Machine Learning – Final Project

TV - conversation

隊伍：NTU_r06922130_CSIElite 戰隊

黃禹程	鄭克宣	葉韋辰	李冠穎
R06944034	R06921083	R06922130	R03922165
Simple baseline 協助者 報告主要彙整者 撰寫部分報告	simple baseline 主要貢獻者 Strong baseline 協助者 撰寫部分報告	Simple baseline 協助者 Strong baseline 協助者 撰寫部分報告	strong baseline 主要貢獻者 嘗試其他主題的 simple baseline 撰寫部分報告

Abstract

因為前後用了兩種完全不同的方法，分別為 word vector 求 cosine similarity（前期做法）以及使用 RNN 作分類（後期做法），因此在這份 report 中，每個章節都會分為前期與後期，各自說明在前期（通過 simple baseline）與後期（通過 strong baseline）的作法。

2 Preprocessing/Feature Engineering

在此章節中，兩個時期的 preprocessing 的作法是差不多的，因此就不分開來解釋。

Preprocessing 的流程大致可分為：

a. 以 Jieba 斷詞

b. gensim 造 word vectors

以下就針對這兩個主要的部分，分別進行細部的流程介紹

以 Jieba 斷詞

以 Jieba 斷詞的流程如下：

1. 以 set_dictionary()設定斷詞字典為 dict.txt (加入字典網址)

2. 在此我們使用了兩種不同的斷詞方式

2.1

對文字檔逐行作斷詞，每行皆會造出包含數個 words 的$list$，亦即 $words = [word_1,\ word_2,\ word_3, \dots]$，最後整個文字檔最後會變成一個 sentences lsit，其紀載為$setences\ list = [words_1,\ words_2,\ words_3, \dots]$

2.2

另一種為將一個文字檔視為一句，直接斷句，也就是 $setences\ list = [word_1,\ word_2,\ word_3, \dots]$，以上和 text8corpus 相似，差別為此為 list，text8corpus 為 generator

3. 檢驗各個 word，若該 word 為{'\n' ,'\t' ,'"' ,'」' ,'「',' '}，則排除該 word4. 替每個文字檔所造出 sentence list 儲存為 pickle 檔，共有 5 個 pickle。

Gensim 造 word vectors

Gensim 造 word vectors 的流程如下：

1. 讀取 word_split.py 所造出的 5 個 pickle 檔，合併為一個$\boldsymbol{list}$Word2Vec()造模型，其中：

I. window = 5：界定 word 與 word 間具關聯性之範圍

II. size = 100：vector 為 100 維

III. min_count = 5：出現次數不小於 5 次的 word 才具相對應之 vector

2. 將 word2Vec 模型儲存

3 Model Description

Cosine Similarity Method (genism)

1. 使用 similarity 比對

   甲、比對對象

   問題和選項的單字一一做 cosine similarity 的比對（也就是假設問題有 m 個單字、選項 A 有 n 個單字，則就作 m * n 次之比對，並紀錄之。

   乙、答案選擇

   對問題和選項各個字詞間的 cosine similarity 取平均，最後選出和題目具有最大平均 cosine similarity 的選項。

2. 使用 n_similarity

   甲、比對對象

   題目的句子對選項的句子算 cosine similarity

   乙、答案選擇

   對最後將問題和選項字詞間的 cosine similarity 取平均，最後選出和題目具有最大平均 cosine similarity 的選項。

RNN

後來使用了 RNN 作 training，採用了「2 Embedding layer & 2GRU(and LSTM)」和「1 Embedding layer & 2 GRU(and LSTM)」

2 Embedding layer & 2 GRU(and LSTM)

訓練資料及標記方式

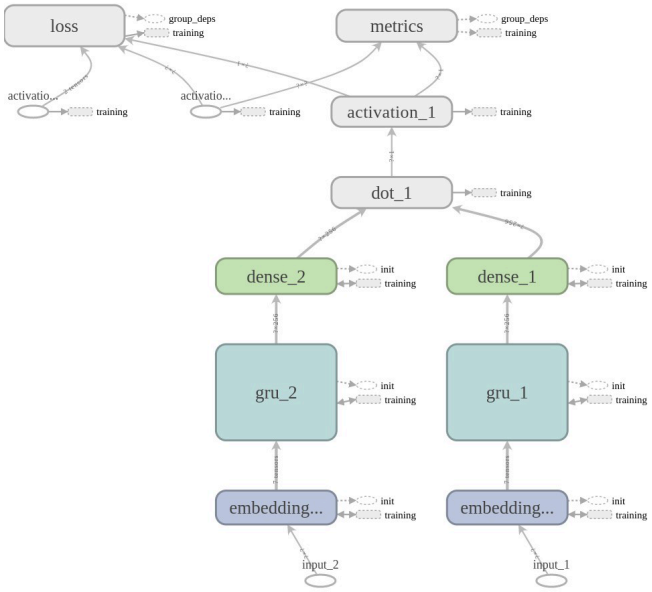
a. 將兩句視為題目，和題目的下兩句視為答案，並標記為 1

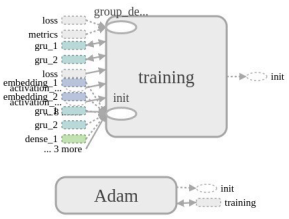
b. 將兩句視為題目，和題目的下 100 句視為假的答案，並標記為 0

模型簡述：

Input 為 (題目，選項)，output 為[0, 1]之值，題目和選項進入各自的 Embedding 和 GRU，最後再 flatten 算出 output

模型架構





1 Embedding layer & 2 GRU(and LSTM)

訓練資料及標記方式

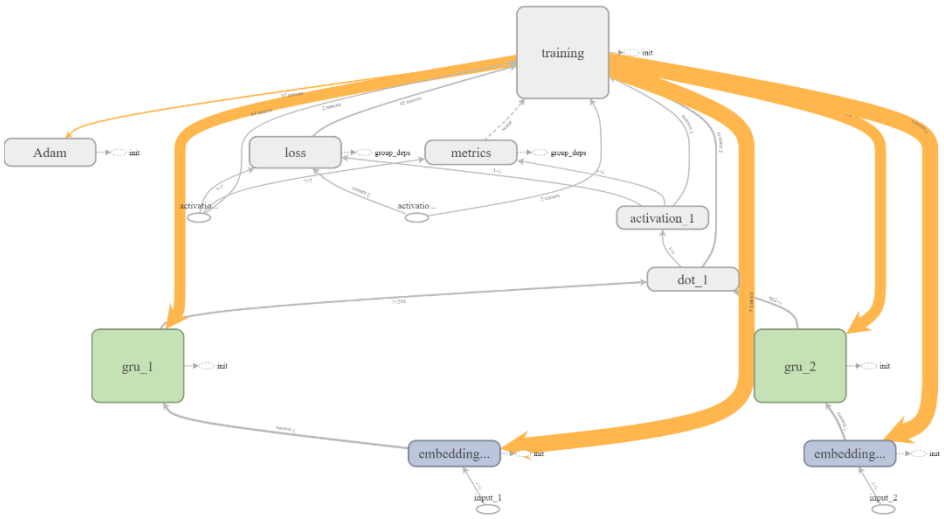
a. 將兩句視為題目，和題目的下兩句視為答案，並標記為 1

b. 將兩句視為題目，和題目的下 100 句視為假的答案，並標記為 0

模型簡述

Input 為 (題目，選項)，output 為[0, 1]之值，題目和選項個共用 Embedding，在各自進入不同 GRU，最後再 flatten 算出 output

模型架構



4 Experiments & Discussions

在此章節中，會各別說明前期過 simple baseline 和 後期過 strong baseline 兩個階段的參數差異。

4.1 前期（simple baseline）參數差異

word vs word / sequence vs sequence

word vs word 與 sequence vs sequence 的差異在於，word vs word 是比較兩個句子中每個 word 交叉比對的相似度累加，舉例來說： A 句子中有 A1,A2 兩個 word ，B 句子中有 B1,B2 兩個 word，則 word vs word 的值是 sum(compare(A1,B1), compare(A1,B2), compare(A2,B1), compare(A2,B2))，而 sequence vs sequence 是直接比較兩個句子的相似度 (wv_model.n_similarity(q_words, a_words)，q_words 為題目句子、a_words 為選項中的句子)。

而後者的作法可以獲得更高的分數，下表顯示這兩個做法的分數：

	word vs word	seq vs seq
Public Score	0.25770	0.35091

4.2 後期（strong baseline）參數差異

1 詞向量訓練

首先使用 Word2Vec 訓練 word vector(dim=150, min_count=1, 其餘皆為 default)

2 訓練資料處理

由於這次問題是選擇題式而不是開放式的 chatbot 問題，因此選擇用類似 retrieval model 的架構來處理。起初有試過 seq2seq model，會在待會的小節做較詳細的說明。

若以 retrieval model 來解問題。另外，因為題目是同學在看過劇本之後所出，因此不需要真的讓機器去理解真的語意(需要做到這件事情，需要的資料量或許不是這次 final 量所可以比擬的，consider how Google and other tech gurus do)，因此或需僅需要知道是否這兩句是否來自同一個文本或主題的即可。

因此我們將每個 text 檔裡頭的句子上下句定義為來自同一個情境，將之 label 為 1。在 negative data 上面，我們取某一句一定距離(distance=100)外的另一句，將這個 pair label 為 0。

在上面的範例中，(在台灣, 早就有很多人在等待他了) 以及 (早就有很多人在等待他了, 丘雅信小姐) 就會被 label 成 1，雖然這樣有些地方因為事實上在劇裡面是已經轉換場景或是開啟新一個話題，導致前後會有不連貫的感覺，如這裡的第 2 個 case，但因為時間的問題，並沒有對這地方做太多的著墨。 關馬西在船上及 伯母 則是一個 negative sample。

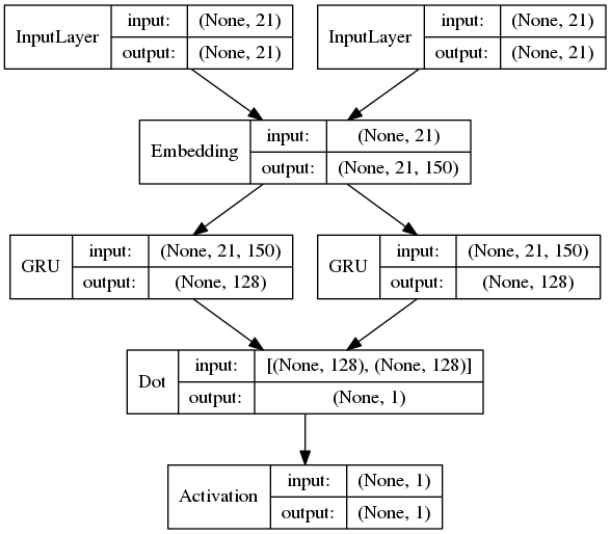
另外，考量到大多時候人理解句子的時候，不單只是但前面一句，而是看上下文的多句話來理解文意，我們也嘗試了兩句對兩句、三句對三句等訓練方式。一併會在待會的比較裡頭詳細說明。

3 模型架構及訓練

以下為我們使用的模型架構，每次的 input data 為一個 pair 經過 tokenize 的句子，在經過 pre-trained 好的 word vector embedding 之後，兩句分別經過一個 RNN，得到經過 encode 完，代表這一句話意思的 vector，再將兩句內積起來過一個 sigmoid，得到最後代表兩句相關性的分數，與前述所提到的 label 算 cross-entropy 得到 training loss。

參數如下：

epochs=10, optimizer=Adam (0.001), batch_size=128, kfold_cross_validation=10



4 測試方式

一樣將題目及選項 tokenize 完後，將(Q, A1), (Q, A2) … (Q, A6)分別餵進 model 裡面，各自得到一個分數，這分數即是 model 認為這兩句的情境相似程度，取最高的一個當作最後答案。惟需特別的是當句子過長超過 model 所能接受的 input 長度時，我們拿 Q 的最後面以及各個 A 的最前面，考量到如此中間不會有其他字詞造成事實上為跳著預測的問題。但實驗若取 A 的後面似乎對準確度也沒有統計上的明顯差異。

5 模型演進史

若將 Kaggle 上面的分數做成一個圖表如下。在過 strong baseline 的前幾個 model 都是用如上的 model 結構，惟 pre-trained embedding 是 non-trainable 的。並且共同 share 一個 RNN 參數，但如此 validation accuracy(判斷兩句是否來自同一個)僅能到 0.68 左右，而 Kaggle 的分數也僅有 0.39960，一直到我們將 trainable 改成 True 並且 RNN 改成不共用才讓 accuracy 提升到 0.74，Kaggle 分數也成功來到 0.46126。

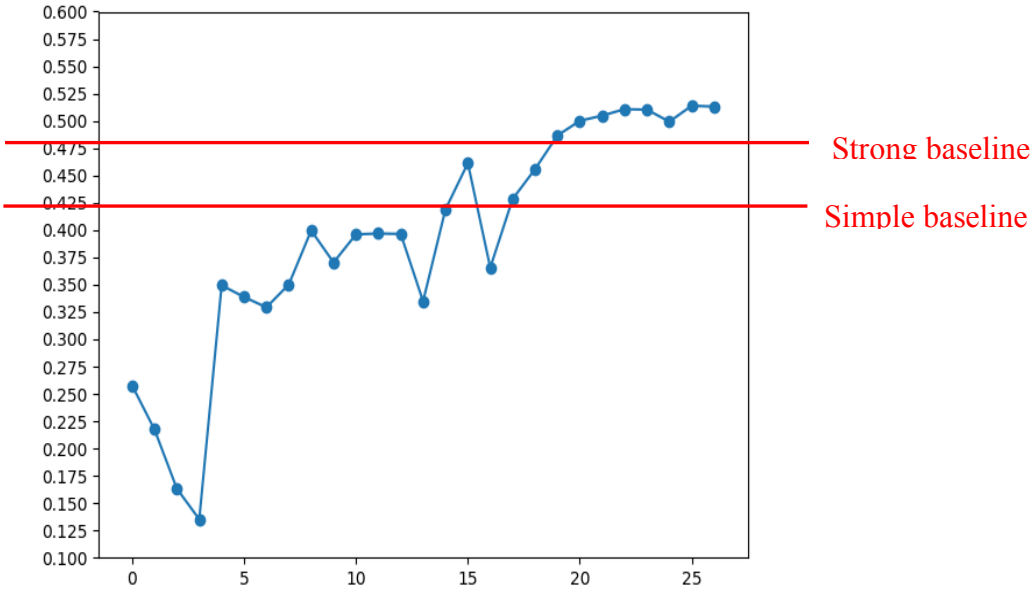
後來也嘗試了三個句子預測三個句子的 model，但發現雖然 validation accuracy 上升不少，甚至可以到 0.80 以上，但 Kaggle 卻掉到剩 0.36561。經過一番檢視後，發現在 testing data 裡題目或選項有 3 句的其實非常少，如此會造成 validation 跟 testing distribution 誤差。(補一下有幾趴是三句話的)

後來，在停滯了好一段時間後，想說試試看 ensemble 好了，因此將 10 個 fold 分別 train 出來的 model 做一番 max-voting，結果出乎意料的好，從 3 個 model 的 0.48656，5 個 model 的 0.50000 到 10 個 model 的 0.50474。接著在考量若使用 average 的方式做 ensemble 會不會比較好，將 10 個 model 各個選項的機率平均之後取 argmax 作為最後答案，如此可以再讓準確率提升到 0.51067。

比較值得一提的點是後來我們想了一個新的 ensemble 模式，分為兩階段。第一步與 average 一樣，惟這時不取第一高的，而是刪去最低的 3 個選項，再把每一個 model 在剩下的 3 個選項的機率 re-normalize 到 1 (a / np.sum(a))，再由最後 10 個 model 平均後的三個 score 裡面選出最高的一個最為答案，如此可以再讓準確率提升至 0.51383。

最後，我們設計了一個新的 wordvec，參數設定是 sg=1, window=7, iter=30，然後採用上面的 ensemble 方式，這樣的組合能夠再把準確率直接推到 0.55417。

下圖為 model 的成長過程（因為最後的 0.55417 是最後一天上傳才得到的分數，因此不在圖中）：


0.600
0.575
0.550
0.525
0.500
0.475
0.450
0.425
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0
5
10
15
20
25
Strong baseline
Simple baseline