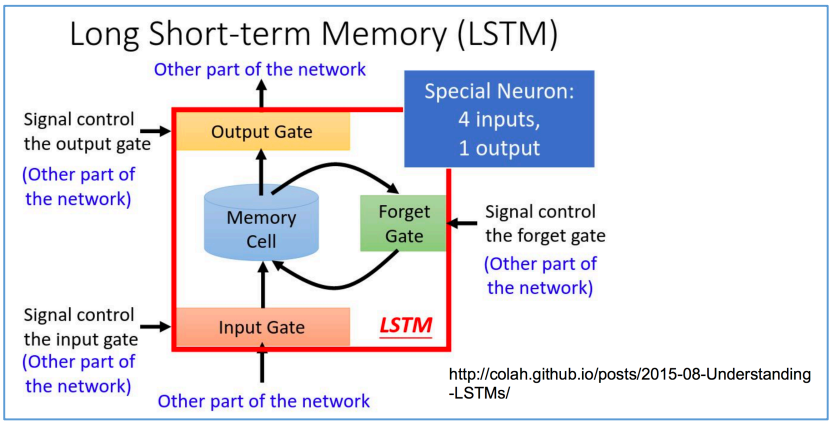
Used platform: Keras

Used dataset: mfcc

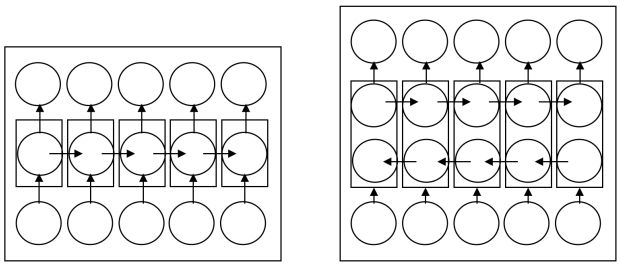
關於這次的作業，我們需要實作 RNN 和 CNN+RNN 兩種 model 去預測不同 frame 的 phone type。

首先先講我 RNN 的架構:

我的 RNN model 主要是採用 LSTM：



因為 LSTM 的 forget gate 有解決一般 RNN 在 learning long-term 時會發生 vanish gradient 的問題，所以如果說今天 model 的 time size 太大可能就還是能夠學到前面的 weight。再者，我使用 Bi-direction 的 RNN 架構:

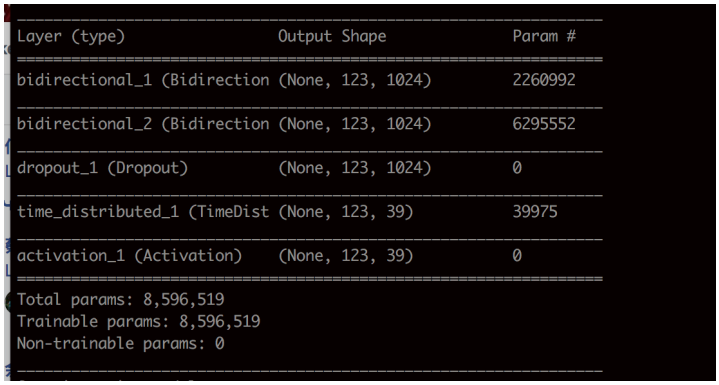


Structure overview
(a) unidirectional RNN
(b) bidirectional RNN

這裡可以發現，如果原本是左邊的架構的話，那下一個 gate 的預測結果只會跟前一個 gate 有關，不過如果是利用 bidirectional RNN，那麼每一個 gate 的預測結果就會跟上一個 gate 和下一個 gate 有關。這樣可以讓 phone 的時序性的強

度增加。

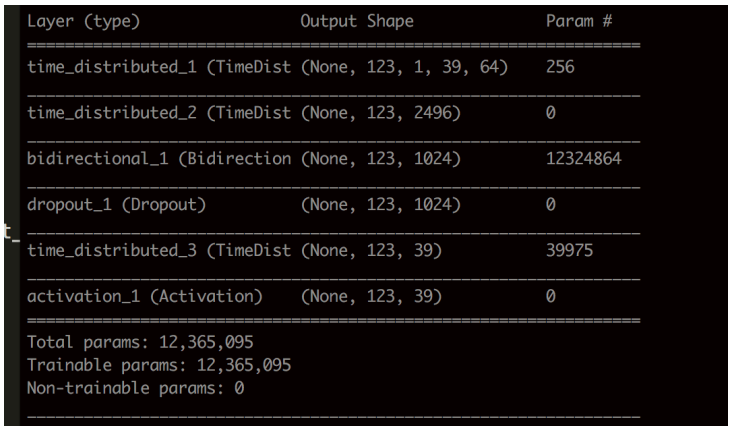
我的 model 架構如下：



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_1 (Bidirection (None, 123, 1024)         2260992
_________________________________________________________________
bidirectional_2 (Bidirection (None, 123, 1024)         6295552
_________________________________________________________________
dropout_1 (Dropout)          (None, 123, 1024)         0
_________________________________________________________________
time_distributed_1 (TimeDist (None, 123, 39)           39975
_________________________________________________________________
activation_1 (Activation)    (None, 123, 39)           0
=================================================================
Total params: 8,596,519
Trainable params: 8,596,519
Non-trainable params: 0
_________________________________________________________________
```

我前面接了兩層 Bidirectional 的 LSTM，然後 input time step size 給了 123（picked by myself），而不是以最大的 data 長度當作 time step size。希望可以間接解決 long term 時 vanishing gradient 的問題。然後我的每一筆資料不做 truncate 只做 padding，也就是說每一筆音訊資料會按造 123 切成好幾個 training data，所以會發生一個 frame 裡面有多筆 training 資料而非只有一筆。

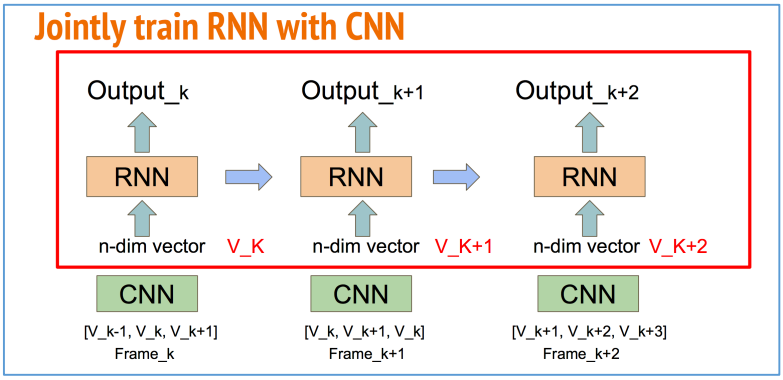
RNN+CNN 的架構如下:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed_1 (TimeDist (None, 123, 1, 39, 64)    256
_________________________________________________________________
time_distributed_2 (TimeDist (None, 123, 2496)         0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 123, 1024)         12324864
_________________________________________________________________
dropout_1 (Dropout)          (None, 123, 1024)         0
_________________________________________________________________
time_distributed_3 (TimeDist (None, 123, 39)           39975
_________________________________________________________________
activation_1 (Activation)    (None, 123, 39)           0
=================================================================
Total params: 12,365,095
Trainable params: 12,365,095
Non-trainable params: 0
_________________________________________________________________
```

簡單來說我在前面多加上 CNN 的 model

kernel size = (3,1) 64 個 filters，在把資料餵進 model 之前會先把每筆資料與前一筆和後一筆接起來，然後放進 CNN，按造講義的方式與 model 連接



在 RNN 和 CNN 在 kaggle 的 performance 的比較：

	Private score	Public score
RNN	8.65542	8.65542
RNN+CNN	10.17590	10.17590

可以發現加上 CNN 並沒有提升 model 的準確率，還使 model 的 performance 下降

How to improve performance：

這裡要開始討論我的 best model 是如何設計的，首先因為從前面的實驗得知 RNN 的架構比較好，所以我直接不採用 CNN 的架構，再來我原本是用兩層 Bidirectional LSTM 現在是用 4 層希望可以增加準確度，再來還用了 drop 解決 overfitting 的問題。

我的改進如下:

1. 2* Bi-LSTM → 4*Bi-LSTM
2. 2* Dropout = 0.5
3. Timesteps: 123 → 777

所以 model 會變成如下：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_1 (Bidirection (None, 777, 1024)         2260992
_________________________________________________________________
bidirectional_2 (Bidirection (None, 777, 1024)         6295552
_________________________________________________________________
dropout_1 (Dropout)          (None, 777, 1024)         0
_________________________________________________________________
bidirectional_3 (Bidirection (None, 777, 1024)         6295552
_________________________________________________________________
bidirectional_4 (Bidirection (None, 777, 1024)         6295552
_________________________________________________________________
dropout_2 (Dropout)          (None, 777, 1024)         0
_________________________________________________________________
time_distributed_1 (TimeDist (None, 777, 39)           39975
_________________________________________________________________
activation_1 (Activation)    (None, 777, 39)           0
=================================================================
Total params: 21,187,623
Trainable params: 21,187,623
Non-trainable params: 0
_________________________________________________________________
```

每兩層 LSTM 就會接 dropout。

Experimental results and setting:

	Private score	Public score
RNN	8.65542	8.65542
RNN+CNN	10.17590	10.17590
RNN(4 LSTM)	7.83373	7.83373

這裡可以發現新的 model 的 performance 明顯大幅提升，由此可以推論讓 NN 的 layer 更深可以讓預測能力變強。
不過我想說我的 model 應該還有增強預測能力的空間？所以這次的 output answer 最後要經過 Trimmimg on Framewise 時，應該可以利用 output 的機率去決定要不要採用，舉例如下：

predicted phones: {sil, sil, a, a, b, c, c, c, d, a, a}
predicted prob: {0.6, 0.9, 0.8, 0.7, 0.8, 0.96, 0.5, 0.4, 0.9, 0.6, 0.8}
如果將小於 0.7 的拿掉會變成:
predicted phones: {sil, a, a, b, c, d, a}
這樣的好處是可以讓 model 在不確定的時候，採用 truncate 掉的方法，因為就算 truncate 掉機率低的 phones，在 trimming 的時候其實反而不容易讓同樣類型的 phone 消失，也能減少誤判的機率。

	Private score	Public score
RNN(4 LSTM)	7.83373	7.83373
RNN (4 LSTM with threshold)	7.02168	7.40677

所以最後 RNN with 4 bi-LSTM+777timesteps+Threshold 變成我的 best model