## ✧ 題目：*Conversations in TV Shows*

## ✧ *Team name, members and your work division*

- 隊伍名稱：NTU_r06921082_糸勺口馬？？
- 隊伍成員：

| | | |
|---|---|---|
| 隊長 | R06921082 | 陳與賢 |
| 隊員 | R06921084 | 陳治言 |
| 隊員 | R06942077 | 洪建鈞 |
| 隊員 | R06942113 | 林怡均 |

- 分工情況：

| | | |
|---|---|---|
| R06921082 | 陳與賢 | 程式撰寫與模型訓練 |
| R06921084 | 陳治言 | 程式撰寫與模型訓練 |
| R06942077 | 洪建鈞 | 文獻查詢與資料彙整 |
| R06942113 | 林怡均 | 報告製作與進度監督 |

## ✧ *Preprocessing/Feature Engineering*

- 因為總共有 5 個不同的劇本，為了方便訓練，因此我們先將 5 個劇本合成到同一個檔案之中。
- 再來透過 *jieba* 來進行斷詞，為了滿足繁體中文的詞彙，我們使用繁體的中文詞庫 *dict.txt.big*，處理完畢之後，我們會看到同一句話中，會利用空白來分隔不同的詞，並將結果寫到目錄之下來做儲存。
- 為了增加訓練的準確率，我們使用 *stopwords* 來進一步處理，透過這個步驟，我們將中文裡一些語助詞，像是「的」或是「了」等較無意義的字，我們將它們從文章中拿掉。

## ✧ *Model Description*

在訓練的過程之中我們使用二種不同的模型，分別為 $word2vec$ 與 $RNN$ 來做比較。

- $word2vec$：我們在訓練的過程之中，將詞彙的 $embedding$ 的 $size$ 調整為 300，這也代表著我們的詞向量維度將會是 300。再者則是將 $window$ 設定為 30，代表著我們在訓練的過程中，會向前向後參考 30 個詞彙，而 $min\_count$ 則是代表出現次數較低會去除掉，我們將它設定為 12 以避免被太多雜訊干擾，最後我們將訓練次數設定為 30 個迭代，並直接透過 $cos\ similarity$ 的方式來做 $predict$。

- $RNN$：我們在訓練的過程之中，則是先將劇本中每 10 句當作一個 $set$ 來我作訓練，將前 5 句與後 5 句的分別串成 2 句話，成為訓練用的 $data$ 與 $label$。再來，我們透過 $LSTM$ 來拼湊出一個非常簡易的 $seq2seq$ 模型，架構圖如下：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 50, 300)           2962200
_________________________________________________________________
lstm_1 (LSTM)                (None, 1024)              5427200
_________________________________________________________________
repeat_vector_1 (RepeatVecto (None, 50, 1024)          0
_________________________________________________________________
lstm_2 (LSTM)                (None, 50, 1024)          8392704
_________________________________________________________________
time_distributed_1 (TimeDist (None, 50, 300)           307500
=================================================================
Total params: 17,089,604
Trainable params: 14,127,404
Non-trainable params: 2,962,200
_________________________________________________________________
```

這邊我們分開來解釋我們的 $RNN$ 架構：

| | |
|---|---|
| $embedding\_1$ | 在這層的結構中，我們將 $tokenize$ 過後的句子送入這層轉換為 $word2vec$ 得到的詞向量，因為這裡已經預先訓練完成　了，因此我們將它設定為不可訓練的參數，並且設定一句話最多 50 個字彙，且每個單詞的維度 300。 |
| $lstm\_1$ | 這一層我們可以視為 $encoder$ 的角色，將一句話轉換為句向量來表示，且其維度為 1024。 |
| $repeat\_vector\_1$ | 為了滿足下一層的輸入維度，我們將上一層的輸出複製成 50 份當作下一層的輸入。 |
| $lstm\_2$ | 這一層我們可以視為 $decoder$ 的角色，我們要將上個階段得到的句向量來產生一句話，因此需要將 $return\_sequences$ 設定為 $True$，才能產生多個單詞的向量。 |
| $time\_distributed\_1$ | 最後將 1024 維的詞向量 $dense$ 成 300 維，且為了保持相同單詞數量，所以這邊使用 $time\_distributed$ 來做包裝。 |

使用 30 個 $epoch$ 來做訓練，並使用 $RMSprop$ 來當作我們的 $optimizers$。

## ✧ *Experiments and Discussion*

在進行 *testing* 的時候，為了將問題與選項分離出來，先透過正規表示法來做預處理，接下來，使用 *jieba* 來進行斷詞，並使用 2 種不同的 *model* 來進行預測。

- *word2vec*：我們將題目與選項的每個單詞的詞向量分別取出並相加來當作它的分數，之後，使用 *cos similarity* 來匹配每個選項與題目的相似度，將相似度最高當作我們的答案並輸出。

- *RNN*：我們將題目 *tokenize* 後丟入模型去做預測，之後便會得到一句相對應的話，我們將輸出與每個選項取其 *Euclidean Distance*，將距離最短的最高當作我們的答案並輸出。

我們將二種模型分別進行 5 次的訓練以及預測，可以得到下列的 *public* 分數。

| | | | | | |
|---|---|---|---|---|---|
| *word2vec* | 0.37944 | 0.43241 | 0.44940 | 0.41845 | 0.38918 |
| *RNN* | 0.43415 | 0.43241 | 0.43122 | 0.43359 | 0.43280 |

我們發現到如果使用 *word2vec* 來訓練我們的劇本，因為本身是非監督學習的緣故，每次預測的分數差異都很大，沒有一定的取向，在我們訓練的 5 次中，只有一次有超過 *Strong Baseline* 而已。

再來討論到 *RNN* 這邊，在 5 次的訓練之中，其實分數算是都非常接近的，代表這個模型訓練是非常穩定的，但是卻都沒有超過 *Strong Baseline*，我自己認為的問題，我的 *data* 和 *label* 取的不好，造成模型本身上有缺陷，因為劇本其實有些地方也不太通順，因此，如果以取 5 句來做為基準的話，可能會放大資料本身的誤差，而造成訓練上的失敗。

接下來我們討論每次的訓練狀況：

在 $word2vec$ 的所有訓練之中，我們都將 $window$ 設定為 30，我們認為如果能夠看到更遠的詞彙，對於詞向量是會有幫助的。而 5 次的訓練過程之中，不同的設定分別如下：

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $size$ | 250 | 300 | 300 | 350 | 400 |
| $min\_count$ | 10 | 11 | 12 | 12 | 12 |
| $score$ | 0.37944 | 0.43241 | 0.44940 | 0.41845 | 0.38918 |

在 $size$ 的情況：我們一開始是從 250 開始，不過準確率非常的糟，我們認為詞向量太短可能沒辦法訓練到非常精準，因為太短的關係，有些資訊可能會被捨棄掉，而如果太過冗長，反而有可能會變成雜訊而影響 $predict$。因此，我們認為 300 是剛剛好的。

在 $min\_count$ 的情況：我們從系統預設的 10 次開始實驗，我們認為這也是一個需要折衷的數值，如果設的太小，有些很少出現的單詞就會影響訓練，進而變成雜訊，然而如果太大，就會出現訊息流失的情況，造成模型本身就 $underfitting$ 了。在我們的實驗結果之下，認為 12 次是最合適的。

在 $RNN$ 的所有訓練之中，我們第一層 $embedding$ 使用在 $word2vec$ 中表現最好的那個，也就是 $public$ 為 0.44940 的那個模型。並都使用 30 個 $epoch$ 來做訓練，加上 $RMSprop$ 來當作我們的 $optimizers$。而 5 次的訓練過程之中，不同的設定分別如下：

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $set$ | 3 | 4 | 5 | 5 | 6 |
| $embedding\ dimension$ | 35 | 40 | 45 | 50 | 55 |
| $score$ | 0.43359 | 0.43241 | 0.43122 | 0.43415 | 0.43280 |

在 $set$ 的情況：這個代表的是我們取幾句話合併為一句，因為劇本本身有些語意有點問題，我們透過平均分散的方式，來減少些許的誤差。不過，以不同訓練來看，感覺這不是主要影響模型的因素。有可能這個方法本身就不太好的原因。

在 $embedding\ dimension$ 的情況：這個數值是代表句子在 $tokenize$ 過後，為了要進行訓練，我們需要進行 $padding$ 才能讓長度相同。如果在不足的情況之下，會進行填 0 的動作。而超過的情況則會刪除某些詞彙，其實這裡跟上面情況相同，都會出現雜訊亦或是訊息蘊含量太少的問題，以我們實驗的結果，感覺落在 50 是最剛好情況。