機器學習 Final Project Report

題目: Delta Chinese QA – 邁向中文問答之路
隊伍名稱: NTU_b04901015_劉碩帶我闖天關
隊伍成員: B04507009 何吉瑞
B04901015 傅子興
B04901022 黃家翰
B04901040 林哲賢

工作分配:

- R-NET model adjust & realization: 林哲賢
- sliding window approach realization: 何吉瑞
- sentence select & adjust approach realization: 傅子興
- FastQA model adjust & realization (final selected model): 黃家翰
- Report writing and discuss: All

Preprocessing & Feature Engineering

- ***R-NET model adjust***
  將 json 資料先依資料意義(分為 title, content, question, ans 等)存入 dictionary，然後我在網路上找到 pre-trained 的 word embedding，但是發現 OOV 實在是太多，所以最後就將 dictionary 的資料先用 jieba 斷句後將原本的東西用 gensim 再 train 一次 word embedding，最後將資料換成 word vectors，至於 OOV 的部份我都用 zero vector 代替。

- ***Sliding window approach***
  將 json 資料先依資料意義(分為 title, content, question, ans 等)存入 dictionary

- ***Sentence select & adjust approach***
  將 test 的 json 資料先依資料意義(分為 title, content, question, ans 等)存入 dictionary，然後對每個文章做 split(“。”) 以將每個句子分開。最後再以 jeiba 將每個句子中的詞分開。

- ***FastQA model adjust***
- 讀取 json 檔資料，先找出文章中與題目對應字詞最多的那個句子當作該題的 context。接著將 context、question、answer 以句號分割並用 jieba 斷詞後儲存。另外，對於 answer，則將其斷詞後找出第一個詞和最後一個詞在該題 context 中的位置並儲存。此外，由於每個問題只會對應到一個句子，因此每個句子在文章中的位置也需儲存以方便把模型預測出的結果轉換回答案在整個文章中的位置。

Model Discription

- **R-NET model adjust**:
  這個 model 是參考 SQuAD 裡面的其中一個 model，如下圖

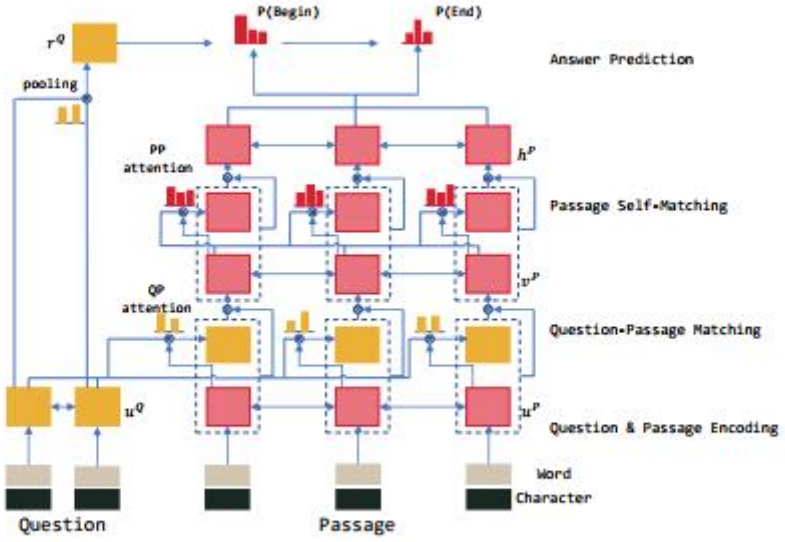


Figure 1: R-NET structure overview.

這個 model 的實做法分為了以下幾個部分:

(1) Question and passage encoding:
這個部分主要就是要把 passage 和 question 做 encoding，這篇 Paper 方法使用的方法是先將 passage 和 question 做 word level (用 jieba 斷句) 和 character level embedding 後再用 Bidirectional GRU 結合這兩個再去生成新的 representation，如以下所示:

$$u_t^Q = \text{BiRNN}_Q(u_{t-1}^Q, [e_t^Q, c_t^Q])$$
$$u_t^P = \text{BiRNN}_P(u_{t-1}^P, [e_t^P, c_t^P])$$

但我刻完了 model 之後在 train 的時候發現，雖然 character level embedding 可以處理 OOV 的問題，可是 train 的實在是太慢了……所以我最後捨棄 character level 只做 word level embedding。

(2) Gated Attention-based Recurrent Networks
這篇 Paper 提出了一個 Gated Attention-based Recurrent Networks，和老師在影片裡講的 Attention-based Recurrent Networks 不一樣的地方是，他在 RNN 的 input 加了一個 gate，跟 passage word 和 question 的 attention-pooling vector 有關，目的是要根據不同的問題聚焦在 passage 上不同的地方，下面的實驗會比較有無這個 gate 的差別。

(3) Self-matching attention

經過了上面(2)的處理後可以得到 question-aware passage representation，但是這個只代表 passage 裡比較重要的一小部分，在得出答案的時候還是要有整個 passage 才行，因此這篇 paper 提出了這個 Self-matching attention 的方法，將 question-aware passage representation 和自己 match 在一起:

$$h_t^P = \text{BiRNN}(h_{t-1}^P, [v_t^P, c_t])$$

where $c_t = att(v^P, v_t^P)$ is an attention-pooling vector of the whole passage (v^P):

(4) Output layer

Output layer 使用了 pointer network 來預測開始位置和結束位置，而 train model 的方法是 minimize 從 distribution 裡 predict 出來的 start and end position 機率的負 log 和。

- **Sliding window approach**:

本作法沒有使用機器學習的模型。以下簡述實作方法:

將切割好的詞每次取固定的字數(數量由 window size 決定，實際最佳值為 35)與題目進行逐字比較，選出與題目關聯性最高的區段再將轉換為 word position 將其輸出。實作分為幾個部分:

(1) 擷取文章片段:

依照 window size 將文章中從第一個字一路 scan 到最後一個字，每次擷取一段片段就執行以下的(2)(3)步驟。

(2) 使用 tokenizer:

對該文章問題作 tokenizer fit，接著以 tokenizer texts_to_matrix 將擷取片段依照同樣的 tokenizer 的 word pool 作出 bag of words 的向量。直接將向量的結果取 numpy where (條件: 值不等於零)的 size 就可以知道擷取片段出現的詞與題目出現的詞的重複數。

(3) 比較:

若是重複的詞數筆紀錄的最多詞數還大，則記錄這個擷取位置的 start point，取代原本的值。

(4) 輸出結果:

將文章 scan 過一遍後，輸出的片段為記錄的 start point 及其往後算 window length 個字。

- **Sentence select & adjust approach**:

本作法沒有使用機器學習的模型。以下簡述實作方法:

取切割好的每個句子與題目進行逐詞比較，並取出相關度最高(重複最多詞)

的句子。將該句子中與題目重複的詞去掉並轉換為 word position 後輸出。
實作上分為幾個部分:

(1) 對每個題目先做 tokenizer:
對於同一個題目，要先對這個詞已經切割開的 list 先做 tokenizer fit，接著對於這篇文章的每個句子執行下面的步驟。

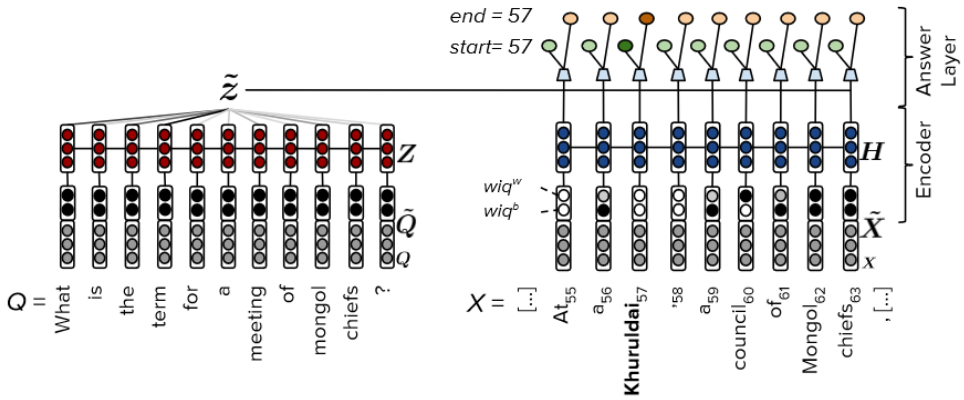
(2) 計算句子與題目相似度:
使用 tokenizer texts_to_matrix 函式將文章中每個句子(詞已經切開)都換成 bag of words 的向量，sum vector 中的每一個值，得到與題目相同的詞的數目和。

(3) 比較與選擇:
將得到的相似度(重複詞數)與記錄的最大值作比較，若是比原本的相似度高，則記錄的 word position sequence 就更新為以新的 start position 作為出發，並去除掉該句子所有與題目重複的詞，最後所留下的字的 word positions。

(4) 輸出結果:
將每個問題都做完之後，就可以直接以 csv write 的方式將記錄的 word position sequence 輸出。

- **FastQA model adjust:**

此次 FastQA 模型參考自 https://arxiv.org/pdf/1703.04816.pdf，該模型架構如下(圖片擷取自連結中論文)：



Word Embedding :

使用 gensim 的 word2vec 建立將 training data 的所有詞彙轉換為 300 維的 vector 的字典，並未使用額外資料。以其 weight 做為 embedding layer 的初始參數並設定為 trainable，使 word vector 能夠參與訓練過程以達到 fine tune 的目的。另外，文本在進入 embedding layer 前會先 pad 成長度為 100 個詞，而問題則會 pad 成長度為 10 個詞。

Encoder :

分別將文本與問題送入一 bidirectional LSTM 中並 return sequences。再將得到的結果透過一層 Dense-layer 投影到一維度相同(300 維)的空間。由於 return squences 的關係，本模型中使用的 Dense layer 皆為 Time Distributed。

Attention layer :

將通過 encoder 後的問題句再通過一層 Dense(1)的 layer 並取 softmax 以得到每個詞的 weight。再將所有 weight 與詞 encode 後的 vector 相乘，取總和後得到 attention vector。

Answer layer (pointer layer) :

將 encode 後的文本、attention vector、兩者相乘(element wise)，以上三個 vector 相接後通過兩層 Dense layer，output unit 為 1。由於文本長度為 100，因此總 output 為 100 為的 vector，通過 softmax 後即代表開頭位置位於該 index 的機率。此 output 以下稱為「start」。

接者找出 start 中最大值的 index，找出該 index 對應到「encode 後文本」中的 vector(以下稱其為「start_feature」)。在將「encode 後文本」、attention vector、start feature、「前兩者相乘」、「一三者相乘」，以上四個 vector 相接後通過兩層 Dense layer，output 同樣會是一個 100 為的 vector，通過 softmax 後即代表結束位置位於該 index 的機率。以下稱其 output 為「end」

Model output :

此模型輸出為[start, end]，代表答案開始與結束的位置。是詞的位置而非字的位置。

其他:

Loss function : categorical_crossentropy

Optimizer : Adamax

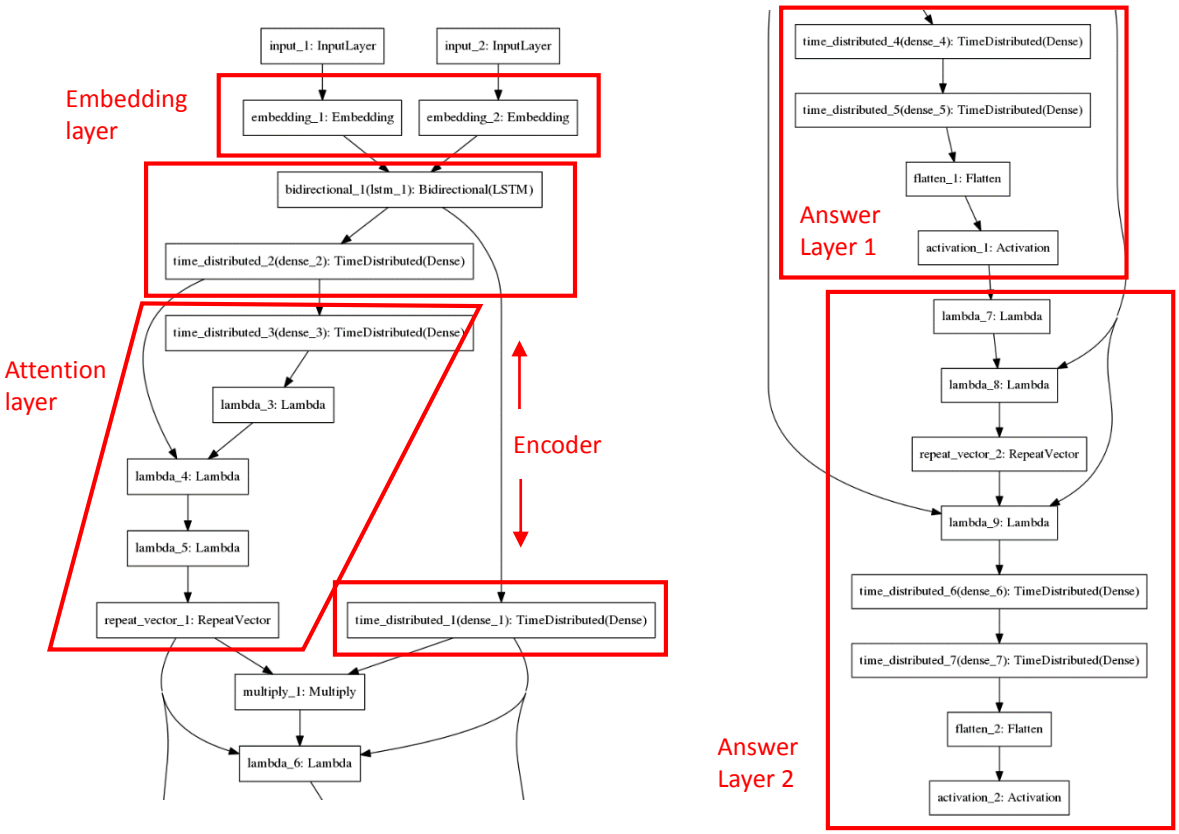
Total parameters : 14890503

[Model Structrue On Next Page]

Model Structure :

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 100)	0	
input_2 (InputLayer)	(None, 10)	0	
embedding_1 (Embedding)	(None, 100, 300)	6183300	input_1[0][0]
embedding_2 (Embedding)	(None, 10, 300)	6183300	input_2[0][0]
bidirectional_1 (Bidirectional)	multiple	1442400	embedding_1[0][0] embedding_2[0][0]
time_distributed_2 (TimeDistribu	(None, 10, 300)	180000	bidirectional_1[1][0]
time_distributed_3 (TimeDistribu	(None, 10, 1)	301	time_distributed_2[0][0]
lambda_3 (Lambda)	(None, 10, 1)	0	time_distributed_3[0][0]
lambda_4 (Lambda)	(None, 10, 300)	0	time_distributed_2[0][0] lambda_3[0][0]
lambda_5 (Lambda)	(None, 300)	0	lambda_4[0][0]
time_distributed_1 (TimeDistribu	(None, 100, 300)	180000	bidirectional_1[0][0]
repeat_vector_1 (RepeatVector)	(None, 100, 300)	0	lambda_5[0][0]
multiply_1 (Multiply)	(None, 100, 300)	0	time_distributed_1[0][0] repeat_vector_1[0][0]
lambda_6 (Lambda)	(None, 100, 900)	0	time_distributed_1[0][0] repeat_vector_1[0][0] multiply_1[0][0]
time_distributed_4 (TimeDistribu	(None, 100, 300)	270300	lambda_6[0][0]
time_distributed_5 (TimeDistribu	(None, 100, 1)	301	time_distributed_4[0][0]
flatten_1 (Flatten)	(None, 100)	0	time_distributed_5[0][0]
activation_1 (Activation)	(None, 100)	0	flatten_1[0][0]
lambda_7 (Lambda)	(None,)	0	activation_1[0][0]
lambda_8 (Lambda)	(None, 300)	0	time_distributed_1[0][0] lambda_7[0][0]
repeat_vector_2 (RepeatVector)	(None, 100, 300)	0	lambda_8[0][0]
lambda_9 (Lambda)	(None, 100, 1500)	0	time_distributed_1[0][0] repeat_vector_1[0][0] repeat_vector_2[0][0]
time_distributed_6 (TimeDistribu	(None, 100, 300)	450300	lambda_9[0][0]
time_distributed_7 (TimeDistribu	(None, 100, 1)	301	time_distributed_6[0][0]
flatten_2 (Flatten)	(None, 100)	0	time_distributed_7[0][0]
activation_2 (Activation)	(None, 100)	0	flatten_2[0][0]

Total params: 14,890,503
Trainable params: 14,890,503
Non-trainable params: 0



Experiments and discussion

- **R-NET model:**

實做出來之後發現效果並沒有很好，出來的 F1score 只有 0.13681(Kaggle Public)，不知道是哪裡寫的不好還是沒有調好參數，因此我們最後選用了下面的 FastQA model，但我們還是用 R-NET model 做了以下幾個實驗(因為不想占用 kaggle 的次數所以以下都用 validation set 的成績):

(1) 將 Gated Attention-based Recurrent Networks 的 gate 給拿掉

我將 gate 拿掉後拿去 train 和我沒拿掉後的結果如下表格

	拿掉 gate	沒拿掉
F1 score (validation)	0.081	0.156

可以發現拿掉 gate 之後 performance 掉了將近一半，這個結果還蠻令我驚訝的，根據不同的 question 聚焦在 passage 上不同的地方原來這麼重要，大幅影響了整個 model。

(2) 比較 GRU 和 LSTM

實驗結果如下:

	GRU	LSTM
F1 score (validation)	0.156	0.161

實驗出來的結果會發現 LSTM 和 GRU 在這個 model 上的效果差不多，但是 GRU 的計算量比 LSTM 便宜，因此在做完這個實驗後我都是用 GRU 來實做。

(3) 更改 bi-GRU 的層數

我在看這篇 Paper 的時候他並沒有說 GRU 要疊幾層，因此我就做了個實驗親自嘗試:

層數	F1 score (validation)
1	0.128
2	0.140
3	0.156
4	0.148

四層以上要 train 太久了我就沒有再繼續試下去了。

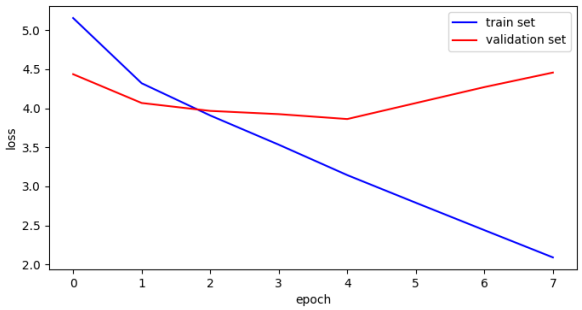
在這個實驗的範圍內，3 層的 Bi-GRU 的 performance 最好。

- **Sliding window approach**:
  在實作時，改變 window size 這個參數以求取得最佳 F1 成績。沒有模型選擇的評斷標準因為不會有 validation 的值(直接作用在 test data 上)，也因此直接以 kaggle 成績作為實驗的結果。

Window size	F1 score (kaggle public)
25	0.13113
35	0.13685
50	0.13015

- **Sentence select & adjust approach**:
  這個實作方式基本上沒有參數可以供調整。傳到 kaggle 上成績為 0.23184。
  以下討論有關選出相似度最高的片段的方式的其他方法:
  原本的方式是直接以與題目重複的詞數作為評斷依據選出最佳句子去除與題目重複詞輸出。但是可以再將選出的最佳句子再用 window sliding 搭配 word embedding 跟 cosine similarity 找出與題目相似度最高的片段。但是最後實作出的結果卻非常的差(0.03)而且比較費時(一題約花費 2 秒完成)。推測可能理由為在選定的句子中再做這樣的 attention 讓總輸出 Sequence 變短時，同時也可能讓涵蓋到的正確答案被濾除，這兩個因素一個使表現上升，另一個使表現變差。兩者的影響由這個 attention 的方式決定，而直接比較 word embedding 的 cosine similarity 可能較不利於選中正確涵蓋答案的位置，因此使 F1 成績降低。

- **FastQA model:**
  <u>Training process :</u>



如右圖，FastQA model 在 training process 的速度十分快速，僅僅只是在第 4 個 epoch 開始就出現 overfitting 的現象。Loss 最低來到約 3.86。

Experiment :

以下實驗之 loss 皆選取 val_loss 最低時的 loss。

(1) 使用 LSTM 與 GRU 之差別：

在 encoder 中改變使用的 RNN layer，觀察結果的表現。

	Train set loss	Val set loss	Public F1 score
LSTM	3.14401	3.86156	0.26155
GRU	3.87393	3.93406	0.24561

由結果得知 LSTM 在此模型中有較好的表現，不僅 loss 較低，在 kaggle 上的 F1 score 也拿到了較好的分數。在 FastQA 的論文中也是選用 LSTM 作為他們的 encoder，看來是有道理的。

(2) 更改 word vector 的維度：

	Train set loss	Val set loss	Public F1 score
128 維	3.54657	3.85806	0.24815
300 維	3.14401	3.86156	0.26155
500 維	3.83727	3.98192	0.22459

由以上結果觀察，300 維 word vector 是表現比較好的，這也是一般常看到在做 word embedding 時的建議長度。太多或太少都會使模型預測準確度下降。

(3) 更改文本 padding 的長度：

	Train set loss	Val set loss	Public F1 score
30 個詞	3.44805	3.73019	0.23915
50 個詞	3.73930	3.86894	0.24345
100 個詞	3.14401	3.86156	0.26155
150 個詞	3.60623	3.86927	0.23678

從實驗結果觀察，除了 padding 長度 30 的模型的 val_loss 較小外，其他三者的 validation loss 都十分接近。可能是由於在 validation set 當中，長度小於 30 個詞的句子較多，較能將預測範圍限縮在其中導致。然而在 public set 上就沒有表現出較好的成績。另外，可以發現其他的模型所得到的 F1 score 大概都在 0.24 左右，只有 padding 長度 100 的模型(也就是本次繳交的模型)特別高分，就算在其他實驗中也是如此。可能是由於該模型在 train 的時候剛好特別去 fit 到了 public set 的關係，可能在 private set 不見得會拿到那麼高的分數。