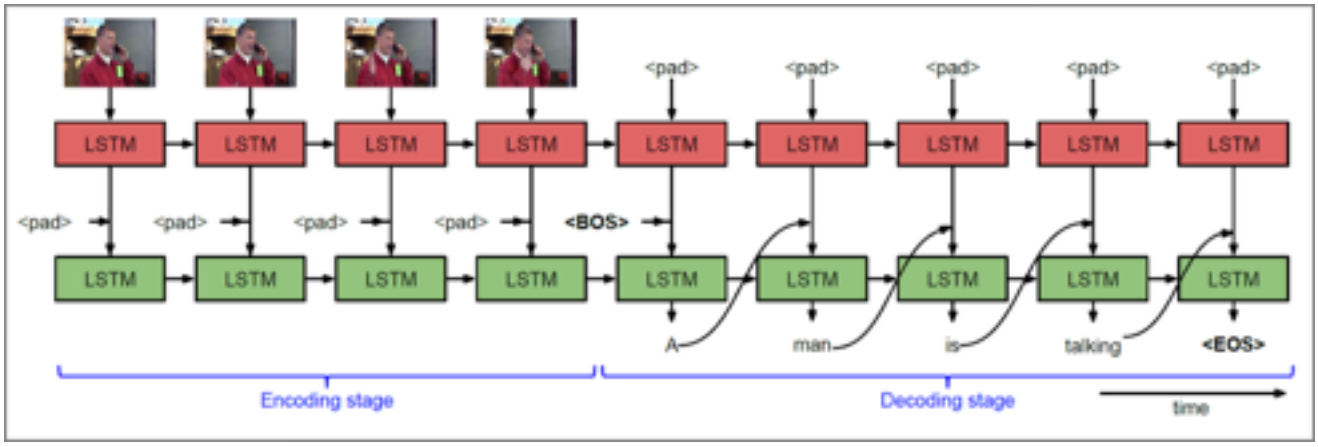
HW2 Report

資管碩一 R06725053 韓宏光

1. Model description

此model為使用兩個LSTM所串聯組成的，分為兩次輸入：第一次為輸入影像的feature及字幕的padding，第二次為輸入影像的padding及字幕的one hot vector，兩個LSTM的hidden layer size皆為256， one hot vector size為3746(過濾出現次數少於2次的字)，learning rate為0.001，而字幕的選擇為隨機選擇(每個影片有多個字幕)。

架構如下圖所示：



2. Attention mechanism

(1) How do you implement attention mechanism

我的實作將其分為四個步驟：

A. 將第一層LSTM(輸入為video feature)的hidden state分別取出並合併為context_vector

```
expand_encode_state2 = tf.expand_dims(state2, 2)

if(i == 0):
  context_vector = expand_encode_state2
else:
  context_vector = tf.concat([context_vector, expand_encode_state2], 2)
```

B. 第二層LSTM裡每一個hidden state與context vector分別進行element wise的相乘，並進行加總，最後會得到一個加總後的vector

```
expand_decode_state2 = tf.expand_dims(state2, 2)
scores = tf.reduce_sum(tf.multiply(context_vector, expand_decode_state2), 1, keep_dims=True)
```

C. 將B步驟輸出的vector取softmax，得到一個總和為1的align vector

```
a_t = tf.nn.softmax(scores)
```

D. align vector與context vector相乘後再進行線性運算，最後套上tanh函式輸出一個新的state

```
c_t  = tf.matmul(context_vector, tf.transpose(a_t, perm=[0,2,1]))
state2_tld = tf.tanh(tf.matmul(tf.concat([state2, c_t], 1), self.W_c) + self.b_c)
```

E. 將D步驟輸出的state附著上第二層LSTM的每一個state，即可得到一個global attention based的模型

```
output2, state2 = self.lstm2(tf.concat([current_embed, output1], 1), state2_tld)
```

(2) Compare and analyze the results of models with and without attention mechanism

在相同參數及訓練epochs的情況下，Attention的表現”稍”優於沒有加上Attention的模型

A. 沒有加attention的模型

```
Originally, average bleu score is 0.291615326069919 25
By another method, average bleu score is 0.6543631844829246
```

B. 有加attention的模型

```
~/ADLxMLDS2017/hw2/MLDS_hw2_data  master ●  python3 bleu_eval.py '../S2VT_results.txt'
Originally, average bleu score is 0.29333806002594676
By another method, average bleu score is 0.6471091259276275
```

由於我是隨機挑一個而非挑選特定的caption出來訓練，因此在舊有的bleu評量方式(平均)會比較適合評量模型的performance，因此在average bleu score上，添加attention的模型在訓練上相較於沒加的模型，在score上增加了0.017。

3. How to improve your performance

(1) Describe the model or technique

A. 將one-vector改為使用word-embedding
為一個3742*256 dimension的matrix。
B. 使用了schedule sampling
雖然叫schedule sampling，但其實我沒有用到論文裡面提到的變動式sampling(一開始比較大的機會將正確的答案輸入，之後隨著訓練次數的遞增，增加輸入自己上一個state輸出的字符的機率；透過彈性調整機率的方式來增加收斂的速度)，我先是訓練了100個epoch，之後才加上sampling(3成輸入自己上一個輸出的字符、7成輸入正確答案)接著訓練了100個epochs。

(3) Why do you use it

A. 將one-vector改為使用word-embedding
因為我認為加了word-embedding並不會拖累多少訓練速度(線性)，而word-embedding著其特性(字跟字之間有著關聯)將會幫助模型訓練上的精準度。
如果時間充裕的話，我會把Google在2013年釋出的訓練幾百萬篇的word embedding資料拿來當作base init vector進行訓練，感覺會很猛。
B. 使用了schedule sampling
我沒完全使用”schedule” sampling的原因是，我覺得與其等模型自己調整抽樣機率(而且裡面的關鍵參數K還要我自己慢慢試)，還不如先train個100個epochs直接讓模型有了一定的強度後，再後面100個epochs才加上3比7的sampling。這加強了模型根據上一個輸出的字符來預測下一個字符的能力。

4. Experimental result and settings

因為單用bleu很難去評斷模型的好壞，因此我都大概抽樣10~15個輸出結果去檢視是否與影片一致。
最後發現訓練180個epochs的模型表現是最好的
而因為好奇因素，我訓練了一個經過9000 epochs的模型，耗時約2天。
這個模型預測出來的結果有些句法很厲害：

```
inzk2fTUe1w_1_15.avi,a man is showing a yellow piece of paper with a paper towel
```

但是實際的影片可能只有一雙手在剝香蕉，而後面有一個毛巾。