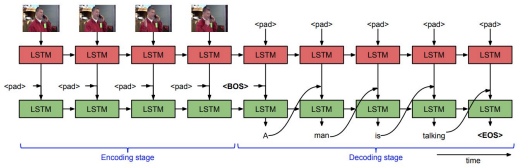
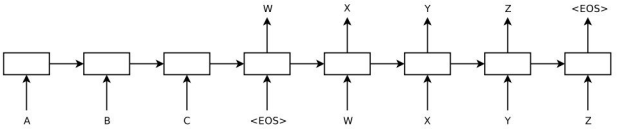
ADLxMLDS HW1 Report

B03902086 資工四 李鈺昇

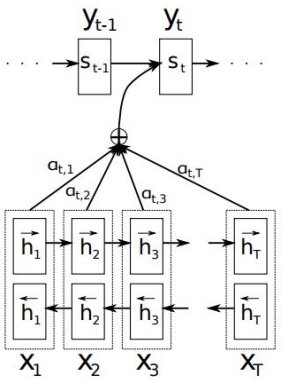
Model Description

我使用的 model 和投影片中介紹的 S2VT [1]（下左圖）很像；不過少了 encoder 的右半段和 decoder 的左半段，因此比較像是 [2]（下右圖）裡面的樣子。





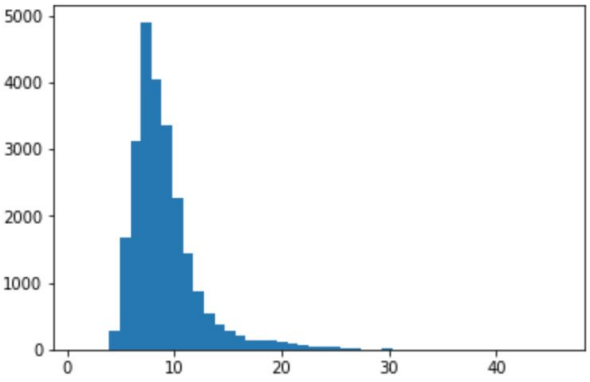
加上 attention 之後，會增加如 [3]（下圖）所描述的機制。詳細的實作請參考下一段。



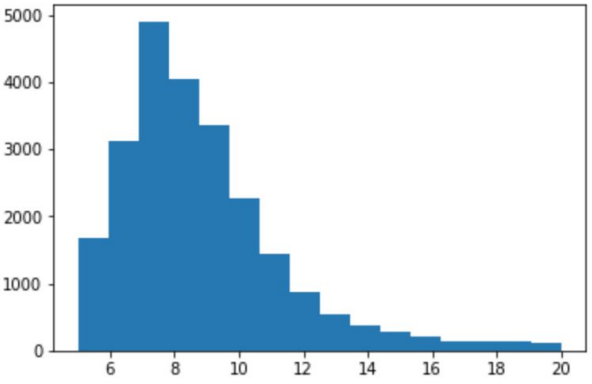
而我另外有做的一些特別處理如下：

- 太長或太短的句子可能都會成為影響 training 的結果的 noise，因此我只保留長度介於 5 ~ 20 之間（包含）的那些 caption 作為 training 時使用的 ground truth。
- 我認為出現太少次的單字也可以不用考慮，因此如果一個 caption 包含了某個出現次數少於兩次的單字，我就直接丟掉不考慮。由下圖可以發現丟掉之後分佈變得比較不極端，並且計算之後少掉的總單字數不到原來的 5%，因此影響應該不大。

丟掉之前



丟掉之後



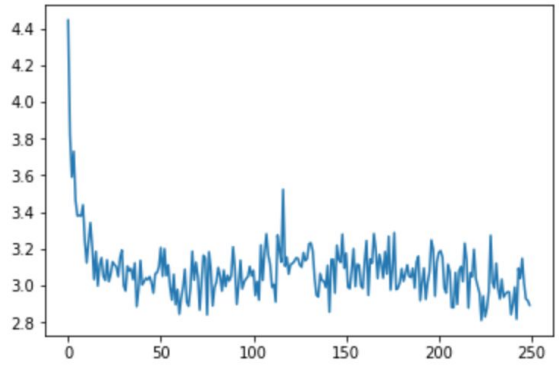
Attention Mechanism

為了實作 attention mechanism，我選擇比較靈活的 PyTorch 來實作。

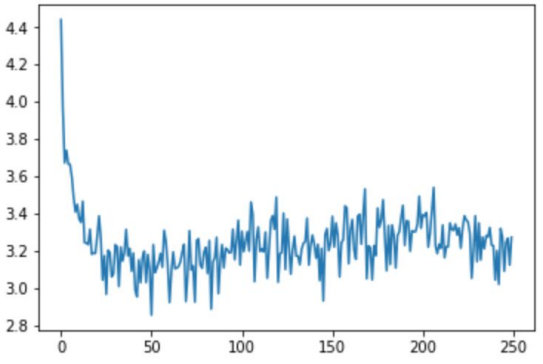
在 decoder RNN 的每一步，RNN cell 會吃進上一步的輸出、上一步的 context，並把輸出和這一步算出的 context 接在一起後經過一個 linear transformation 再找出最大的值的 index，作為這一步的輸出。
而重點就在於 context 的計算：這一步的 context 是經由 encoder 的輸出作為一個可以參考的基準向量 u，並與這一步 RNN 的輸出 v 做一些線性運算後得到一個機率分佈，就是 context。實務上有許多種常用的線性運算，我採用的是 u^TAv，其中 A 是要學的矩陣參數。

分別使用有與沒有 attention 的模型後，可以觀察到如下的差異。

沒有 attention



有 attention



很奇怪的是兩者差不多，甚至沒有 attention 的版本算出來的 BLEU 還更高（見下段表格）！但這跟預期的結果不太一樣：我的期待是有 attention 的話應該會學得比較快、比較好，不過看起來兩者在這個 dataset 以及我使用的參數下沒有這樣的現象。

Improving the Performance

最原始的設定是

(1) RNN 採用 256-hidden-dimension GRU，dropout = 0.05，SGD 跑 100000 epochs
(2) 使用固定的 scheduled sampling rate = 0.5（隨機決定）
(3) 直接把 decoder 輸出的結果原封不動作為答案。

這樣的結果用第一種 BLEU 算法就有 0.2524，直接過了 baseline！
後來我有分別嘗試

(1') RNN 把 256 改成 512、但是跑的 epochs 數量減半、
(2') 使用隨著 epoch 線性變化的 scheduled sampling rate（一開始都用 ground truth，線性變化到最後都用 decoder 上一步的輸出）、
(3') 對 decoder 輸出的結果做一些處理之後才作為答案，

以下表格列出一些比較顯著的情況。

	BLEU 算法一	BLEU 算法二	過 baseline
原始設定	0.2524	0.5198	算法一
(2')	0.2429	0.4760	X
(3')	0.2640	0.6158	算法一
(2') + (3')	0.2558	0.5936	算法一
(1') + (2') + (3')	0.2618	0.6130	算法一
(1'), no attention	**0.2683**	**0.6719**	算法一、算法二

其中 (1') 的表現沒有變好，但是我想是因為 epochs 少了一半的關係；如果兩者 epochs 一樣多的話，在沒有嚴重 overfitting 的情況下我認為應該是會變好的。

而 (2') 的表現也沒有變好，這點我認為還滿值得研究，因為理想上應該是線性變化可以讓它學得比隨機還要更好，但我這次並沒有找到原因。

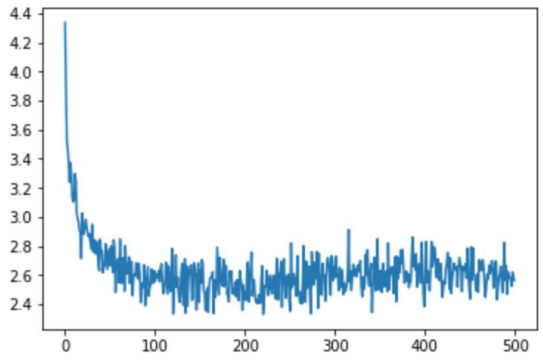
最後，(3') 的「處理」是指在不影響大致結果的情況下，把結尾的一些東西刪掉。我有分別試過把所有結尾的 "a" 和 "." （句點）都拿掉；以及拿掉句點後再把結尾的 "a" 刪到最多只剩一個。會想這麼做是因為經過觀察發現這兩者最長出現在句尾並重複很多次，而第二種做法的主因是 "a" 的情況比較特別，例如 "a woman is eating a a . . ." 這種句型，雖然最後的 "a" 乍看之下可以全部刪除，但其實動詞 "eating" 後面有極高的機率會接上 定冠詞+名詞 的組合，因此保留一個 "a" 反而是合理的。
由於第二種做法的分數都比第一種高，表格的 (3') 都是指第二種做法。

Experiments and Settings

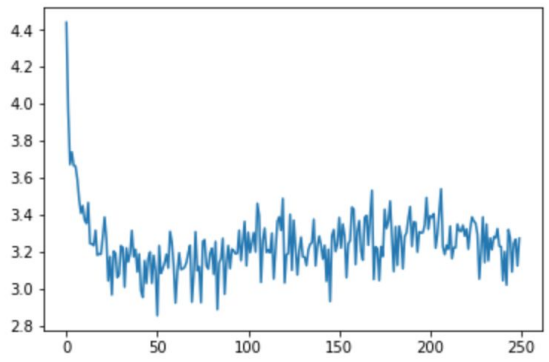
實驗一：hidden dimension

一開始我是根據投影片上給的 256 作為 hidden layer 的大小，不過後來發現 BLEU 沒有想像中的好，因此還是想要試試看調大會發生什麼事情。以下是兩者的 training loss 的圖表：

256



512

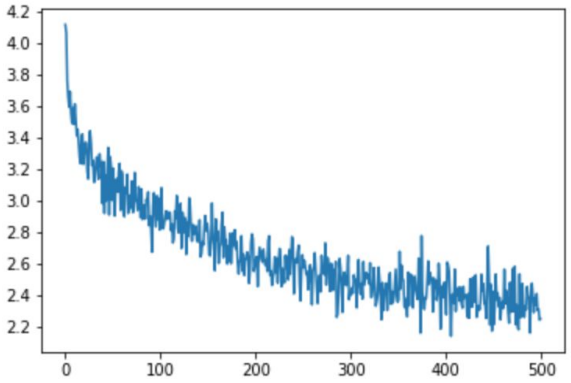


可以發現將 hidden dimension 增大兩倍後 loss 變得比較不穩定，值也都比較高（但這可能是因為 model 本身變得複雜），我認為是因為 training data 的樣本數只有一千多，不足以學好這麼多的參數，因此我認為 256 基本上應該就足夠了。

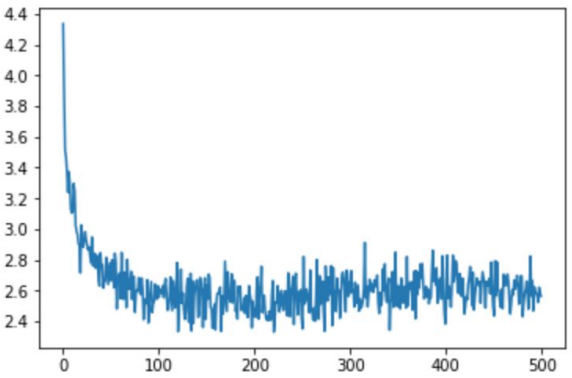
實驗二：scheduled sampling rate

我一開始為了減少變因，只採用固定為 0.5 的 scheduled sampling rate（隨機決定），但是後來覺得使用線性的應該比較好（見上一段），因此對於兩者的 training loss 畫出了如下的圖表（其他參數固定不變）：

0.5



線性



可以觀察出，線性的 loss 比 0.5 降得快很多（雖然在這組參數下最後並沒有收斂到夠低的 loss），而這也是符合預期的，畢竟一開始什麼都還沒學到的時候儘量給 ground truth 會比較穩定，而在最終階段讓 decoder 使用自己前一步的輸出可以稍微避免 overfitting。

Packages and Versions

使用 pip3 freeze > requirements.txt 之後，找出比較重要的幾個如下：

arrow==0.10.0
numpy==1.13.3
torch==0.2.0.post3

References

[1] Sequence to Sequence – Video to Text
[2] Sequence to Sequence Learning with Neural Networks
[3] Neural Machine Translation by Jointly Learning to Align and Translate