Report

R04922108、R04922098、R04922086、R04922065

Environment

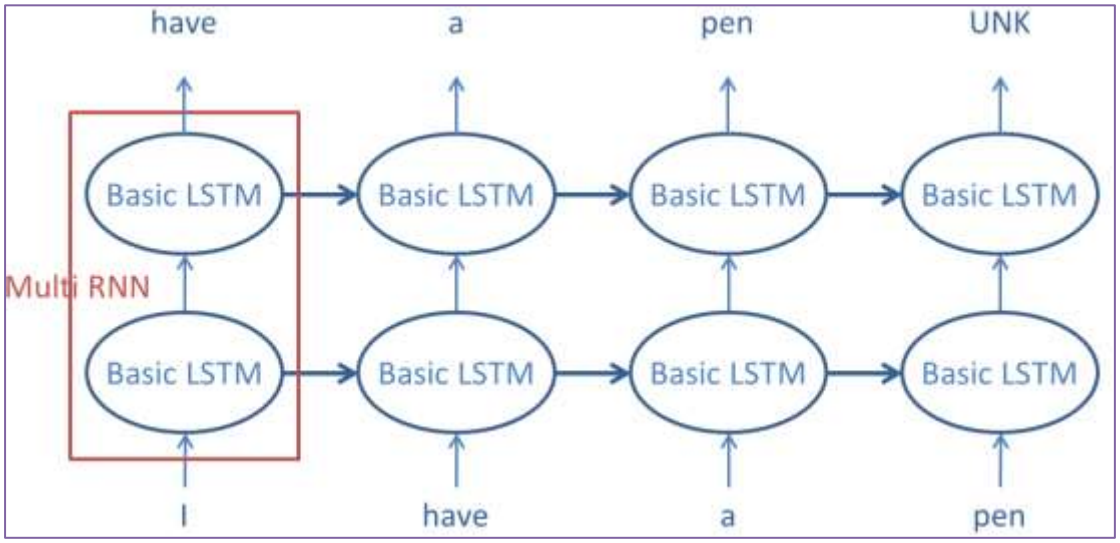
OS	=	Ubuntu 14.04	CPU	=	i76700
Lib	=	Tensorflow 1.0	GPU	=	1080
Python	=	2.7	CUDA	=	8.0

上圖為最終測試的環境，開發時有各自的環境，因此不一一列舉。

Model Description and Improvement

接下來會分別介紹我們使用的各種 Improvement 方式，以及其使用的 Model。

(1) Basic



一開始我們以 RNN 作為原始的基礎，利用 Tensorflow 的 Basic LSTM，以及 Multi RNN 架構兩層 RNN 的模型，如上圖，並且使用 Sample Loss 作為 Cost Function。因為此次的目標為有選項的預測克漏字，所以我們會藉由算出的 Output 機率陣列，找出每個選項字的對應機率，並取最大者當作答案。另外我們僅使用 One-Hot Encoding，所以為了加速我們訓練速度，我們會僅取最常出現的字，稀少的字則用 UNK 來代替。

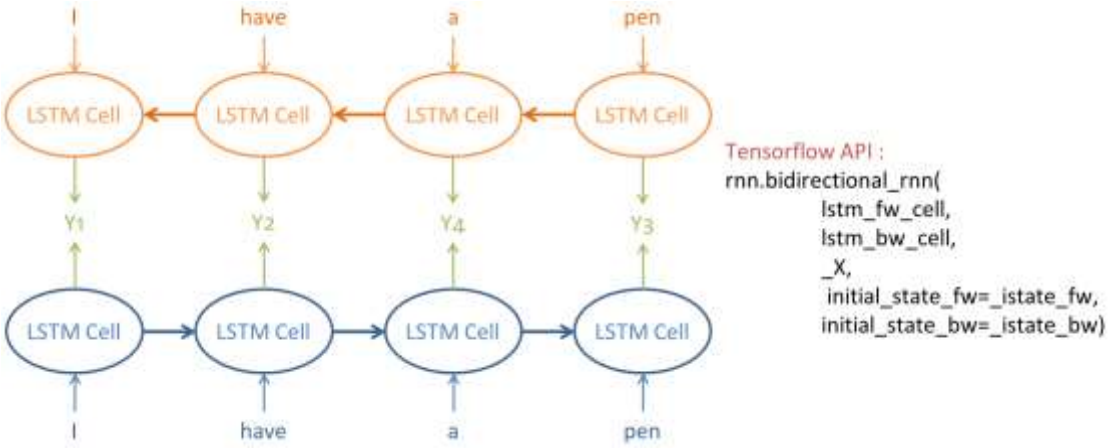
(2) Cut Sentence

先前的方法為單純把整個文章的字當作 Training Data，並依據一開始設定的 Num Steps 作為句子的長度，如上圖左。

因此這邊目標為考慮標點符號，依此來正確取出完整的句子，並加上起始符號以及補足結束符號來做為開始、結束，如上圖右。另外會限制句子的長度，以增進訓練上的速度。

使用 Cut Sentence，只單純影響到一開始 Data Set 的處理，所以 RNN Model 依然是跟隨 Basic 方式。

(3) Bi-directional RNN



此不同於單向 RNN 僅考慮前面單字的影響，他會額外考慮後面單字造成的影響，也就是看完前後文在進行預測單字，如上圖。在此直接使用 Tensorflow 的 API 來實現 Bi-directional RNN。以下圖可以發現，Bi-directional RNN 的預測效果。

```
--- Test ---
as i descended my old ally the UNK came out of the room and closed the door tightly behind him
--- Predict ---
i descended my old ally the UNK came out of the room and closed the door tightly behind him UNK
```

(4) Adding Sentence

如同前面 Bi-directional RNN 會考慮整個句子，在此一樣會考慮後續的字，但是在訓練上仍然是使用單向的 RNN。

因為已經知道選項的單字，所以我們可以將每一句 test data 在選項前的句字與五個選項合起來，再與選項後的句字合起來，ex: test data: [a b c _____ d e f], option: 1 2 3 4 5，則針對這組 test data，我們會產生五筆 data 分別為[a b c 1 d e f], [a b c 2 d e f]…等等，再利用原先 train 好的 model，針對每一筆 data 拿到 lstm 的 output 通過 softmax，得到每一個位置生成下一個字的機率，記做 P，再利用 P(選項)*P(選項後 1 個字)*P(選項後 2 個字)*….= P(選項)*P(選項後整句)，ex: test data:[a b c 1 d e f]，則這筆 data 的機率為，取 lstm 在 c 位置的 output 得到 P(1)，再取得 1 位置的 output，得到 P(d)…，最後將這幾個機率相乘，即為此 test data 的機率。

Experiment and Performance

(1) Experiment Setting

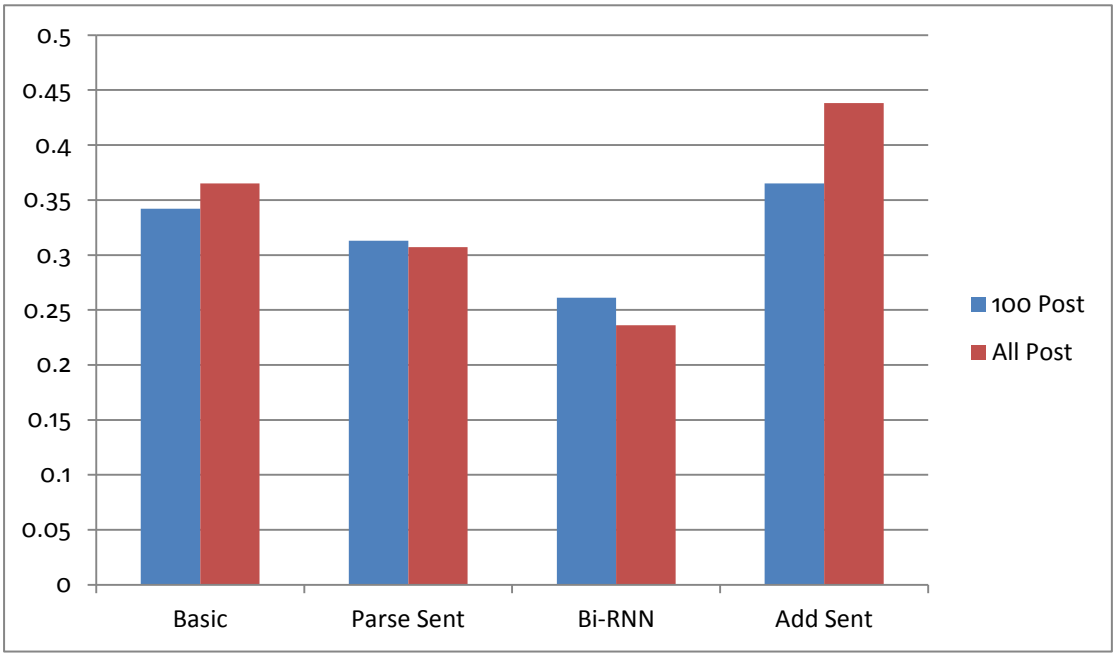
以下圖為固定的參數設定。

Cost Function	=	Sampled Softmax			
Cell	=	BasicLSTMCell	Learning Rate	=	GradientDescent
RNN Layers	=	2	Drop-out Rate	=	0.5
Initial	=	Uniform[-0.05, 0.05]	Hidden Size	=	6
Num Steps	=	35	Batch Size	=	20

接下我們會分別比較，上述的不同 Model、Training Data 的文件數、Epoch 數目、以及取的 Vocabulary 字數所對應的 Kaggle Public 的分數。

因為時間的關係，我們 Training Data 為 100 篇時 Epoch 為 10；Training Data 為全部文件，則 Epoch 為 2。

(2) Performance



我們可以發現 Add Sentence 的方式可以有效增進準確率，可能這是因為他有考慮到後續句子的關係。但是很好玩的一點在於，Bi-RNN 一樣考慮了全部的句子卻反而降低了準確率，在此我們特別去測試 Bi-RNN 發現如果降低 RNN 的 Layer 數目，反而能夠有較好的準確率，從 0.261 上升至 0.269，因此我們推測，Bi-RNN 會有過於 Overfitting，造成 Testing 上不准的原因。

而且我們可以發現，相對於 Bi-RNN，Add Sent 跟 Basic 都會隨著 Training Data 數目上升，準確率跟著上升，也更加證明，Bi-RNN 會過於 Over fitting。另外我們也嘗試過增加 Vocabulary Size，的確能在 Add Sent 時取得更好結果 0.47。

Team Division

R04922108	R04922098	R04922086	R04922065
Add Sentence Report	Bi-RNN Report	RNN	RNN