題目：**Delta Chinese QA**

組別：**NTU_b04901060_大腿和他的小腿們**

成員：**b04901060 電機三 黃文璁**（分工：
1. Report: R-NET experiments
2. Implementation: Naïve model

b04901061 電機三 蔡忠紘（分工：
1. Report: R-net model

b04901080 電機三 戴靖軒（分工：
1. Report: Naïve model
2. Report: Naïve model experiments

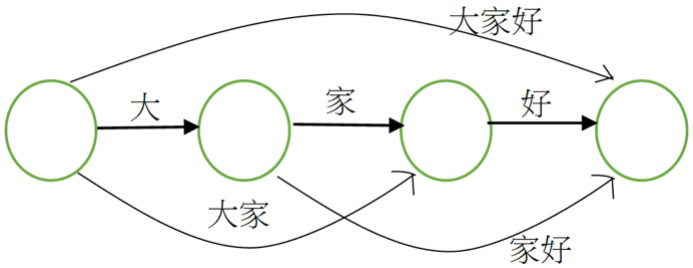
b04901108 電機三 翁齊宏（分工：
1. Report: Preprocessing
2. Report: R-net model

一、Preprocessing / Feature Engineering

本次 Final Project 中使用了兩種模型，分別為 naïve model 和 Microsoft Research 提出的 R-NET 架構**[參考資料]**，其中 naïve model 並沒有使用到中文斷詞或 word embedding 等技術，詳細做法請參考第二段 Model description 部份。而 R-NET 的部分則使用到 jieba 的中文斷詞以及 gensim 中的 word2vec 演算法進行 word embedding。以下簡單介紹這兩個技術的原理。

（一）jieba**[參考資料]**

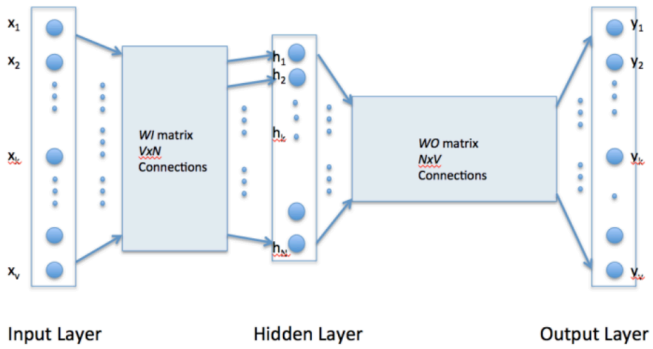
jieba 中自帶有字典，先根據字典生成 trie（字典樹），再比對需要分詞的句子生成 DAG（有向無環圖），DAG 中包含了多種分詞的可能，利用動態規劃的方法找到機率最大的可能，接著對於該種分詞方式內不在字典中的詞組以 HMM（Hidden Markov Model）並採用 Viterbi Algorithm 算出該詞組最可能的分詞方法。以下為句子「大家好」的簡單 DAG 範例：



（二）Word2vec**[參考資料]**

Word2vec 是將 word 投影到相對低維空間中，以向量表示這些 word，並且能從向量間的關係，得到對應 word 之間的關係。

下圖為 word2vec 架構示意圖：



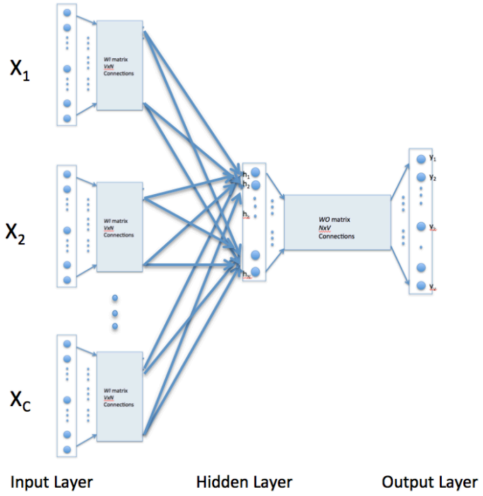
Input Layer 與 output Layer 的 V 等於 dictionary 的大小，Hidden Layer 中向量 $\boldsymbol{h}$ 的維度 N 則是投影至低維空間的維度。Input 輸入一個 one-hot encoding 的向量，代表前一個詞，經過已 train 好的 $\boldsymbol{WI}$、$\boldsymbol{WO}$ matrix 再通過 output Layer 的 sigmoid activation，得到最接近 1 的 y_i，對應到的 one-hot encoding 的詞即為預測的下文。當兩個詞有相同的預測下文時，代表兩個詞彼此之間有相關性，由於是通過相同的 $\boldsymbol{WO}$ matrix，所以兩個詞的 Hidden Layer 的向量 $\boldsymbol{h}$ 也會有相關性，因此可用 $\boldsymbol{h}$ 來表示 word。

接下來的問題是如何訓練 $\boldsymbol{WI}$ 與 $\boldsymbol{WO}$ 兩個矩陣？我們可以從已知句子的前後文來訓練 $\boldsymbol{WI}$、$\boldsymbol{WO}$。

設 $\boldsymbol{h}_{1\times N} = \boldsymbol{X}_{1\times V} \cdot \boldsymbol{WI}$，$\boldsymbol{Y}_{1\times V} = sigmoid(\boldsymbol{h}_{1\times N} \cdot \boldsymbol{WO})$，令 $\boldsymbol{WI}$ 的每個 row 為 $\{a_1, a_2, \dots, a_v\}$，$\boldsymbol{WO}$ 的每個 column 為 $\{b_1, b_2, \dots, b_v\}$，當 input（前文）的詞是一個在 x_j 為 1 的 one-hot encoding 向量，而 output（後文）的詞是一個在 y_k 為 1 的 one-hot encoding 向量時，則需使 $a_j \cdot b_k$ 變大（即縮小兩向量間的夾角），並讓 a_j 與其他 $\boldsymbol{b}$ 向量，b_k 與其他 $\boldsymbol{a}$ 向量內積變小（即增加兩向量間的夾角），用所有已知的句子來反覆更新 $\boldsymbol{a}$、$\boldsymbol{b}$ 向量，即可訓練出 $\boldsymbol{WI}$ 和 $\boldsymbol{WO}$。由於 input 為 one-hot encoding 向量，所得到的 $\boldsymbol{h}$ 即為 $\boldsymbol{WI}$ 的 row，即可用 $\{a_1, a_2, \dots, a_v\}$ 來表示 word。（更新向量的數學可參考 reference 的說明）

不過通常 word2vec 不只是用一個詞預測下一個詞，目前常見的模型大致可分為兩種：

1. CBOW：用上下文預測，數個 input 向量依序通過對應的 $\boldsymbol{WI}$，得到加總的 $\boldsymbol{h}$，再通過 $\boldsymbol{WO}$，如右圖。
2. Skip-gram：用一個詞預測上下文，input 通過 $\boldsymbol{WI}$ 後得到 $\boldsymbol{h}$，$\boldsymbol{h}$ 再依序通過多個 $\boldsymbol{WO}$，得到多個詞。



（三）本次實作細節

關於本次的 naïve model 實作，其中有幾個參數可以調整，請參考第三段 experiments 部份的實驗結果。

另外關於 R-NET 的參數，我們用於 jieba 的字典為 jieba 於 github 上提供的 dict.txt.big，更適用於繁體中文斷詞。而 word2vec 使用的套件為 gensim，維度為 300，使用的是來自[參考資料] pretrain 好的 word vector。

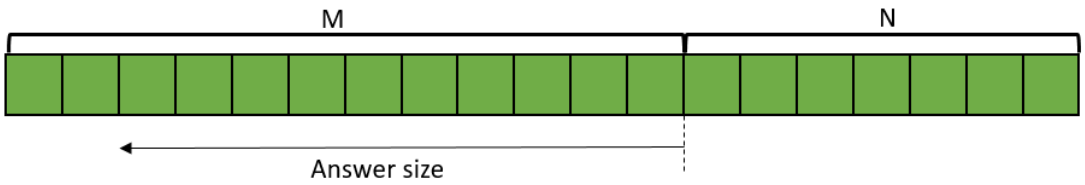
二、Model Description

（一）Naïve Method（Sliding WIndow）

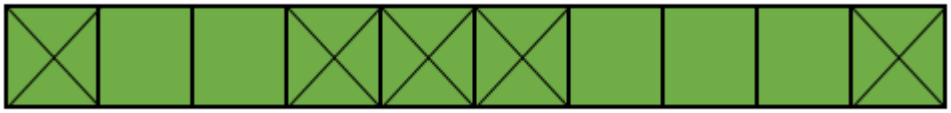
正常人在做 QA 的時候通常是先看完整篇文章，然後再回答問題。然而這並不是做 QA 最有效率的方式，若是想要加快回答速度，可以改成先看問題的敘述，再直接找出文章對應位置所提供的解答，雖然並未看完整篇文章直接回答問題有機會找到錯誤的段落，但在時間上可以取得相當大的優勢。這種先看題目再看文章的方式是這個做法的核心概念。

一般來說答案應該會在題目關鍵字的附近，所以我們先建立一個 window 掃過整個文章（如下圖，其中 window size 是可自訂的參數），比對 window 中的文字與題目的相似程度，作為一個簡單的實現方式，當然不是用 cosine similarity 之類的做比較，而是記下 window 內與題目中相同的字（不計重複）的數目，如此便找到了答案可能會出現的範圍。

在找到大略的範圍後，我們再做一個假設「答案會出現在 window 的前段」，所以我們在最有機會出現答案的地方取一個位置，並把一定長度以前的文字通通當作答案（如下圖，其中 MN 的比例與 answer size 也是可以自訂的參數）。



我們為了確保答案會落在 answer size 之內，通常會把這個值設的大一些，但是若答案涵蓋的範圍相當大，F1 score 則會變低，所以必須找到一個方法 trim 回報的答案範圍，對於正常的問題「世界第八大港口在中國的哪？」，答案「廣州」是不會在題目之中的，所以我們便把所有在答案範圍，且文字與題目相同的部分通通移除（如下圖），這麼一來能夠改善回答的精確度以提升 F1 score。

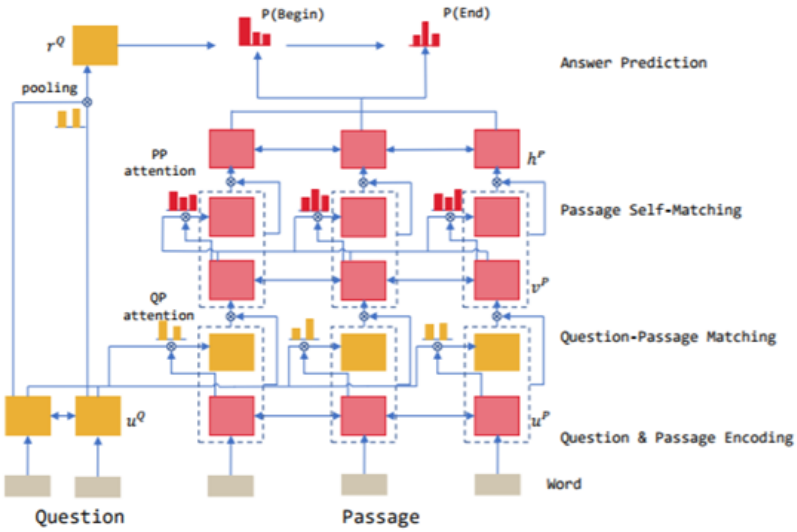


▲ 由於答案不用連續，故移除題目中出現過的字提高 F1 score

如此下來可通過 strong baseline。Kaggle 上 Public F1 score 為：**0.16779**
至於上面提到的參數對結果有什麼影響可參考第三段的實驗結果。

（二）R-NET

以下為 R-NET 論文中的架構：



接著我們整理出 R-NET 的大致原理：

1. Question and Passage Encoder

首先對所有 question 和 passage 做 preprocessing（本次中文版本為使用 jieba 斷詞，和 gensim 的 word2vec 進行 word embedding）得到 e_t^Q 和 e_t^P，為了使這些詞與上下文相關，將兩者分別通過三層 BiGRU 得到 u_t^Q 和 u_t^P。RNN 有 memory 的特性能符合 NLP 需要，而目前論文多使用 GRU 是因為結果和 LSTM 相似但有計算量更為精簡的優勢。

2. Gated Attention-based Recurrent Networks

接著我們要講 question 與 passage 的資訊相結合，在此使用 match-LSTM，先對 u^Q 做 attention pooling（基於 u_t^P、v_{t-1}^P）得到 c_t，再將 c_t 和 u_t^P 通過 RNN 得到「Question-aware passage representation」v_t^P。

由於 passage 內僅有一部分與 question 有關，為了篩選這些段落，會在 match-LSTM 中多加一個以 $[u_t^P, c_t]$ 為 input 的 gate。

3. Self-matching Attention

Q 和 P 可能有很不同的用詞／句型，所以做 attention 的部分除了與 Q 有關，也必須與 P 本身有關，因此我們對 v^p 做 attention pooling（基於 v_t^P）得到 c'_t。將 v_t^P 與 c'_t 一起通過 BiRNN 得到 h_t^P，如此一來 h_t^P 便同時考慮了 Q 和 P 的資訊。

4. Output layer

因為我們希望 output 是起點與終點，所以使用 pointer network。Pointer network 的架構類似 seq2seq，但輸出長度可根據 input 長度不同而有改變，適合處理 QA 問題。將 attention 過程中的機率當 pointer，讓 network 在不同的 t 時，指向當時 encoder 中機率最大的 cell，從起點逐一對 passage 做 attention，一直到 end cell 機率最大時結束，由此可標出 passage 中起點和終點的位置。

Answer Recurrent Network 以 attention-pooling vector c_t 作為 input，並以對 Q 做 attention 得到的 r^Q（$r^Q = att(u^Q, V_r^Q)$）作為 RNN 的 initial state。在 training 的過程中，以最小化 ground truth 開始和結束位置的 negative log probabilities 為目標。

5. Attention mechanism

$c_t = att(h^P, h_{t-1}^a)$，而 $h_t^a = \text{RNN}(h_{t-1}^a, c_t)$。在 attention 過程中，$c_t$ 是 passage/question representation 的線性組合：

$c_t = \sum_{i=1}^{n} a_i^t h_i$

$p^t = argmax(a_1^t, a_2^t, \dots, a_n^t)$

h_t^a：Pointer network hidden state

a_i^t：Predicted probability

h_i：Passage/question representation

p^t：Output (begin and end of the answer)

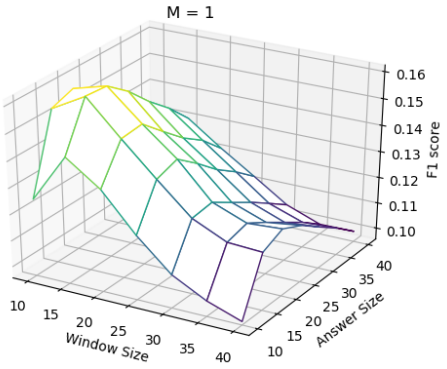
三、Experiments and Discussion

（一）Naïve Method Parameters

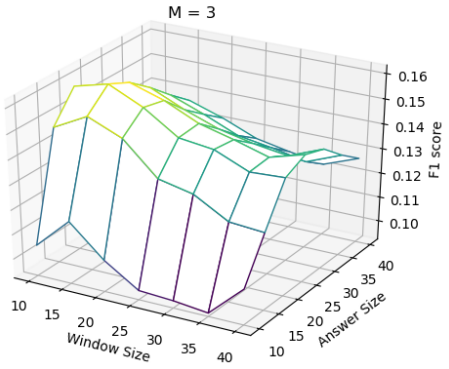
由於前面提到的 Naïve method 是不需要 training 的，故我們直接用 training data 來進行以下實驗，更改先前提到的三種參數：

1. Window size
2. Answer size
3. M/N ratio

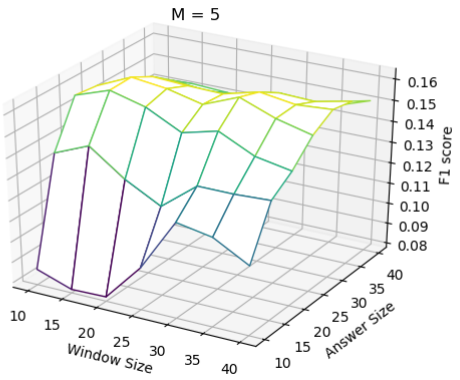
以下每張圖固定 MN 的比例，然後對 window size 和 answer size 作圖（利用 training data 中前 1000 個問題計算 F1 score）



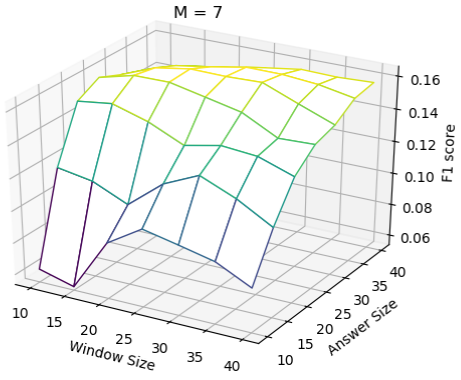
▲ M：N＝1：9



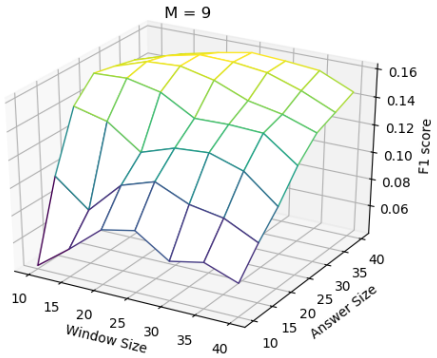
▲ M：N＝3：7



▲ M：N＝5：5



▲ M：N＝7：3



▲ M：N＝9：1

■ **討論：**

M:N = 1:9 和 M:N = 3:7 有類似的趨勢，可以看出最大值約在 window size 和 answer size 都偏小的位置，推測是答案並沒有離所選取到的 window 太遠，而且取答案的時候在偏前方的位置開始，所以 answer size 不用太大。至於為什麼會需要偏小的 window size，應該是因為若 window 太大，為了滿足與題目最相似的條件，答案的位置可能已經在起始位置的後面了，所以 window size 增加不會提升 F1 score。

剩下的圖則有另一種趨勢，最大值大概在 answer size = window size + 10 的線上，這在某種程度上印證了我們的假設：「問題的答案在題目的前面」，所以 answer size 要比較大才能包住正確答案的位置。我們取的起始位置偏後方，就算解答的範圍跟 window 大量重疊，藉由前面的步驟把題目中出現的字剔除，還是可以得到比較精準的答案。

■ **小結：**

這是一個簡單的方法，用了一些假設和技巧可以勉強可以提升 F1 score 至 strong baseline，但是相較於 ML 的做法差距就顯得相當顯著，這也顯示了 ML 的表現在這類不易直接計算的問題上比起直接寫死規則還要好上許多，印證了 ML 的強大之處。

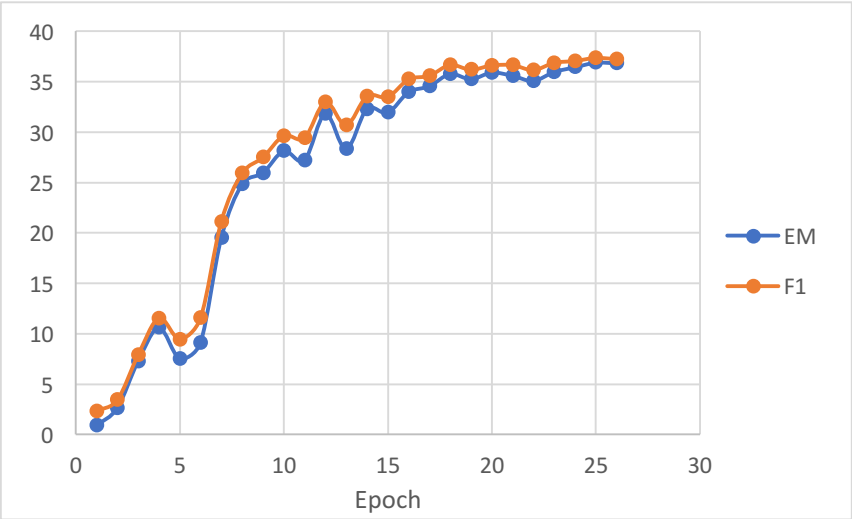
（二）R-NET

■ **實作**

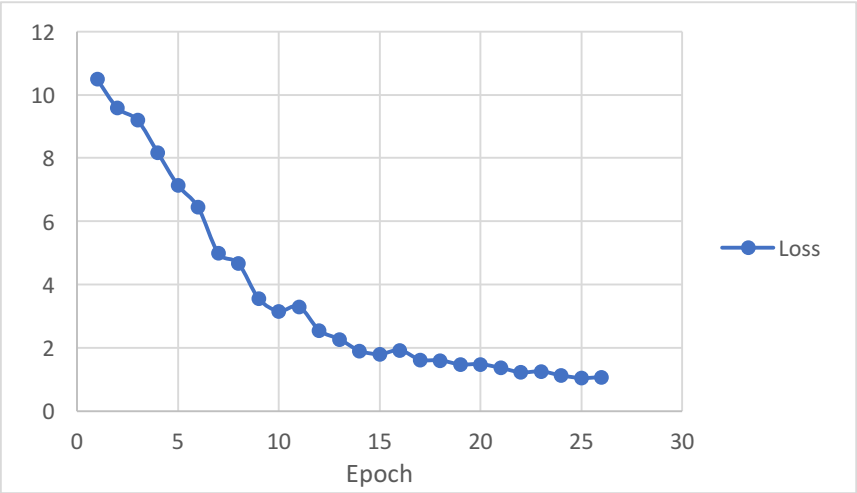
本次 R-NET 實作我們修改了 github 上的其中一份 PyTorch 實作 **[參考資料]**，由於該份實作是用於英文的 SQuAD dataset 上，故要做一些修改，例如將斷詞改為 jieba，word embedding 從原實作中的英文 Glove 改為中文 word2vec，其他部分大致上仍可以直接採用。

■ **Training 過程**

Training 時由於不像 SQuAD 有提供 dev set，故直接從 training set 中切 10% 當作 validation set。以下為 training 時每個 epoch 的 loss 和 F1 score 作圖：



▲ Training 過程中 validation set 的 EM 和 F1 score



▲ Training 過程中的 loss

■ 結果

由於 R-NET 的訓練過程在我們的機器上較為漫長，一個 epoch 大約需要 40 分鐘，故沒辦法做更完整的實驗，不過從 validation 的結果來看，訓練過程中準確率確實有逐漸提升。

最後將 test set 的結果上傳至 Kaggle，Public F1 score 為：**0.17538**，超過了原本的 naïve method。

四、參考資料

[1]