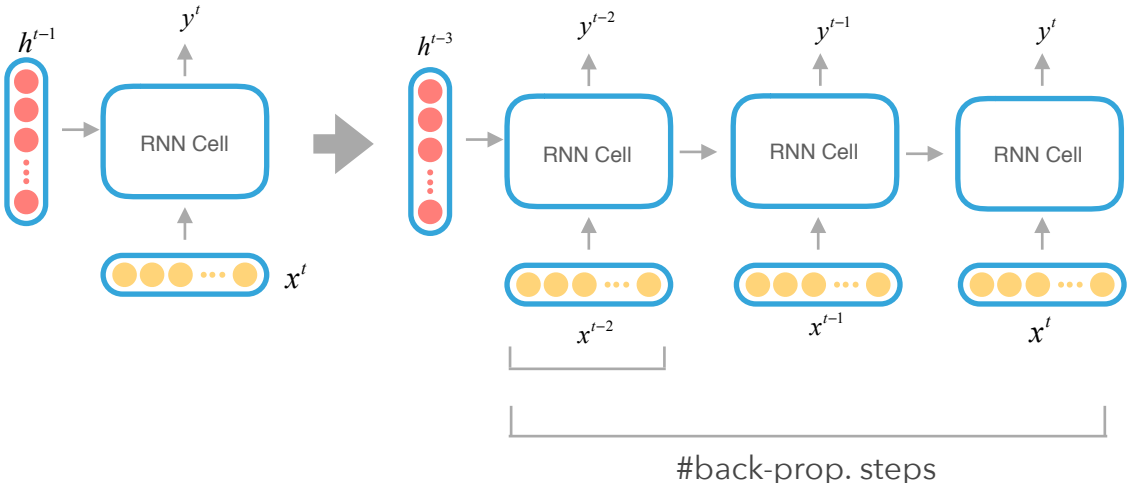
b03901057 詹晉誠

A. Model Description

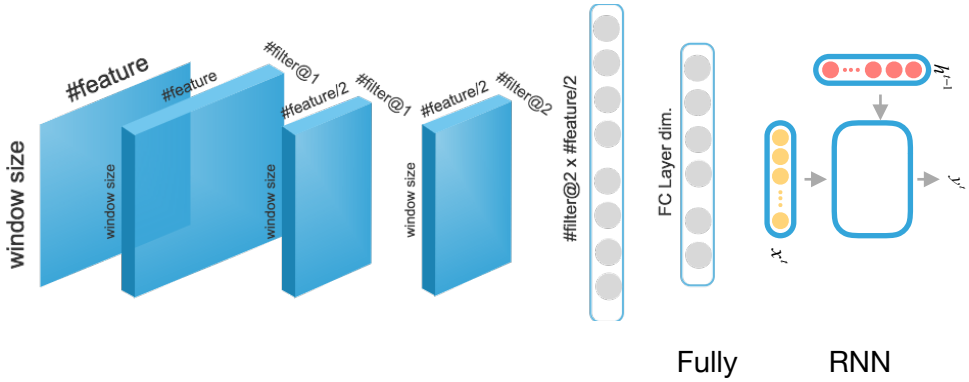
• RNN 架構說明：

在這次的作業中使用的RNN架構基本上如下所述，而基於這樣的基本架構下，我也嘗試過許多不同種的變化，並將於後面章節說明實驗結果。



上圖中顯示了一個基本的RNN cell以及沿著時間軸unfold開來的RNN cell。在RNN中的back-propagation因為計算效率考量，通常不會將最後一個時間的errors propagate到第一個時間，而是會限制propagation的時間長度，因此unfolded RNN只會有固定長數量的RNN cells（圖中的#back-prop. steps，通常為20)。而每一個RNN cell的input是一個feature vector，其長度就是feature的dimension (圖中的#feature) 。使用這樣的network，我一次會將raw data切出長度為#back-prop. steps的小片段，然後將這個 (#back-prop. steps x #feature)的矩陣丟入network進行training。之後network會輸出不同時間的label，我再將不同時間的labels與ground-truth labels進行比較，然後使用cross-entropy作為loss function。此外，實際上我有使用mini-batch的技巧，因此每次input的矩陣是(batch-size x #back-prop. steps x #feature)的三維陣列，而輸出則是(batch-size x #back-prop. steps x #classes)的三維陣列。

• CNN-RNN 架構說明：



上圖顯示了這次作業中使用的CNN-RNN架構。我一開始會將raw data取幾個frame (由window size決定），然後組成一張大小為 (window size, #feature)的image。之後將這張image過第一層的convolution layer，得到一個 (window size, #feature, #filter@1)的三維矩陣，其中#filter@1是第一層的filter數量。之後我會對feature dimension做pooling，得到大小為(window size, #feature/2, #filter@1)三維矩陣。之後再經過第二層convolution layer後我將neurons 攤平，但是這邊要注意的矩陣的大小：假設再flatten前矩陣的大小為(window size, #feature/2, #filter@2)，而我會對2,3維做flatten，因此flatten後的矩陣大小為(window size, #feature/2 x #filter@2)，是一個二維陣列。之後再接上一個fully-connected layer，並將其輸出接到RNN。此外，我也有使用mini-batch的技巧，因此上述的矩陣都要在多加一維（第一維，大小為batch size)，而flatten過後的矩陣是(batch size x window size, #feature/2 x #filter@2)，還是一個2維矩陣。

B. How to Improve the Performance

- **LSTM**

LSTM就像是一個RNN Cell，但是LSTM 可以彈性的控制過去的資料是否會對現在的輸出產生影響，因此效果會比傳統的RNN Cell更好。在這次作業中，因為每個字母的長度可能不一樣（例如母音通常會較長，而b, p之類的音可能就會較短），所以如果可以選擇性的參考過去的input以及state，可以提升辨識的準確率。

- **Bidirectional LSTM**

傳統的RNN (或是LSTM）都只會參考現在跟過去的input（或者說states），而bidirectional LSTM會同時參考現在、過去、未來的input。這樣做的好處是在有些應用

中，未來的input也會和現在的input有correlation。例如在本次作業中，如果只參考過去的input，而現在的這個phone剛好有一些雜訊，現在的這個phone就很有很能被誤判。但如果同時參考過去與未來的input，並發現過去與未來都是同個phone，則有很大的機率現在也是同個phone，因此即使有些雜訊干擾，還是可以從前後的input預測出正確的答案。

- **同時使用MFCC與fbank feature**

不同的feature extractor著重在不同的聲學特徵，因此很難說哪種feature extractor醉於speech recognition來說是最好的。而machine learning的好處就是可能自己學會將兩種features結合在一起的方式。我是將MFCC的39維feature直接concatenate fbank feature的69維feature變為108維的vector，並將此vector當成input，然後再使用CNN做為feature extractor（在CNN+RNN的model中）。因為108還不至於太大，因此還不至於遇到curse of dimensionality的問題。

- **Dropout**

在train deep neural networks時常常遇到overfitting的問題，而overfitting往往會降低model在testing data (unseen data)上的表現，因此我使用dropout來避免overfitting的問題。Dropout會在training時選擇性忽略掉部分的neurons，而因為沒有這麼多parameters，所以也比較不容易產生overfitting的問題。而在testing時則會使用全部的neurons，經過實驗發現，使用dropout的確可以提升在unseen data的表現（Kaggle public set)。

- **Post-processing**

實驗發現少數情況下同一個phon可能只會連續出現一兩次，但實際上每個音不太可能只有1~2個frames，因此那些連續出現次數很少的phone大概就是預測錯誤了。所以我在model predict完後對預測結果進行後處理，也就是把只連續出現一兩次的phone拿掉。經過實驗發現，這樣一個簡單的步驟可以有效提升辨識的準確率。

C. Experiments

- RNN v.s. CNN+RNN

	Double Layer Bidirectional LSTM	CNN + Double Layer Bidirectional LSTM

Input feature	MFCC+fbank	MFCC+fbank
RNN state size	256	256
Dropout ratio @ training	0.1 (applies to RNN cells)	0.5 (applies to fully connected layers)
Batch size	128	128
CNN filter size	N/A	3x3 in each layer
CNN #filter in each layer	N/A	[32, 32, 32]
CNN pooling in each layer	N/A	[2, 1, 1]
Fully-connected layer size	N/A	[640, 512, 256]
#epoch	25	25
Loss @ training	~0.45	~0.45
Kaggle score	**8.4**	**8.6**

本次實驗比較雙層Bidirectional LSTM，與雙層Bidirectional LSTM加上CNN兩個model的表現。實驗中除了Dropout ratio （丟到的neuron 的比例），已盡量控制參數一致。實驗結果可以發現加上CNN後，表現有必較好一些，但是沒有顯著的進步。原因可能是因為input data就已經是抽取過得features，因此CNN沒辦法作為發揮feature extractor。此外另一優勢是可以看到前後的frame，但是因為bidirectional LSTM就已經具有這樣的功能了，所以使用CNN沒有顯著的進步。

• LSTM v.s. Bidirectional LSTM

	CNN + Double Layer LSTM	CNN + Double Layer Bidirectional LSTM
Input feature	MFCC+fbank	MFCC+fbank
#epoch	70	25
Loss @ training	~0.3	~0.45
Kaggle score	**10.64**	**8.6**

本實驗比較LSTM 和 Bidirectional LSTM的表現。沒有列出的參數代表和前面一個實驗相同（RNN v.s. CNN+RNN）。比較結果可以發現使用Bidirectional LSTM雖然沒有讓loss降的更低，但是Kaggle score卻有明顯的進步。之所以可以進步是因為Bidirectional LSTM會考慮前“後”的phone，因此準確率可以較高。同時Bidirectional LSTM也不需要這麼多epoch來train。但同時這個實驗也反應當loss低到0.4時，loss已經不能完全代表recognition accuracy了