MLDS HW2

王垣尹 許哲瑋 李中原

2-1

Describe the model (3%)

有關資料預處理的部分，首先我們建立了 word to index 以及 index to word 的兩個字典，選字的標準是只要出現過的字就加進字典(min frequency = 1)，字典中總共加入 4 組特殊字元，分別是 PAD、BOS、EOS 還有 UKN，各自代表 padding、句子首尾符號以及未知符號。 Seq2Seq 模型本身我們實作了 S2VT 論文裡面的 Model，改模型主要的特色是 Encoder 和 Decoder 使用一樣的參數；在 training 的過程中，為了不讓 LSTM 困在一樣的 minimum，在每次新的 Epoch 中，我們都會 re-Shuffle 全部的 data，由於總共有 24232 句句子，如果每一個都對上一部影片的話，所需要的記憶體會非常大，因此我們也為每句句子做影片對應的字典。 輸入及輸出的部分，我們先將影片的每個 time step (4096 維的向量)乘上一個(4096 * 512)的矩陣做簡單降維，其中 512 代表的是 LSTM 的大小。句子的部分，每一個字我們都做 one-hot 的編碼，由於句子的長度多多少少都有不同，在 optimize 的過程中我們對句子加入 masking (影片因為固定大小就沒有另外做)。

Write down the method that makes you outstanding (1%)
Analysis and compare your model without the method. (1%)
Why do you use it (1%)

我們 implement 了 Attention 的機制，只是為了方便實作，對最原本的 Attention 機制做了簡化(不想用 Wrapper)，我們將 LSTM2(S2VT 裡面下層的 LSTM)裡面每個 Encoding time step 的狀態(State)都存到一個 List 裡面，之後 Decode 時，將 Decoding 的 Output 與每個 time step 的 input 做內積並正規化，之後選取內積結果最大的那個 State(對每筆資料而言從 20 個 Step state 裏頭選最大的)，與”本來 Decode 的輸出相加”當作新的輸出，由於輸出的向量

使用 Attention 機制的原因是因為 ML 就聽過這樣的技巧，但一直沒有試過來看看表現，因此想要試試看，另外兩個方法我們則實作在 2-2 Chat-bot 裡面。

與沒有 Attention 的結果相比，我們簡化版的 Attention 的版本效果並沒有比較好(甚至比較差)，我們認為可能的原因有幾點:

1. 這不完全是原本的 Attention 機制，因為我們沒有讓每個 Time step 的 Encoding state 都有表現機會，我們只取內積完之後，應該要完全 Focus 的地

方，這樣的好處是時作容易也很直觀，但壞處是我們加強了這個 State 以及他之前的 State(如果有被這個 State 正確的記住)對輸出的影響，卻讓之後的影響完全忽略。

2. 可能的第二個原因是在 S2VT 裏頭，或許要存有完整的 State 資訊必須去使用兩個 LSTM 的 State 而我們只有使用第二層 LSTM 的 State 來做 Attention 位置的判斷，所以很有可能其實不是真正的 Attention 該鎖定的位置。

Loss with attention: 6.70 (480 epochs, not lowering anymore)
Loss without attention: 4.213 (final loss, 609 epochs)
Bleu Score with Attention: 0.465
Bleu Score without Attention: 0.622

Experimental results and settings (1%)

Parameter Tuning:
有關訓練參數的部分，在試過很多種可能之後，我們選用了 Adam 當 Optimizer，初始 Learning rate 設成是 0.0001，我們有發現只要稍微比 0.005 大就會無法收斂，我們嘗試過的 Learning rate 分別有 0.0001、0.0003、0.0005、0.001、0.0015、0.003、0.005。在第 200 個 epoch 時分別收斂到 loss 值 5.86、6.71、6.68、7.05、6.54、9.20、15.33。 Batch 的大小也略有琢磨，分別嘗試了 30、50、100、150、200、300、500。其中 30 太慢了沒等他跑完就停了，500 出現了 Resource Exhausted 問題，200、300 的收斂效果不是很好，在權衡速度與精準度的情況下，最後的模型選擇了 Batch size=100。 有關 S2VT LSTM 本身的大小，我們也做了一些嘗試: 測試了包括 128、256、512 以及 1024 四種大小，其中發現 512 及 1024 的表現差不多，都比 128 和 256 來得好，再更大的 Model 我們並沒有嘗試。

Fine-tuning:
儘管上述最後 Adam 的 learning rate 是使用 0.0001 且 Adam 已經是 adaptive 的 Gradient Descent Optimizer 了，我們仍發現 Loss 在一定的 epoch 之後會掉部下去，我們猜測是 Adam 的 Learning rate 變小的速度太慢了，所以我們在 Model 訓練完之後又另外將存好的 Model 用 Learning rate=0.000015 進行訓練，Loss 有繼續在往下掉，這樣不斷以 1/3 learning rate 做調整，人為的 fine tune，Loss 會很緩慢但持續的下降(final loss = 4.2 for S2VT(model-609))。

Testing Bleu Score (100 samples):

```
C:\Users\RUMI\Desktop\MLDS_hw2_1_data>python bleu_eval.py test_22.txt
Average bleu score is 0.6227056704642364
```

Sample Output

glrijRGnmc0_211_215.avi,a is a and on back along
HAjwXjwN9-A_16_24.avi,a using a is to makes
He7Ge7Sogrk_47_70.avi,a is a in glasses playing
HV12kTtdTT4_5_14.avi,a holding a is on bed
IhwPQL9dFYc_124_129.avi,a the is a in with up
inzk2fTUe1w_1_15.avi,a is a substance of
J---aiyznGQ_0_6.avi,a is a on white while colored

Comment:

不知道為什麼，我們產生的 output 大部分都跑不出主詞，都是 a+….，如此的結果 Bleu Score 是 0.62，如果將每個句子的第一個”a”後面都加入 man，Bleu Score 會上升許多，不過感覺這樣有點犯規，所以就沒這麼做來衝高 Bleu Score。 唯一有稍作修正的就是由於不是所有 Decode 最後都會輸出 EOS，所以對於長度特別長的句子很明顯會造成誤差，我們將長度大於 8 的句子都直接長度除 2，比如長度 18 的句子就取他的前 18/2=9 個字，並將跳針重複的字砍掉。

分工表

	王垣尹	許哲瑋	李中原
2-1	Model(S2VT) and Code And report	Technical Support	Model Fine Tuning and Report/ proper format
2-2	Report Support	Model(Seq2Seq) and Code And report	