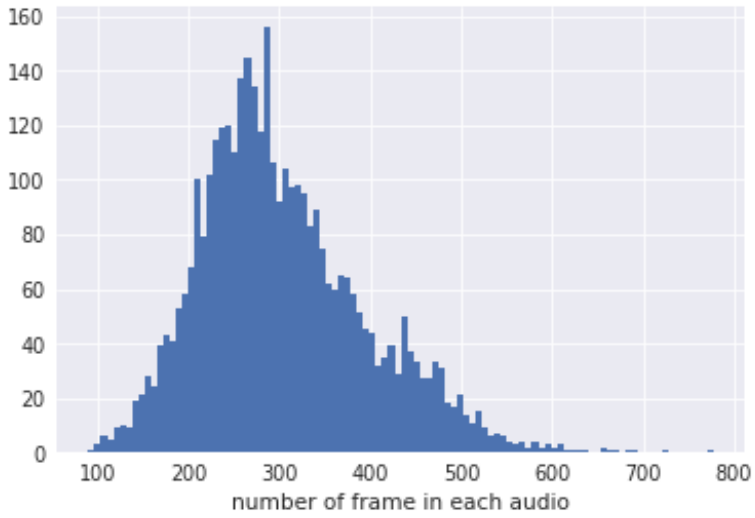
R06946003 湯忠憲 ADL HW1-TIMIT Sequence Labeling

Model description

RNN

- Preprocessing:

  由於 RNN 需要給定三個維度，(batch size, frame size, feature size)，而其中，frame size 也是固定的，所以需要切固定數量的 frame 給單一 sample。這裡我用 200 個 frame 當一個 sample，同一音檔有可能會用到兩至三個 sample (即超過 200 個 frame，相關統計可見圖一) ，因此，在同一音檔，sample 跟 sample 之間有用 overlap 的方式讓 RNN 學到連續的特徵。特徵的部分則是將 mfcc 跟 fbank concatenate 起來變成 108 維的 feature。最後 input shape 就是(batch size, 200,108)。Batch size 的部分經實驗後發 128 表現比較穩定。



圖一、音檔 frame 個數統計

- Model structure:

  圖二是我的初始版的 RNN 架構，用兩層 LSTM 接 2 層 fully connected layer，最後輸出 49 維的 array (48 phone + 1 zero padding class)，optimizer 用 Adam。經過 100 個 epochs 後，kaggle 上的 Levenshtein distance 約為 12.3551。

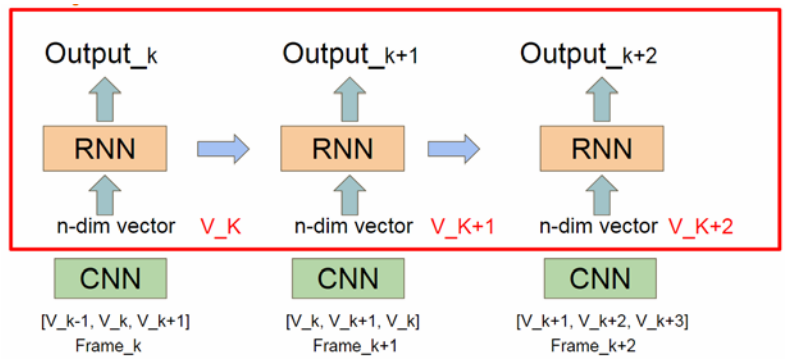
```
model = Sequential()
model.add(LSTM(input_shape = (200,108),units=64,return_sequences= True))
model.add(LSTM(units=128,return_sequences= True))
model.add(Dense(512,activation="relu"))
model.add(Dense(49, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

圖二、RNN 程式架構

CNN+RNN

- Preprocessing:

  CNN+RNN 的架構則是根據助教投影片的說明建構的(如圖三)。因此原本一個 frame 是用 1*108 維度的特徵來表示，現在則是 3*108 的一個二維陣列來表示。所以 inpute shape 變成 (batch size, feature size, feature size, number of lookup frame)。CNN 掃的方向則是與時域垂直，因為時域的在之後的 RNN 終能被學到。



圖三、CNN+RNN 模型 (助教 PPT)

- Model structure:

  模型中 CNN 跟 RNN 是 Jointly train，因此我用 Keras 中，TimeDistributed 的 function 讓每一個進到 RNN 中的 sample 的 frame 都個別經過 CNN，以符合圖三中的架構。CNN 層則是用 1D 的 Convolution layer，因為我們只在單一方向掃 feature。之後則是接兩層 RNN 和兩層 fully-connected layers，optimizer 用的一樣是 Adam。

```
model = Sequential()
model.add(TimeDistributed(Conv1D( filters=16, kernel_size=6),input_shape =(200,108,3)))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(units=64,return_sequences= True))
model.add(LSTM(units=128,return_sequences= True))
model.add(Dense(512,activation="relu"))
model.add(Dense(49, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

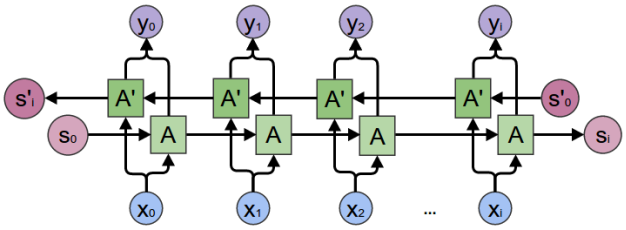
圖四、CNN+RNN 程式架構

經過多次實驗，包含調整 filter 數量、kernel_size 等等，發現這樣的作法對 kaggle 上的表現沒有太大的幫助，Levenshtein distance 約為 12.4871。

How to improve your performance

為了讓 performance 更好，我主要嘗試了五項 techniques 提升效果:

1. ***Bidirectional LSTM:***
   雙向 LSTM 可以從兩個方向看 frame data，可以學到更堅實的前後關係，概念則如圖四。



圖五、Bidirectional RNN (https://github.com/huseinzol05/Generate-Music-Bidirectional-RNN)

   其 output 得維度會是單項 RNN 的兩倍。

2. ***Dropout:***
   Dropout 在這裡也可以發揮不錯的效果，該方法可以讓每個 neuron 更強一點，最後再用 ensemble 的方式，平均 neuron 的 output。這樣可以防止模型 overfitting。此外，將 Dropout rate 調的 aggressive 一點，例如 0.5、0.3，可以有效提升表現。

3. ***增加 CNN 看到的 frame 數量:***
   原本使用 3*108 的二維陣列給 CNN，試著增加前後 frame 的個數可以讓 CNN 一次看到更多資訊，同時調整 filter 和 kernel size。

4. ***BatchNormalization:***
   在 activation 前後增加 BatchNormalization 層。

5. ***設計 Decoder:***
   透過觀察和統計 label 出現分佈(如圖五)，可以發現全部的 phone 都會至少重複三次，因此 prediction 在 decode 後如果只有單獨一個或者兩個 phone 連在一起，都將他 decoder 程後面的 label (等同直接砍掉)。實際實驗後發現即使是三連音砍掉都還能提升效果。

```
label list: ['sil', 'sil', 'sil', 'sil', 'sil', 'sil', 'sil', 'sil', 'sil', 'sil', 'sil', 'sil', 'ax', 'ax', 'ax', 'ax', 'ax', 'ax', 'ax', 'a
x', 'ax', 's', 's', 's', 's', 's', 's', 's', 's', 's', 's', 's', 's', 's', 's', 's', 's', 's', 'uw', 'uw', 'uw', 'uw', 'uw', 'uw', 'uw', 'uw',
 'm', 'm', 'm', 'm', 'm', 'm', 'm', 'm', 'm', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'ao', 'ao', 'ao', 'ao', 'ao',
 'r', 'r', 'r', 'r', 'r', 'r', 'ix', 'ix']
```

圖六、label list

Experimental results and settings

Model	Basic RNN	Basic CNN	RNN+Dropout	CNN+Bi-Lstm+Dropout	Bi-Lstm + Dropout
Levenshtein distance	12.3551	12.3551	11.16098	8.5539	7.904895

表一、各模型表現

- Compare and analyze the results between RNN and CNN

  實作兩種模型，並且多次修改模型層數後發現兩個模型得到的效果類似，(兩種的 RNN 均用 LSTM 的情況下)。有可能是 LSTM 在本次 dataset 上已經能有很強的表現，將 CNN 的效果稀釋了。若將 LSTM 換程 simpleRNN 的話可以發現兩個模型中，CNN+RNN 會有較好的效果。

  此外，若將 CNN+RNN 的模型中加上 aggressive 的 Dropout 和使用 Biderictional LSTM 後，Levenshtein distance 可以降低到 8.5539 (平均)，這也是我在 kaggle 上最後成績。

  模型程式碼如下圖:

```
model.add(TimeDistributed(Conv1D(filters=16,
                                 kernel_size=5,
                                 activation="relu"),
                                 input_shape =(frame_count,108,3)))
model.add(TimeDistributed(Flatten()))
model.add(Bidirectional(LSTM(units=128, return_sequences= True)))
model.add(Dropout(0.3))
model.add(Bidirectional(LSTM(units =64, return_sequences= True)))
model.add(Dropout(0.5))
model.add(Bidirectional(GRU(units=32, return_sequences= True)))
model.add(Dropout(0.5))
model.add(Dense(512))
model.add(Dropout(0.5))
model.add(Dense(49, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

圖七、CNN+Bi-Lstm+Dropout

- Compare and analyze the results with other models

除了上述模型之外，寫 report 的過程中意外發現不加任何 CNN 層也可以取得不錯的效果。用 4 層 LSTM 各帶 0.5 的 Dropout rate 可以拿到約 7.904895 (平均) 的 Levenshtein distanc。而 BatchNormalization 在這個模型中沒有特別的效果。

```
model = Sequential()
model.add(Bidirectional(LSTM(units=256,return_sequences= True),input_shape = (200,108)))
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(units=128,return_sequences= True),input_shape = (200,108)))
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(units =64, return_sequences= True)))
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(units =32, return_sequences= True)))
model.add(Dropout(0.5))
model.add(Dense(512))
model.add(Dense(49, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

圖八、Bi-Lstm + Dropout