ADLxMLDS HW2 報告

b03902083 陳力

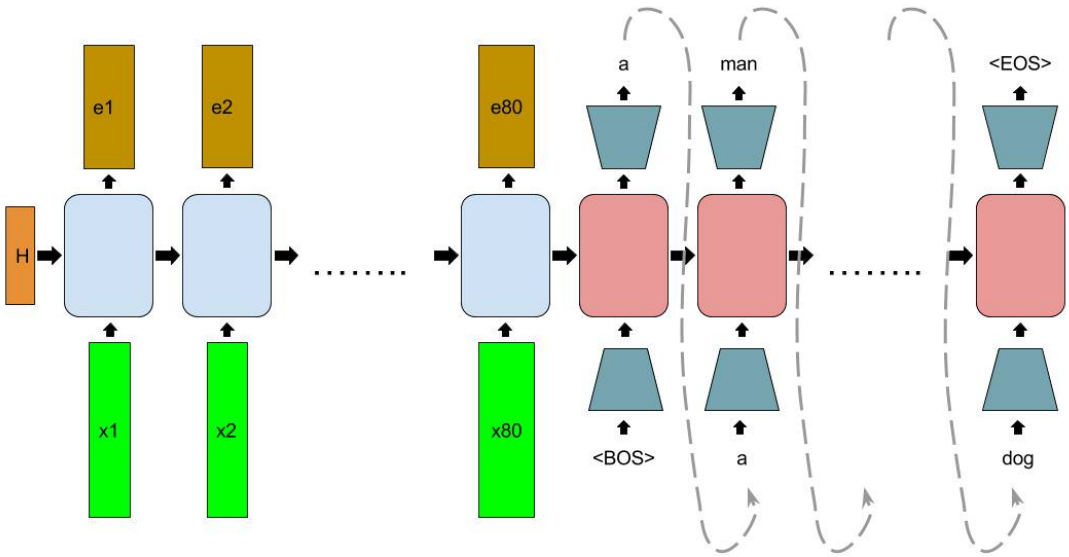
Model Description

輸入是一條長度為 80 的影片特徵序列 $\overrightarrow{x} = x_1, x_2, \dots, x_{80}, x_i \in \mathbb{R}^{4096}$，輸出則是一條長度為 50 的 one-hot encoding 的句子 $\overrightarrow{y} = y_1, y_2, \dots, y_{50}$，以 `<BOS>` 開頭 `<EOS>` 結尾，不足的部份補上 `<PAD>` ；我的模型由兩部分組成: encoder E 與 decoder D。

encoder 的是一個 LSTM unit，$E(\overrightarrow{x}) = (\overrightarrow{e}, h)$，$\overrightarrow{e}, h$ 分別為 LSTM 的輸出、隱藏狀態(包含控制狀態)。

decoder 的核心是一個 LSTM unit $R(\cdot)$，一開始的隱藏與控制狀態是直接取自 encoder 的輸出 h，每個 step 都會拿上一個 step 生成的單字與隱藏狀態生成下一次的單字，寫成數學式就是: $z_{t+1} = \phi^{\dagger}(R(\phi(z_t), h))$。因為句子都是 one-hot encoding 所做成，為了模型的合理性，在這我們會把單字做一次 word-embedding (ϕ, $\phi^{\dagger}$ 是將 embedding 過後的向量轉回 one-hot encoding space 的線性轉換)，再塞進去 LSTM。

整體來看，encoder 生成的隱藏與控制狀態像是句子的內涵，decoder 再把它轉換成一句話。



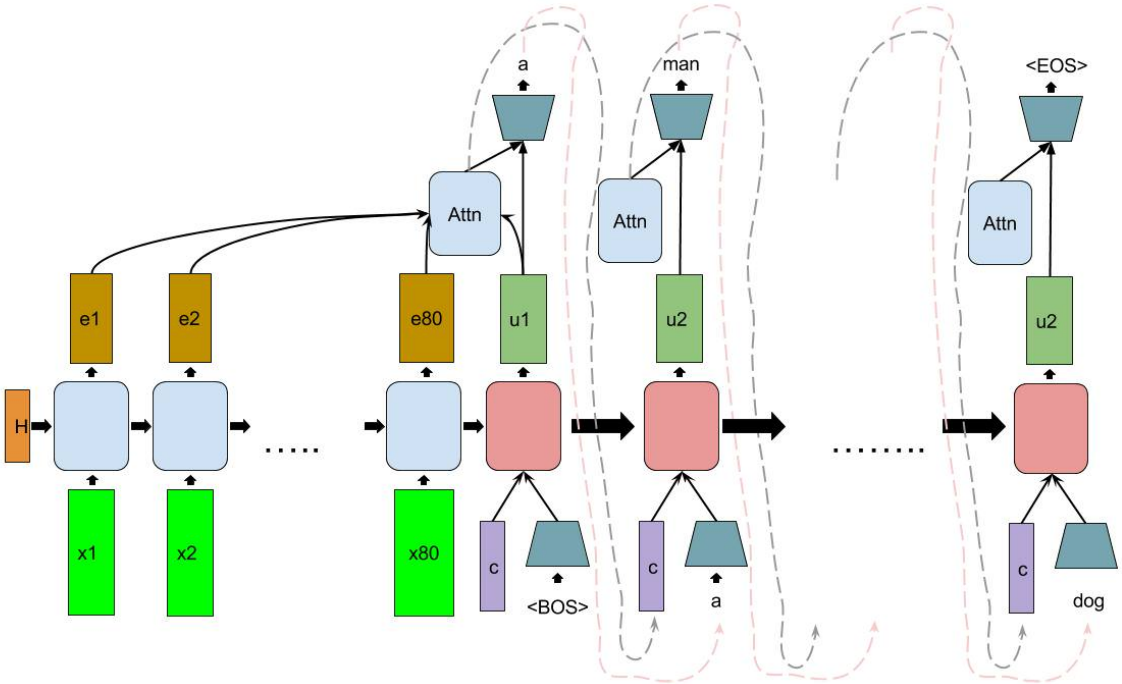
Attention Mechanism

decoder 每一個 step 在生字的時候，attention 機制要我們同時去考慮上下文關係，也就是去考慮 encoder 的輸出 $\overrightarrow{e}$。

仔細來說，每一個 step t 我都會有一個代表上下文的向量 c_t。下個 step 的輸出 z_{t+1} 不再只是 $\phi^{\dagger}(R(\phi(z_t), h_t))$ 這麼簡單。

而是 $z_{t+1} = \phi^{\dagger}([u_{t+1}; c_{t+1}])$，其中 $u_{t+1} = R([\phi(z_t); c_t], h_t)$, $c_{t+1} = \sum_{i=1}^{80} score(u_{t+1}, e_i)e_i$，$[a; b]$ 表示把兩個向量接起來。

從中可以看出，上下文向量 c_{t+1} 只是把 encoder 的輸出 $\overrightarrow{e}$ 做線性組合的結果；而 e_i 的係數則是 $score(u_{t+1}, e_i) = e_i^T H u_{t+1}$ 其中 H 是一個矩陣。



以下是比較有無 attention 機制的實驗結果，其中 hidden size 表示 LSTM 裡的 hidden unit 大小。每個模型都是使用 Adam optimizer，learning rate 0.0001，batch 大小皆為 16，訓練了 80 個 epoch。

Hidden size	BLEU w/ Attn	BLEU wo Attn
256	0.2698	0.2602
512	0.2864	0.2655

從表格可以發現，有 attention 的模型可以達到較好的 BLEU 分數。用肉眼觀察輸出可以發現，attention 模型的輸出會包含更為精準的動詞。

Performance Improvements

Schedule sampling:

每個 step 的輸出 z_{t+1} 會依賴於上一個 step 的輸出 z_t，但在 training 的時候，一開始這些輸出都會很爛，所以很難學到對的輸出方式，有一招就是直接拿正確答案當作 z_t 作為輸入去生成下個 step 的輸出。

Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks (https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43984.pdf) 提出了一種折衷方案，也就是有的 batch 是全部拿 y_t 當成 z_t，有的直接用 z_t，並用隨機選擇的方式去決定這次的 batch 用哪種方法。

而兩種方式出現的機率也會隨著 training 的過程有所變化，基本原則是 train 越久之後 model 會變得好一些，這時候就讓他比較有可能拿自己的輸出 z_t 當這個 step 的輸入。

我決定機率的方式是線性遞減，也就是第 k 個 epoch 會拿正確答案當 z_t 的機率是 $1 - \frac{k}{\text{total epoch}}$。

Loss computation:

這次的 data 有個特色是每個影片都有好幾個 (小於 40 個) 正確答案，在 training 的時候因為 schedule sampling 的關係，必須從中選擇一個作為參考依據。

每個 batch 在 training 算 loss 的時候，從正確答案中隨機選 5 個(包含參考依據)跟 model 的預測值一個字一個字算 cross entropy loss。

Vocabulary set:

儘管整個 corpus 有近 7000 種單字，為了降低模型複雜度，我只留下最常出現的 3000 種單字。

Sentence output:

把 model 所預測的句子轉換成輸出時，連續重複的單字會被縮成一個，遇到 <EOS> 就直接丟掉後面的單字。

Experimental Results

以下的實驗數據的共同參數為:

1. batch 大小 = 16
2. learning rate = 0.0001
3. train 300 個 epoch

有 schedule sampling

Hidden size	LSTM層數	embedding size	Attn?	BLEU
256	1	256	no	0.2598
256	1	256	yes	0.2673
512	1	256	no	0.2657
512	1	256	yes	0.2846
512	2	256	yes	0.2820
1024	1	256	yes	0.2730
1024	2	512	yes	0.2736

無 schedule sampling

Hidden size	LSTM層數	embedding size	Attn?	BLEU
256	1	256	yes	0.2736
1024	2	512	yes	0.2733

從中可以看出，有 attention 的 model 比沒有的好，而且 hidden size 越大越好，但大到 1024 就無法再分數上取得優勢。

而有無 schedule sampling 對結果並無顯著不同。

但從肉眼看起來，hidden size 1024 的那兩組輸出明顯的比較完整。

Reference

https://github.com/spro/practical-pytorch/blob/master/seq2seq-translation/seq2seq-translation.ipynb (https://github.com/spro/practical-pytorch/blob/master/seq2seq-translation/seq2seq-translation.ipynb)

https://arxiv.org/pdf/1508.04025.pdf (https://arxiv.org/pdf/1508.04025.pdf)

https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43984.pdf (https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43984.pdf)