深度學習應用-期未報告(Machine Comprehension)

R04725040 黃柏睿, R04725042 葉展奇

1. INTRODUCTION

Machine comprehension 指的是讓我們撰寫的程式，依據給予的文章內容以及問題，能夠產生相對應的答案。本次的期未專案所使用的資料集是 SQuAD 以及托福的題目，兩個資料集的問題型式都是選擇題，而我們會比較 model 產生出來的答案以及四個答案，選出最大可能性的那個選項。

我們參考了老師提供的 paper[1]以及作者在 github 上發佈的 code[2]，attention 機制的出現賦與 MC 更強的能力，有別於傳統的 uni-directional attention，這篇 paper 將多個 NN 串在一起，提出了一個多階層的處理流程，以表示不同 granularity 的內文，以及使用了雙向(bi-directional)的 attention 機制使得系統可以直接找出在內文當中與問題相關的句子當作答案來完成回答問題的任務。

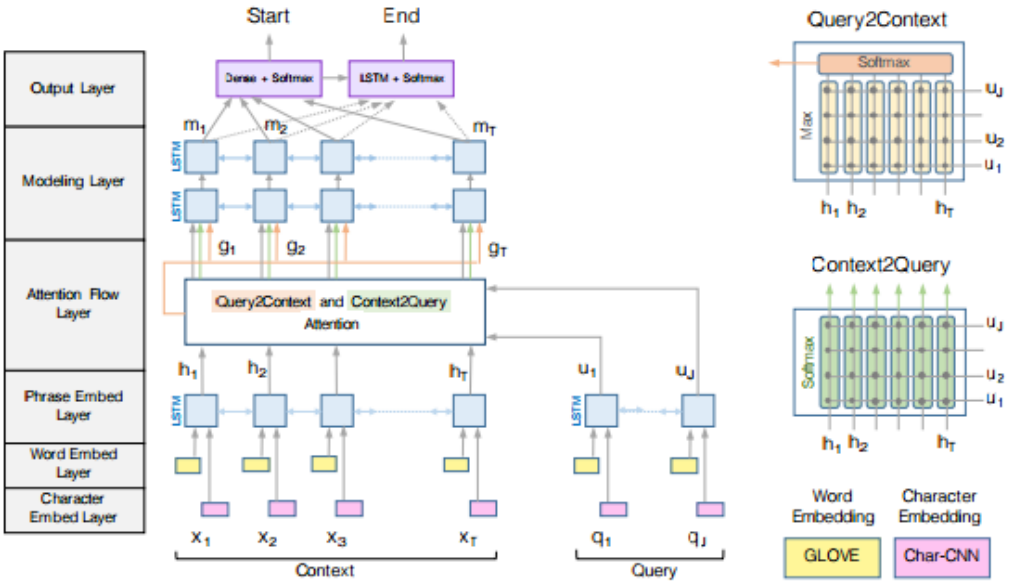

Figure 1: BiDirectional Attention Flow Model *(best viewed in color)*

2. Model:

BIDAF 總共分成六個 layer:

I. Character Embedding Layer:

我們會將文章內容以及問題連接在一起，作為這個 model 的輸入，這一層做的事情，是要將每一個字利用 CNN 轉換成為一個高維度的向量。

II. Word Embedding Layer:

第二層要做的事情一樣是要將每個字轉化成為一個高維度的向量，我們使用的是一個 pre-train 的 GloVe word vector。

III. Phrase Embedding Layer:

使用了 bi-direction LSTM 去接收從第 1, 2 層傳過來關於每一個字的 vector，並將每一個 cell 的 output 傳遞給第四層。

IV. Attention Flow Layer:

在這一層系統會對 Context2Query(C2Q)及 Query2Context(Q2C)使用 bi-direction attention 的技巧，C2Q 著重在找出哪個 query word 與 context 最相關，而 Q2C 著重在找出哪個 context word 是可以回答該 query 的最關鍵的字。之後將上一層的輸出(phrase-level embedding) 以及得到的 attention vector 合併，再傳給下一層。

V. Modeling Layer:

這一層則是利用 2 個 LSTM 去讀取整個 context(這個 context 已經是 query-aware 的 context 了)。

VI. Output Layer:

利用 softmax 預測出在文章中找出某一個段落以作為問題的答案的起點(第幾個字)以及終點，並預測出答案。

3. Method:

由於這個 model 輸出的答案是根據問題，尋找文章中較為相符的一個段落，因此我們還要撰寫一個函式比對輸出答案以及 testing data 中的四個選項，去計算分別的 similarity，之後 output 出最相似的選項。

一開始我們寫了一隻程式分別去計算四個選項以及 NN 預測出來的答案的包含關係，也就是去計算每個選項包含了幾個預測出來的答案(以字為單位)，後來查了一些 python 內計算兩個 string 相似度的方法，最終我們使用了 Levenshtein ratio[3](必須額外 pip install python-levenshtein)來計算兩個字串的相似度，最後將 Levenshtein ratio 最高的選項當作答案並寫到 answer.txt 裡。

而計算 Levenshtein ratio 必需先計算 Levenshtein distance ，它是由 Vladimir Levenshtein 提出的方式，是描述由一個字串轉化成另一個字串最少的操作次數，其中的操作包括插入、刪除、替換。Levenshtein ratio 是(sum- Levenshtein distance)/sum 的值，其中 sum 是兩字串的字數相加，值域為[0-1]，越大代表兩字串越相似，在後來的實驗當中這樣的計算相似度方法，也比只接算包含關係的方式更能夠找到關聯性最高的選項，讓正確率上升。

4. Preprocess:

由於我們的 data 跟作者原始設計的不一樣，首先需要把正確的選項轉為選項的文字，以及他在文章中出現的位置。這裡採用直接比對的方式，只要文字有出現在文章中，我們就將它視為答案出現的位置。

托福的部分，由於選項並不是直接出現在文章中，我們一樣運用 Levenshtein ratio 來比對文章中每一句話與正確選項相似度，標記最相似的句子為正確答案進行訓練。

5. Environment:

- Ubuntu 14.04

- Python 3.5.2 (無法執行 3.4.x)
- Tensorflow: 0.11.0
- Other Packages: tqdm, python-levenshtein

6. Results：

Squad：Model 大約在 8000 個 steps 左右會有最高的分數，超過之後分數就開始降低，可能是 overfitting 的問題。

Toefl：也是大約在 9000-10000 之間收斂。

7. Performance:

Squad : 正確率約為 60%

Toefl : 正確率約為 31%

8. Conclusion:

Squad 由於 data 本身的特性（答案出現在文章中），做出來的表現相當不錯，大約在 8000 個 step 就能夠達到很好的分數。如果要改進的話或許可以嘗試使用不同的 word embedding (我們使用的是以 glove 方式，事先 train 好的 vector)，或是用更多的 features 來嘗試。

而 Toefl 的部分只能免強通過 baseline，我們猜想因為使用了別的方式來標注答案出現的地方，而這麼做的話，最後得出來的答案好壞會取決於我們標注 training data 答案的那個 model 的好壞，造成分數無法上升。由於時間的因素，最後只來得及採取較為簡單並訓練時間短的方式。未來也許可以嘗試在原本的 output layer 上面在多放上一層 Rnn，作為原本 model output 出語句輸出成，對於每個選項為正確答案的機率，並一起 train 它的參數，或許能達到更好的結果。

9. Future Work

因為這一次的時間有一點趕，我們提出了幾個可能可以增加 performance 的方式，列舉如下：

1. F1-score[4]

   F1 分數是一種同時兼顧 precision 與 recall 的度量方式，應用於 IR 領域，而我們可以利用 F1 分數去計算 model 預測出來的答案以及各個選項的相似度，而不是只是去看正確率(只計算包含的字數，這樣對長度較長的答案相對有利，因為可能包含比較多字，但並不一定是正確答案)，而 F1 就可以解決這個問題，因為加入了 recall，讓出現不包含字的情況出現 penalty，這樣我們可以更加容易的去選出正確的答案選項。

2. Ensemble

   Ensemble 的技巧其實就像是在做投票，我們在這次實驗中所使用的方式都是使用單一個 model 預測出來的結果，但是如果使用多個 model，則可以達到互補的效果，讓一大堆的 model 去投票，然後選出最多票的那一位，這樣的方式可以集結每個 model 的力量(雖然每個 model 的 accuracy 都一般般，但 ensemble 的話卻可以得到超過最好的 model 的 performance)。

3. Stop word

   或許在比較 similarity 的時候，我們可以先濾掉 stop word，這樣可以避免在算 similarity 的時候，將一些較無意義的字也包含在內。

Reference：

[1] : Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi & Hananneh Hajishirzi, *Bi-Directional Attention Flow For Machine Comprehension,* ICLR 2017

[2] : The code of BIDAF on github. Available from : https://github.com/allenai/bi-att-flow

[3]: Levenshtein ratio. Available from : https://pypi.python.org/pypi/python-Levenshtein/0.12.0

[4]: F1 score. Available from : https://en.wikipedia.org/wiki/F1_score