Chinese QA

Team name, members and your work division (1)

我們一開始決定對三個題目平行工作，一人一個題目，到後半段的時候決定好最終題目後，再合力專研這個題目，以下是我們準備 final project 期間中各自的分工。

隊名		
NTU_r06922118_IRLab		
學號	姓名	工作內容
R06922118	吳政軒	TV-conversation project、研讀 paper、寫 report
R06944032	倪溥辰	Listen and Translate project、寫 report
R06944049	黃敬庭	Chinese QA project、code、寫 report

Preprocessing/Feature Engineering (3)

preprocess 的部分我們用所有的 training data，包括 passage 和 question，利用 gensim 及 jieba 分詞訓練一個 word-level embedding model 和一個 character-level embedding model，word-level 的 embedding 為 300 維而 character 為 8 維。最後的 word embedding model 大概包含了 110000 個左右的單詞和 5500 左右個單字。

另外我們將 training data 切了一塊 validation data，training data 為 13023 筆，validation data 為 1588 筆。訓練的時候我們參考助教的 sample code 與網路上的 squad 模型將一筆一筆資料包為一個一個的 example，裡面包含的資訊有文章、標題、id、問題、答案 span 等，在訓練 R-Net 的時候我們以前後兩個 span 的 loss 平均作為 model 的 loss。

Model Description (At least two different models) (7)

1. Cosine similarity:

我們使用的第一個 model 是用 cosine similarity 來判斷答案，方法如下：

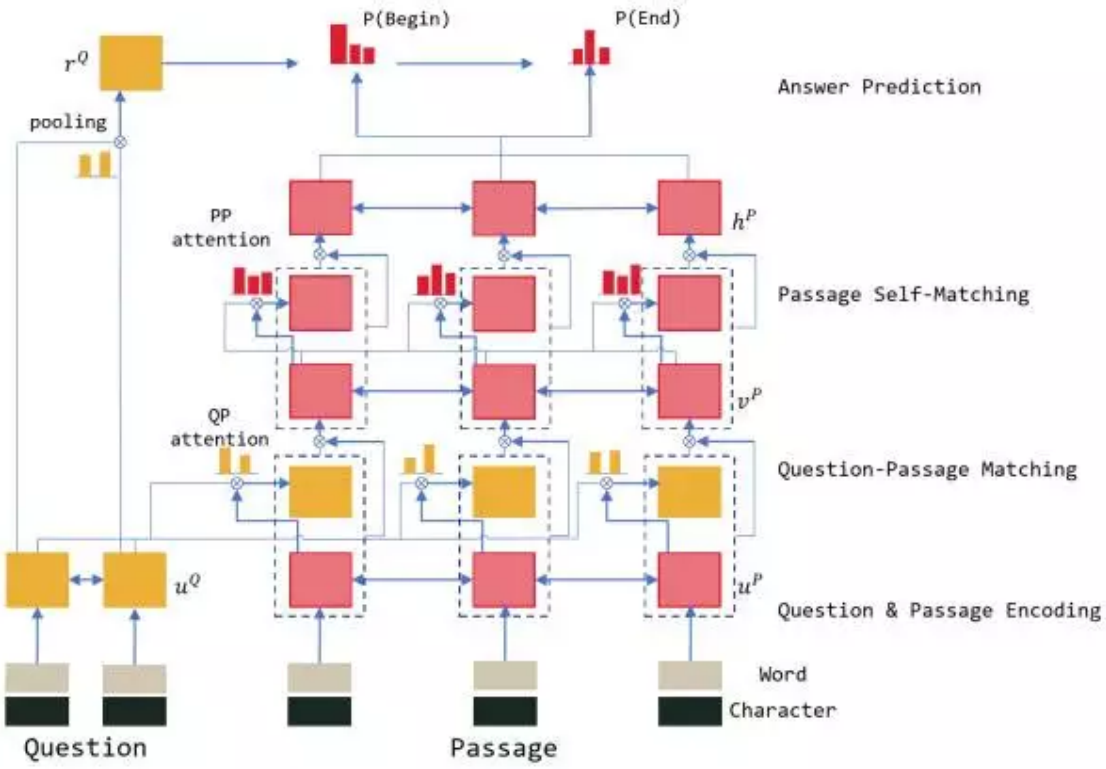
取固定 window size，也就是答案的長度，將整個 passage 從頭掃到尾，可以得到若干個 windows，然後比較哪個 window 和 question 的 cosine similarity 最高，取最高的當作答案。

而每個 windows 和 question 的 similarity 計算方式如下：

將 windows 內的每個字和 question 內的每個字加總 cosine similarity，就是這個 window 的 similarity，例如 window 內為「'a'、'b'、'c'」，question 為「'm'、'n'」則 window 的 similarity 為
sim(a,m)+sim(a,n)+sim(b,m)+sim(b,n)+sim(c,m)+sim(c,n)。

因為這個 model 考慮的資訊太少，只有比較字和字之間的相似度，並沒有考慮到前後文的資訊，另外每個字的權重也都相同，無法著重在重要的字上面，整個 model 幾乎就取決於 embedding model 訓練的好壞，而且可能是因為我們實做方法的關係跑一次要 3 個多小時並且成效也有限，所以我們決定要找尋其他的解法，上網找了許久與參考老師給的 Squad 網頁後，最後決定使用 R-Net 做為我們的第二個 model。

2. R-Net:



R-Net 是由微軟亞洲研究院的 Natural Language Computing Group 所發表，模型的架構如上圖所示，是一個多層的 neural network 架構，分別從四個層面對整個閱讀理解的演算法進行建模。人類在做閱讀理解的過程中，一個常見的順序是這樣的：首先我們會先閱讀整篇文章，對文章有一個初步理解之後再去看問題，從而對問題也有了一定認知。第二步，可能就需要將問題和文章之中的部分段落和內容做一些關聯。例如題幹中出現的某些關鍵已知信息（或證據），找出一些候選答案，舉例來說：如果問題問的訊息是時間，那麼文章中出現與時間相關的訊息就有可能是候選的答案。第三

步，當我們將候選答案與問題進行對應之後，我們還需要綜合全文去看待這些問題，進行證據的融合來輔證答案的正確性。最後一步，就是針對自己挑出的答案候選進行精篩，最終寫下最正確的答案。

而 R-net 本身也是基於這樣的順序，首先我們先將 Question 和 Passage 分別使用其 Word Embeddding 和 Character Embedding 作為輸入(character-level 的 embedding 主要是为了對應 OOV 的影響，以往 OOV 的 vector 直接就是 0，這裡可以減緩 OOV 的影響)，接著在 GRU 中產生 Question 和 Passage 的 embedding(u_t^Q和u_t^P)。這一步就是 Question&Passage Encoding。

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

第二步使用 gated attention-based RNN，獲得 question aware word representation(v_t^P)。這裡說的 gated-attention RNN，是在 RNN 和 attention 基礎上增加一個 gate。這一步被稱為 Question&Passage Matching。

$$v_t^P = \mathrm{RNN}(v_{t-1}^P, [u_t^P, c_t])$$

而 gate(g_t)的用處主要用於控制 passage 表示的不同部分重要性，這裡我們可以理解為將問題中的向量和文章中的向量做一個比對，這樣就能找出問題和哪些文字部分比較有關連

$$g_t = \mathrm{sigmoid}(W_g[u_t^P, c_t])$$
$$[u_t^P, c_t]^* = g_t \odot [u_t^P, c_t]$$

第三步與第二步的結構大致上是一樣的，可以想成是：以目前現在針對問題對文章的理解，再看一遍文章，更新我的理解。這個更新後的理解就是h^P，這個步驟被稱為 self-matching。

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

最後一步，使用 Pointer-Network 方法，即直接將 attention 向量當做候選答案位置的機率，總共輸出兩個位置：答案的起始(p1)和答案結束(p2)

$$s_j^t = \mathrm{v}^{\mathrm{T}} \tanh(W_h^P h_j^P + W_{\mathrm{h}}^a h_{t-1}^a)$$
$$a_i^t = \exp(s_i^t) / \Sigma_{j=1}^n \exp(s_j^t)$$
$$p^t = \mathrm{argmax}(a_1^t, \ldots, a_n^t)$$

實作上我們主要參考這篇論文的架構，不過在細節上我們參考了其他的 R-Net 實做做了一些調整。

Experiments and Discussion (8)

這個部分我們是針對 R-Net model 做討論。

訓練過程：
訓練時使用 adam optimizer，batch size 設為 64 (再上去就有機會 OOM QQ) ，在 encoding、attention、pointer layer 裡面都使用 dropout 來避免 overfitting，並在每 200 個 iteration 記錄數據與更新 learning rate，當過 200 個 iterations 而 model val loss 沒有進步的時候我們就將 learning rate 調為一半，但比較神奇的是我們發現 val f1 score 與 kaggle 的 f1 score 有很大的落差，我們的模型會在約 8000 ~ 8500 iteration 的時候停止進步，而這時候 val f1 score 大約為 56 而 kaggle 的 f1 score 會在 66，隨著訓練繼續進行 val f1 score 會停在 53 左右而 kaggle 成績會在 59，我們猜想可能是因為我們的 validation 資料(training 的後面 1588 題)的分佈與 test 資料的分佈不太一樣導致如此。

另外我們也做了一些實驗，針對 word embedding dimension、character GRU hidden size、word GRU hidden size、encoding GRU layer 做了些微調整，以下是我們的實驗結果整理，並且在最後附上實驗結果的圖。

Best model 的參數為

- word embedding dimension 300 維
- character GRU hidden size 100
- word GRU hidden size 75
- encoding GRU layer 3 層

word embeding dimension

dimension	Train F1 score	Val F1 score	Train loss	Val loss
50	69.97	53.39	0.60	3.33
100	72.34	52.93	0.53	3.48
300(Best)	80.48	55.03	0.26	3.64

由以上表格可以發現當 embedding size 為 50 或 100 時，表現比較差，表示 50 或 100 維還不足以學習字和字之間的關係，不過 embedding size 也不能一直往上增加，那樣很有可能會學習到很多 noise，因為時間的關係，我們並沒有做更高維度的實驗，這也是後續可以進步的地方。

character GRU hidden size

size	Train F1 score	Val F1 score	Train loss	Val loss
50	73.51	53.20	0.49	3.62
100(Best)	80.48	55.03	0.26	3.64

word GRU hidden size

size	Train F1 score	Val F1 score	Train loss	Val loss
25	51.04	41.84	2.59	3.98
50	64.20	50.90	1.23	3.30
75(Best)	80.48	55.03	0.26	3.64

GRU hidden size 在我們的實驗中是呈現正相關的，hidden size 較高表現也會比較好，其中比較值得注意的是 character 和 word 兩部分的差異，由上表可以在相同的 size 變化量(50->100，25->75)，word 的部分差距比較大，應該是因為我們 character 的部分主要是用來處理 OOV 的問題，大部分是用 word，所以 word 影響較劇烈。

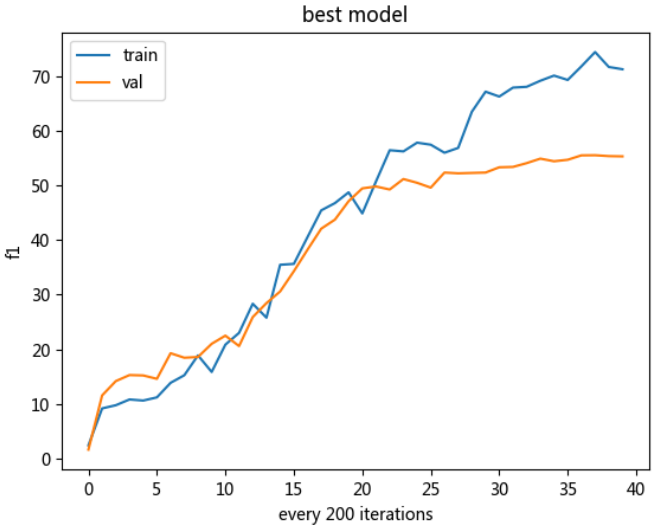
encoding GRU layer

layer	Train F1 score	Val F1 score	Train loss	Val loss
1	67.52	50.68	0.97	3.23
2	76.15	53.59	0.59	3.27
3(Best)	80.48	55.03	0.26	3.64

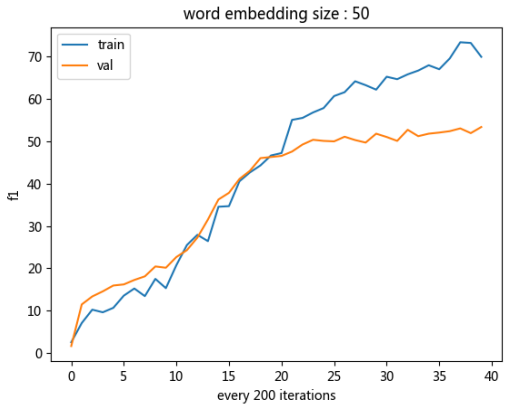
我們發現當 GRU 多層時的表現比較好，我們認為是因為多層的 GRU 可以記得比較多資訊，整個 model 比較 powerful，所以表現會比較好。

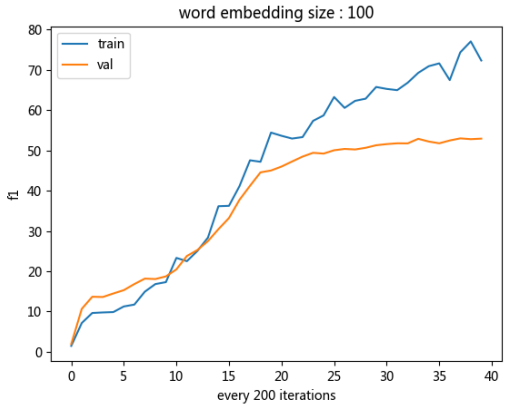
底下是所有實驗畫出來的圖：

Best Model:

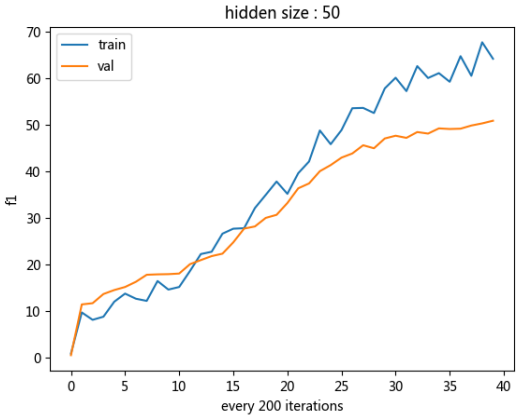


word embedding dimension:

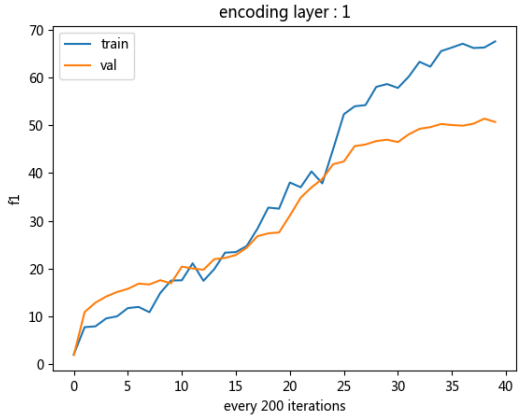


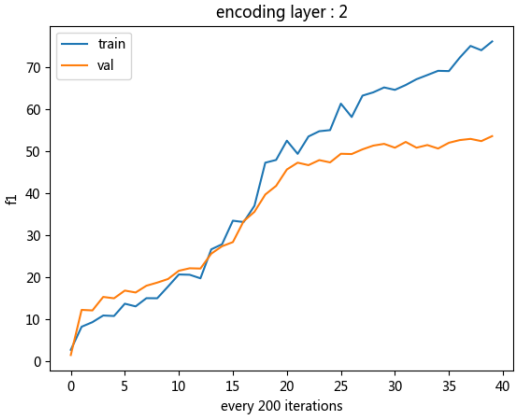


GRU layer hidden size:

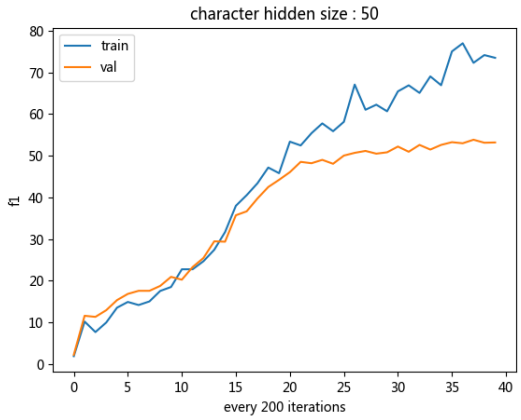


GRU Layers number:





Character GRU layer hidden size:



Referrence:

Rnet 的 論文
https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/r-net.pdf
數個 tensorflow & keras 的 rnet 實做
https://github.com/minsangkim142/R-net
https://github.com/unilight/R-NET-in-Tensorflow
https://github.com/HKUST-KnowComp/R-Net
https://github.com/YerevaNN/R-NET-in-Keras