題目：TV-conversation

隊名：NTU_b04901067_燃燒肝和 GPU

組員：B04901067 陳博彥、B04901068 劉力仁、R06942143 籃聖皓、B04505008 施伯諺

一、Introduction & Motivation

我們這次所選的 final project 主要是在處理自然語言的預測問題，透過台灣連續劇及鄉土劇的台詞作為訓練資料，訓練 model 來預測日常生活對話之選擇題的答案。之所以會選擇這個問題，主要是因爲我們都對於自然語言的處理非常有興趣，並且認為這是一項很有挑戰性的任務；而且也剛好有組員正在修習「數位語音處理」相關的課程，並想藉由這次的機會應用自己從數位語音處理所學習到的技法，進而精進用 Deep Learning 處理 NLP 的能力。

我們認為這個 final project 在處理上，主要有以下困難：

1. 由於中文不像英文、法文、德文等語言會採用空格去區隔開每一個字詞，故需要經過分詞處理。
2. 本次 project 所使用的分詞套件為 jieba，其資料庫與演算法均只對簡體中文做最優化。
3. Dataset 所提供的 training data 與 testing data 句子長短的分布存在著明顯的不均。

二、 Data Preprocessing/Feature Engineering

(一) jieba 斷詞

為了訓練出好的 sentence vector，好的分詞及 word vector 是不可或缺的。原版的 jieba 是簡體版的，一開始使用分數並不理想，後來上網找到繁體版的字典，引用後進步了很多。

(二) gensim 的 training data

我們 RNN 架構的輸入是由 gensim 所訓練出來的 word vector，因此 gensim 的訓練資料也不得馬虎。

1. 我們首先觀察到 training data **文本中**的**句子都**很短，只要有語氣稍微暫定就會換行，常常兩三行是同一個人說的，因此我們將 training data **文本中的兩句或三句合併成**一句話輸入給 gensim 的 word vector model，藉此讓 gensim 能學到更完成的詞意關係。
2. 文本雖然只要語氣一停頓就會斷句，但仍然有一些標點符號，我們將這些標點符號都轉換成空格，並在每句話的結尾也都補上空格，所以三句文本合併時，合併處都會有一個空格，目的是希望 model 可以看出合併句子的停頓處，且不會受特殊的符號影響。

(三) LSTM model 的 training data

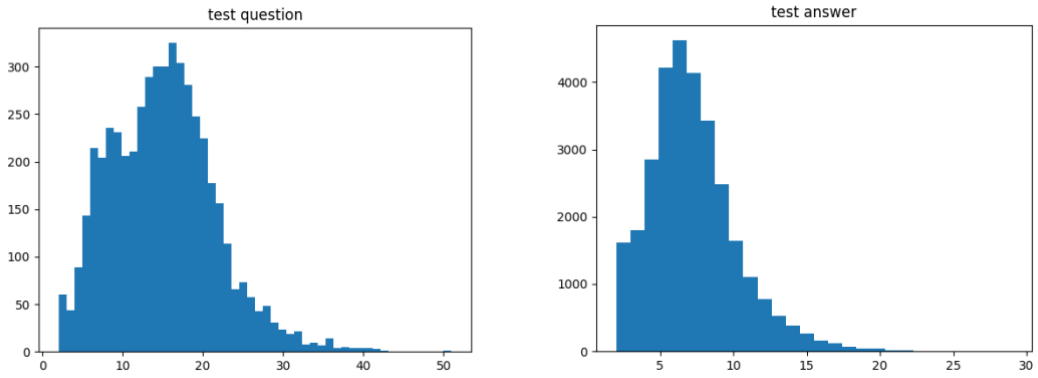
我們的 RNN 模型是兩排的 LSTM 分別算出問句和答句的向量，內積後推斷出它們是否為文意上接續的上下兩句。因此需要從文本切出問句和答句，有的給錯誤有的給正確 label。

1. 起初我們照著訓練詞向量的 gensim model 的方法一樣給 training data。考慮到文本太短的問題，我們將三句文本合併當一個問句，後兩句文本合併當成它的答案，而每個問句也會順便給一個錯誤答案，也就是在文本中隨意挑兩個連續的句子當反例。

然而，training data 中有些句子太短，短到就算把三句拼在一起還是太短，有些則很長，一兩句拼起來就夠了，因此一味地將文本中相連的三句拼成一句未必是一個很好的做法。我們因而又想出了第二種做法。

2. 我們在拼 training data 句子的時候，先數一數這句 training data 被 jieba 斷成幾個詞，再根據這個決定要把幾句拼在一起。我們盡可能讓餵給 LSTM model 的問句跟答句的句子長度和 testing data 的問題/選項平均長度一樣，而不是單純得找三句跟兩句分別拼在一起當問題和答案。實驗後發現準確率有提生，因此我們又進行了第三種切法。

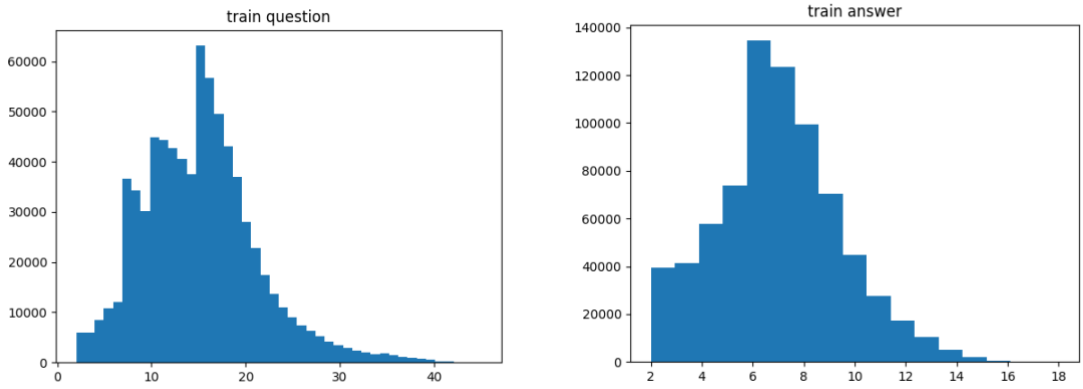
3. 在發現 LSTM model 的輸入句子長度跟 accuracy 有很大關係後，我們就對 testing data 做了分析，找出問句和答句不同長度的分佈狀況，分佈如下圖：



由圖表可以看見，testing data 的問句，長度在 10 詞到 20 個詞這個區段最為常見，選項的長度則是 5 個詞到 10 個詞最為常見。我們希望我們 training data 的句子長度，不但要跟 testing data 的平均長度類似，最好連「分佈」都很接近。

我們在根據文本切 training data 時會根據目前切出來的句子長短來決定它有多少機率要繼續增加長度還是停止，動手調整調每個長度之前提下停止增加長度的機率，想辦法讓 training data 的句子長度分佈接近 testing data。

下圖是我們切出來的一份 training data，問句和長度的分佈圖：



從圖中可看見，我們 training data 的問句/答句的長度分佈，跟 testing 的問題/選項機率分佈十分接近。使用這樣的 training data 讓我們的 model 準確率大幅提升，後面 Experiment and Discussion 的部分會再做更詳細的比較。

另外，由於我們切 training data 的 code 是有隨機性的，也就是說，每次執行所切出來的 training data 都不盡相同，但是「分佈」卻是一樣的（且都與 testing data 相同）。我們在訓練 model 的時候，故意為每個 model 都切一次 training data，如此一來，每個 model 所用的 training data 都不大一樣，ensemble 效果更好。

三、Model Description

(一) gensim word2vec + sentence average cosine similarity

先把 training data 經過 Data Preprocessing 提到的方法做處理，放到 gensim 中 train 一個 word2vec model，testing 時，把 testing data 的句子中每個字的 word vector 做平均當作 sentence vector，再把問題跟每個選項的 sentence vector 算 cosine similarity，並選擇 similarity 最高的當作答案。

gensim word2vec 的訓練參數為：

sg = 1, size = 150, window = 5, min_count = 2, iter = 200

sg = 1 代表使用 skip-gram 來訓練，根據網路上的說法，在少量 training data 中，skip-gram 會比 CBOW 有更好的效果，而我們的實驗結果也的確實如此。

size = 150 是 word2vec 的維度，我們比較過不同維度的表現，如下：

word2vec 維度	250	200	150
private/public score	0.37944/0.35810	0.38063/0.36086	0.39011/0.36679

由上表可看見，維度越低效果越好。理論上，維度越高，word vector 能夠帶有越多資訊，但是，應該是因為我們的 training data 太少，低維度的 vector 能夠攜帶重要資訊，且不會帶有過多不必要的資訊，表現反而比較好。

之後，我們又在這個架構下，再加上一些進階一點的做法，以下是比較：

- 方法一：

  最基本的 sentence average cosine similarity，方法就是上面描述的那樣。

- 方法二：

  在去平均算 sentence vector 之前，先將每個字的 word vector 乘以這個字「在文本中出現頻率的倒數」。這個方法的想法是：如果問題跟選項同時出現一個很少見的字，這就表示這個選項很有可能是正確答案，相反的，如果問題跟選項同時出現的字是「的、好、我」這種很常見的字，代表性就沒那麼高了。

  在其他的試驗中，我們發現「濾掉停用詞」的效果不如「保留停用詞」，那是因為濾掉停用詞後，句子太短，含有的資訊太少，而這個做法（**乘**上頻率倒數）則間接達到濾掉停用詞的效果，且不會損失太多資訊。

- 方法三：

  將越後面的字的 word vector 乘以較高的權重，在去平均算 sentence vector 之前，將句子中越後面的字乘上越大的權重（第一個字乘的權重是 1，然後線性遞增，最後一個字的權重是 2）。

這個方法的想法是：越後面出現的字，跟下一句越有關聯，尤其是那些很長的句子，最前面幾個字跟答句的關係應該很低了。

➢ 方法四：

將 word vector 乘以出現頻率倒數再做平均 + 將越後面的字的 word vector 乘上比較高的權重，**事實上就是**同時採用方法三跟四的機制。

方法	一	二	三	四
private/public score	0.45573/0.43478	0.46086/0.45454	0.44901/0.43952	0.46166/0.45533

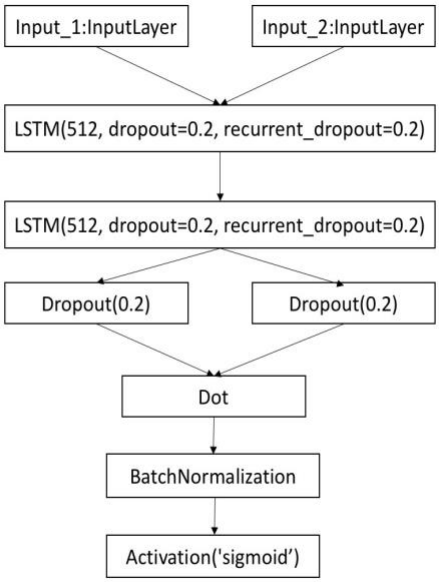
gensim word2vec 的訓練參數為：

sg = 1, size = 64, window = 7, min_count = 2, iter = 200

由上表可知，「乘以頻率倒數」跟「後面的字乘以比較大的權重」兩種做法都能有效提高準確率，兩者合併效果更好。

(二) LSTM（share weight）+ cosine similarity

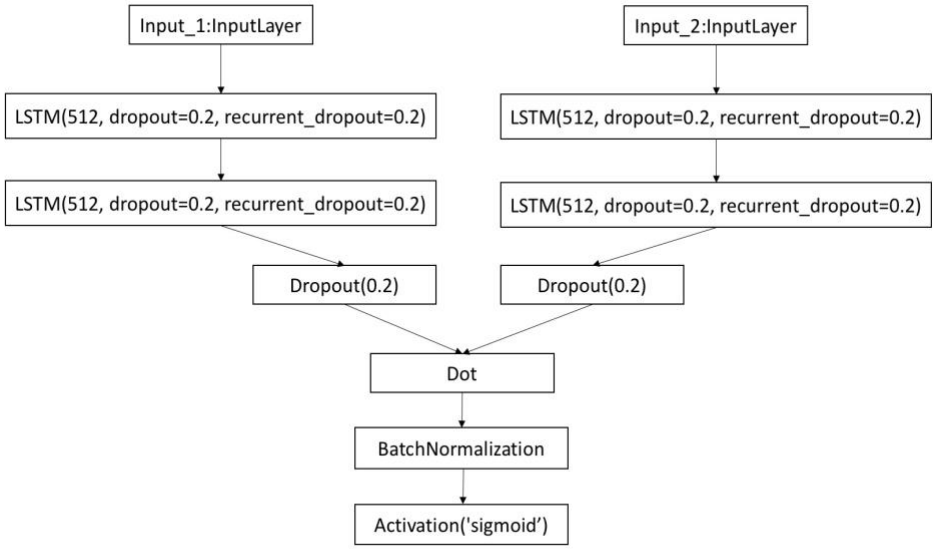
Model(一)並不能考慮字詞在句子中出現的順序，準確率到 0.47 後就上不去了。LSTM 可以考慮字詞出現的順序，因此理論上會有較好的表現。而實際上，我們用 LSTM 取得相當不錯的準確率，這也是我們最終採用的模型架構。模型的架構圖如下：



我們先用 gensim train 好一個 word2vec 的 model，然後將問題/選項的句子分別 embedding，再分別餵到 LSTM 中，LSTM 可以將兩個句子分別 encode 成向量，我們再將這兩個向量做內積算 cosine similarity，並把此 similarity 當作 output。其中，兩個 LSTM 是 share weight 的。

對於正確的 training pair，我們給的 label 是 1，錯誤的 pair 的 label 是 0。其實當初在決定我們的 label 要給 { 1, 0 }還是{ 1, -1 }**時**，我們經過了一番思考及實驗，在 Experiment and Discussion 部分有詳細的說明。

(三) LSTM（non share weight）+ cosine similarity



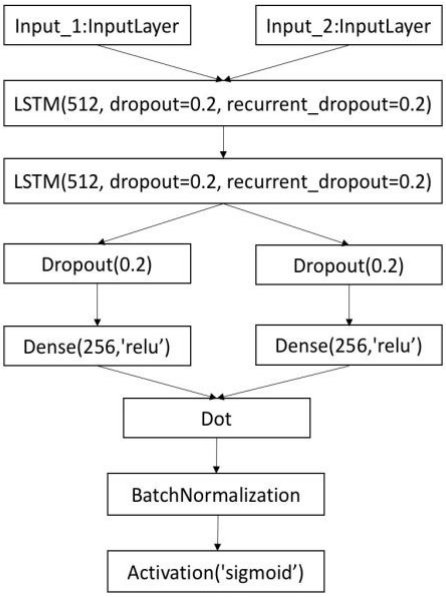
Model(二)其實是一個完全對稱的模型，因為問題跟答案共用同樣的 LSTM，而 dot 也有交換律，也就是說，「我來到教室——看到老師在台上」跟「看到老師在台上——我來到教室」這兩個問句/答句組合對 model 來說是完全一樣的，但很明顯只有第一句是正確的。為了解決這樣的問題，我們採用了 non share weight 的做法，也就是處理問題跟答案的 LSTM 有不同的 weight。我們希望這樣的 model 可以學到句子的因果關係，獲得更佳的準確率。

(四) LSTM（share weight）+ Dense + cosine similarity

Model(三)有下列幾個缺點：

1. model 太大，訓練很慢。
2. 兩個 LSTM 都只分別看到問題或者答案，相較於 share weight 的做法，只利用了一半的 trianing data。
3. 直觀來講，LSTM 是將一個句子 encode 成代表句意的向量，而這樣的機制不應該隨著這個句子出現在問題還是答案而有所不同，所以 non share weight LSTM 未必是一個合適的做法。

在討論以上三個缺點後，我們想出了這個 Model 架構。



這個 Model 透過 LSTM 將問題 encode 成 sentence vector，再將 sentence vector 分別經過一層 Dense 的轉換，最後再餵到 Dot 中算 cosine similarity，這樣的架構，model 就有機會學到什麼樣的句子應該是「因」，什麼樣的句子應該是「果」，同時也能解決 Model(三)的三個缺點。

Model(一)只是 final project 前期我們所使用的基礎作法，後來就被我們淘汰了，而 model(二)~(四)在 Experiment and Discussion 部分有詳細的比較。

四、Experiment and Discussion

(一) Activation 和 Label

Cosine similarity 的範圍理論上要在-1 ~ +1 之間，然而 cosine similarity 為-1 就代表問題跟答案是兩個相反的向量，我們用以下情境來討這樣會遇到的問題：

假設現在我們有三個句子「今天天氣真好」「我晚餐吃牛排」「你明天幾點有空」，這三個句子任兩個組在一起都會是「錯誤」的組合，但是無論我們用什麼方式來 encode sentence vector，都不可能讓三個向量兩兩的 cosine similarity 為-1。

而實際上我們經觀察發現，所有 training pair 的 cosine similarity 幾乎都在 0~1 之間分佈，很少會到負數，因此 label 給{ 0, 1 }似乎是比較合理的方式。然而，假設有少數的「錯誤 training pair」他們的 cosine similarity 已經是負的了，則如果 label 是 0，model 還會自動修正，讓他們的 cosine similarity 上升，這卻不是我們所樂見的。為了解決以上的問題，我們想到了這個方法：

```
sim=BatchNormalization()(sim)
sim=Activation('sigmoid')(sim)
model = Model(inputs=[question, answer], outputs=sim)
model.compile(loss = 'binary_crossentropy', optimizer='adam'
```

我們將 cosine similarity 後的數值做 BatchNormalization，再透過 sigmoid 餵到 output。即便我們不知道 cosine similarity 應該要是多少，我們能確定的是，正確 pair 的 cosine similarity 會比較高，而 BatchNormalization 會將所有的 similarity 平移到 0 的左右，再透過 sigmoid，我們就可以把它們分開了（注：label 是給{ 0, 1 } ）。所有錯誤的 pair 的 similarity 會被往小的方向推，但是由於 sigmoid 的

特性，如果該錯誤 pair 的 similarity 已經很小了，則不會一直強力把它再往負更多的地方推，因此可以解決以上討論所面臨的困境。

我們在 training 的過程中發現，這樣的架構在 train 的時候會學得特別快，收斂到最後的 validation 也會比其他方式多 1 ~ 2%，是很不錯的架構，我們所有上傳 kaggle 的 LSTM 做法都是按照此架構去實施。

注：為了要讓 BatchNormalization 之後的 similarity 以 0 為「錯誤—正確」的分界，training data 中「錯誤—正確」的比例要是 1：1。

(二) 比較不同 Model

回顧一下再上一個章節，我們提出了四種 Model，在此我們將它們整理成以下表格：

model	(一)	(二)	(三)	(四)
做法描述	Sentence average cosine similarity（或再乘以詞頻倒數、線性權重）	LSTM（ non share weight ）+ cosine similarity	LSTM（share weight ）+ cosine similarity	LSTM（non share weight ）+ Dense+cosine similarity
構想	最基本的做法	考慮字詞出現順序，用 LSTM 將句子按照句意 encode 成 sentence vector，但 model 完全對稱	考慮句子的因果關係，使用非對稱的 model	考慮句子的因果關係，使用非對稱的 model，同時改善(三)的缺點
Validation Accuracy	X	76~79	81~82	81~82
Kaggle Accuracy（約略）	36~47	51~53	48~49	49~50
優點	很簡單，predict 很快，可以 ensemble 數十個 model 都不成問題。	有很好的 Kaggle accuracy，也是我們最終採用的架構。	學到比(二)更多資訊，因此 validation accuracy 很高。	1. 學到比(二)更多資訊，因此 validation accuracy 很高。 2. (三)的加強版，學得比(三)更快更好。
缺點	1.太過粗糙，accuracy 很低。 2.沒辦法做 validation，一定要上傳 Kaggle 才知道結果。	無法考慮句子的因果關係，「我來到教室——看到老師在台上」跟「看到老師在台上——我來到教室」對它來說是一樣的。	學到「太多」training data 的 domain knowledge，在 training data 上 overfit，因此 Kaggle accuracy 不高。	

Model(一)是個很粗糙的做法，在 final project 的後期就漸漸被我們淘汰。

Model(三)、(四)相對於(二)能夠在 training data 中學到更多的資訊，因此在 validation 中有很高的 accuracy，然而這次的 project，training、testing 的差異過大，Model(三)、(四)會攜帶過多的 domain knowledge 去做 testing，成效反而輸給 Model(二)。

為了驗證以上提出的解釋，我們在 Kaggle deadline 之後，做了這個實驗：

1. 步驟：

(1) 將五個表現不錯的 model 一起**對** testing data 做 prediction，把「五個 model 都認為是正確」和「五個 model 都認為是錯誤」的選項當作 ground truth，把這些篩選出來的選項和題目組成 pair 並分別給 label 1/0，當作 fine tune data。

(2) 將已經 train 好的 Model(二)、(三)、(四)針對這些 fine tune data 再跑 5 個 training epoch，並上傳 kaggle。

(3) 比較 Model(二)、(三)、(四)的 validation accuracy 和 fine tune 前後的 accuracy。

2. 結果：

model	(二)	(三)	(四)
validation accuracy	76 ~ 79	81 ~ 82	81 ~ 82
kaggle accuracy(fine tune 前)	51 ~ 53	48 ~ 49	49 ~ 50
kaggle accuracy(fine tune 後)	54 ~ 55	55 ~ 56	55 ~ 56

3. 分析：

Model(三)、(四)比 Model(二)有更高的 validation accuracy，但是由於攜帶過多的 training data domain knowledge 去做 testing，成果反而不如 Model(二)，而我們利用篩選出來的 testing data 對這些 model 分別做 fine tune，可以看見 Model(三)、(四)在 fine tune 後的 accuracy 就超越了(二)。這樣的結果驗證了兩件事：

(1) 無論是在 training data 上學習或者在 testing data 上 fine tune，Model(三)、(四)都學得比 Model(二)好，model(三)、(四)確實學到比較多資訊。

(2) Model(三)、(四)之所以在 testing 上表現不如 Model(二)是因為攜帶過多 training data 中的 domain knowledge 到 testing，亦即在 training data 中 overfit。

總結以上，Model(二)在 Kaggle 中有最佳的 accuracy，是比較好的 model。

(三) 比較不同 training data

1. 我們首先比較有沒有去標點符號和在句子合併處補空格的差異，實驗 model 之參數如下：

rnn_dim = 512, batch_size = 512, dropout = 0.2

並以 Data Preprocessing (三)中 1. 題到的切割方式，問句固定為三句、答句固定兩句的方式來比較（皆為 private/public 分數）：

沒有去標點符號且無補空格之 training data 成績	0.47430/0.47628

以空格代替標點符號且在句子間補空格之 training data 成績	0.47667/0.47233

可以發現成績並沒有多大改變，但身為 engineer 的其中一位組員非常肯定自身的直覺(engineering sense)，認為**後者才是比較合理的做法**，因此就沿用至其他之後的模型中。

2. 比較 Data Preprocessing (三)第 1.和第 3.所切的不同 training data 對分數造成的影響：

實驗 model 之參數如下：

rnn_dim = 512, batch_size = 512, dropout = 0.2

(1) 用第 1.切法，也就是問句固定三句、答句固定兩句的切法：

0.47430/0.47628

(2) 用第 3.切法，也就是 training data 的問句和答句長度的機率分佈和 testing data 的相近：

0.53359/0.53241

這是一項驚人的發現，這兩個 model 是在參數完全沒改變，只改變 Data preprocessing 的情況下多了 6%的準確率，我們有以下幾點推測：

- 原始問句固定三句答句固定兩句的做法，我們只能讓 training data 的句子長度平均貼近 testing data，卻無法讓整體分佈接近，手法太過粗糙。
- 在訓練 model 時，最好讓它看過長句子也看過短句子，比較能夠應對 testing data 中或長或短的句子。

(四) Ensemble

這部分我們將比較三種 ensemble 方法，分別是 voting、averaging 跟長短句 ensemble。

1. Voting：

我們將五個不同的 model（各自準確率都在 51~53%之間）分別拿去 predict，並選擇最多 model 選的選項的答案當作最終的選擇。

2. Averaging：

我們將跟 averaging 同樣的五個 model 分別拿去 predict，並將每個選項所有 model predict 的分數做平均，再選出分數最高的選項。

3. 長短句 ensemble：

在意識到 training data 的長度會對結果好壞影響非常大之後，我們想說這樣如果把長句和短句的模型分開訓練，是否會比較好？於是我們以 testing data 問句長度的中位數 14 當成長句和短句的分界，讓短句模型的 training data 句子長度平均約在 8 而長句模型的平均在 20。Testing 時，我們使用了三個 model，長句、短句、跟一般（已知 accuracy 為 0.53），當遇到句子長度小於等於 14 的問題時，使用短句跟一般兩個 model 的預測結果做加權平均，遇到長度大於 14 的問題時，則使用長句跟一般的 model 之預測做加權平均。

Ensemble 方法	Voting	Averaging	長短句
private/public score	0.56403/0.55968	0.55652/0.56086	0.51936/0.50750

我們在網路上找到一種說法：

Voting 較適用於 classification 的問題，而 averaging 較適用於 regression 的問題。

由於 classificaition 的預測，未必每筆 data 的信心度都會很高，這時如果採用 averaging，容易受到 outlier model 的影響。舉例來說，在 hw5 的 sentence sentiment classification 問題中，如果我們遇到的題目是「Well...I am okay.」，這應該是屬於正面的句子，但語氣卻非特別強烈，大多數好的 model 對於這個句子的評分可能會在 0.6 附近，但在 ensemble 時，如果有一個很爛的 model 把這個選項評為 0.1 分，就算大說數的 model 都認為它是正面的，ensemble 過後依然很可能選到錯答案。而至於 regression 無法做 voting，自然 averaging 較為適用。

這次的 final project 的性質介於 classification 跟 regression 之間，雖然是要選一個選項作為答案，但事實上也是要選一個「相對於其他選項最適合」的答案，因此同時具有 classification 跟 regression 的特性。根據我們的實驗，voting 的效果稍微好一些，我們認為的原因如下：

當最適合與第二適合的選項相差不遠時，voting 較不會受到 outlier model 的影響。
舉例來說，當問題是「今天天氣真好」而選項有「對啊，天空很晴朗」跟「嗯，我要出去玩」，直觀看來第一個選項是更貼切地回答，但第二個也未必不可，因此一個好的 model 在 predict 的時候，第一個選項的分數會最高，但第二個選項的分數也不低。這時，如果採用 averaging 的 ensemble 機制，只要有一個 outlier model 把第二的選項的分數預測的遠比第一個選項高，ensemble 後就很可能因此選到錯誤的答案。

長短模型 ensemble 的觀測：

結果可以從上方圖表看出，並不太理想，尤其我們是用一個準確率為 0.53 的模型來輔助，但結果不升反降，我們推測的可能有幾個原因：

- Weight 也許有調整空間，我們在預測時是將長短句 model 跟一般 model 做 2:1 的分數加權，但這只是很直覺嘗試，畢竟我們的預測分數是有經過 sigmoid 的，也許要有一些數學上更合理的加權。
- 如果只拿短句訓練短模型，也許會發生像當初我們訓練 gensim model(詞向量)一樣的問題，也就是句子都太短，無法訓練出好的語意關係。
- 長句模型 training data 的句子長度平均大約是二十，然而 training data 在切得很長的情況下，也許會把很多不相干的句子拉在一起訓練，而無法有好結果。
- 即便 testing data 的問題長度分佈從 1 到 51 都有，但絕大多數的 data 長度是在 10~20 間分佈，因此硬是拿很長的句子跟很短的句子去分別訓練長/短句模型未必是個好的做法。

(五) Other Tips

以下是我們針對許多小細節做調整並做實驗所得到的結果，實驗時，一次只改變一個操縱變因，顯示方式為 kaggle 分數的 private/public。

1. word2vec dimension 小 > 大
   在 model(二)的架構下實作

   dimension = 64 ➔ 0.47628/0.47430
   dimension = 250 ➔ 0.46877/0.46482
2. gensim sg=1 (skip-gram) > sg=0 (CBOW)
   從一開始用最簡單的 sentence average cosine similarity 中就發現 sg=1 高了很多，之後就沒改過。
3. OOV zero padding > random padding
   當遇到 OOV 時所補的 word vector，可使用零或 random。(我們 random 一個向量，然後將 2 norm normalize 至 1)

   random ➔ 0.52766/0.51857
   zero ➔ 0.53359/0.53241

4. pad front > pad back

   當句子長度小於 LSTM 的 max_squence 時，可以選擇在句子的前面或者後面做 padding。在做實驗的時候，我們有先用小量訓練資料跑過模型，發現 pad front 的 validation_accuracy 比 pad back 好大約 1%，但針對這個問題，我們並沒有用大量資料訓練過然後上傳 kaggle 測試。
5. Dot > concatenate + DNN

   同 pad front 與 pad back 的實驗，我們也是以小量資料測試模型學習結果，發現 Dot 模型的 validation accuracy 比較好，因此在之後的模型我們都選用 Dot。
6. LSTM share weight > LSTM none weight

   share weight ➔ 0.53359/0.53241

   none share weight ➔ 0.49209/0.48339
7. Dropout 小 > Dropout 大 Dropout

   我們是同時調 recurrent dropout 和 dropout，在 Dropout 大於 0.2 之後，training accuracy 就會上不太去，因此設定在 0.2。

五、Conclusion

經過一整個月的奮鬥，我們認為，這次的 final project 最大的要點不是在於怎麼 train 我們的 model，而是在於如何減少 training 和 testing 環境的差異。

在訓練 model 的部分，很多參數都會影響 model 的好壞，像是標點符號的處理、斷詞方式、word2vec 維度、gensim 訓練參數、LSTM 維度、單向 LSTM/雙向 LSTM、Dropout Rate、batch size、learning rate、OOV padding(pad 0 或者 pad random)、sentence padding 方式(pad 前面或者 pad 後面)，這些參數都或多或少會影響 testing accuracy，每個參數影響 accuracy 的幅度都不大(大多在 1%以內)，卻需要花大量的時間去嘗試。相反地，如果成功降低 training 跟 testing 環境的差異，準確率將會有飛躍性的成長。

至於怎麼減少 training 和 testing 環境的差異，我們得到了兩大結論：

(一)Data Preprocessing

以 gensim 來說，我們發現將文本一次取二到三句進去訓練最合適，因為是訓練詞向量，不需考慮 testing data 的長度，因此有越完整的字句越能有好效果。相較於 LSTM，因為關係到輸出就是預測的結果，如果 training data 能夠在詞向量越完整、長度和 testing 類似，較能實現出更好的結果，所以我們採用和 testing data 相同的句子長度機率分佈來切文本，這個方法讓我們的 accuracy 整整提升了 6%，是我們最終得到 ranking 第一名的最大關鍵。

在處理資料的過程中出現很多分歧，有很多種方法值得我們去試（斷詞、標點、空格等等），然而電腦一天只能訓練兩三個模型，更何況沒有給可以做 validation 的資料，得上傳 kaggle 才能得知模型的好壞，此時直覺有時比照順序的每個都試還要好。

(二)避免在 training data 上 overfit

如同在 Experiment and Discussion 的(二)所提到的，太複雜(學習能力太強)的 model 在 validation 會有很好的表現，但卻會攜帶過多 domain knowledge 到 testing 環境，因此不能太過看重 validation accuracy，而要看重 model 在 testing data 上的表現。

六、**Reference**

1. https://medium.com/mlreview/implementing-malstm-on-kaggles-quora-question-pairs-competition-8b31b0b16a07 LSTM sentence similarity example

2. https://github.com/ldkrsi/jieba-zh_TW jieba 繁體中文版

3. http://www.cs.cmu.edu/~rayid/talks/Ensemble%20Classification%20Methods.ppt Ensemble methods