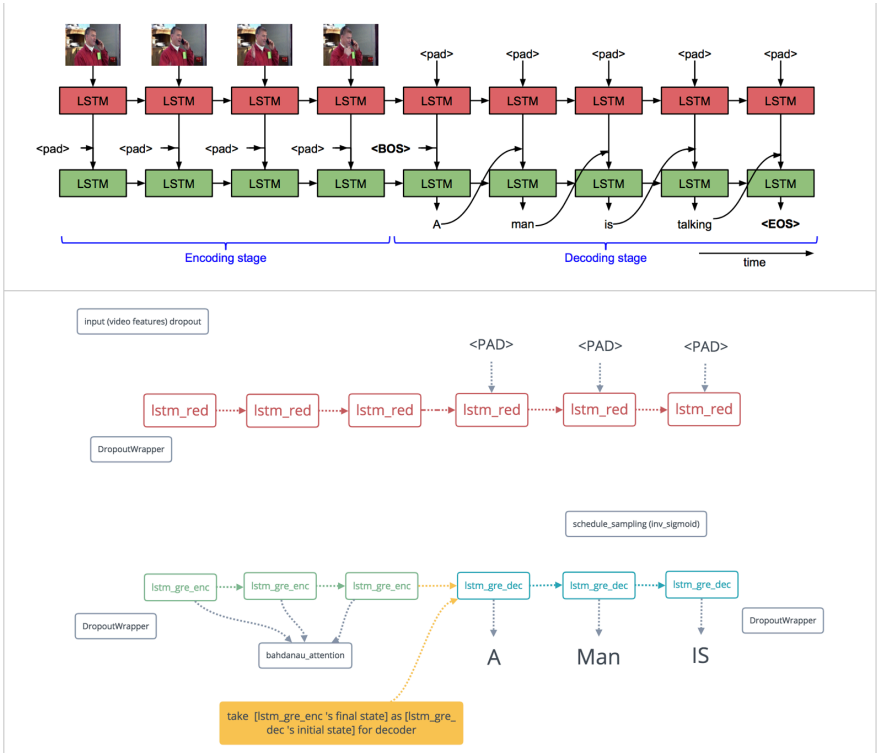
Video Captioning - MLDS hw2-1

分工表：蔡丞昊 b03901028: hw2-1; 許秉鈞 b03901023: hw2-1; 楊正彥 b03901086: hw2-2

Model description (3%)

Describe your seq2seq model (3%)



上圖為 paper 的原圖，下圖是我們的 model 示意圖。為了避免混淆（例如：與原 paper 重疊的線段以及 concat 操作...等）這張圖就不重述，只寫上我們自己設計的部分。

我們創建兩個 lstm，分別為 `lstm_red` 以及 `lstm_gre`，左半部是 encoding state、等到 80 張 frames 都讀進後、就進到 decodeing state、餵入 `<PAD>`。上半 for loop 是屬於 encoding stage、下半 loop 則是decoding stage。

```
for i in range(0, n_frames): # encoding stage
  with tf.variable_scope("LSTM1"):
    output_red, state_red = lstm_red(image_emb[i,:,:], state_red)

  with tf.variable_scope("LSTM2"):
    output_gre, state_gre = lstm_gre(tf.concat([padding, output_red], axis=1), state_gre)
    h_src.append(output_gre) # 待會 attention 會用到，見上圖虛線線段

#####

for i in range(0, max_caption_len): # decoding stage

  with tf.variable_scope("LSTM1"): # lstm_red 吃的是 <PAD>
    output_red, state_red = lstm_red(padding_in, state_red)
  if i == 0:
    with tf.variable_scope("LSTM2"):  # 第一次時 concat 進去的是 <BOS>
      con = tf.concat([bos, output_red], axis=1)
      output_gre, state_gre = lstm_gre(con, state_gre)
  else:
    if phase == phases['train']:
      if sampling[i] == True: # schedule sampling, 選擇吃 ground truth  還是自己的 output
        feed_in = captions[:, i - 1]
      else:
        feed_in = tf.argmax(logit_words, 1)
    else: # testing phase 則一定是吃自己 output
      feed_in = tf.argmax(logit_words, 1)
      with tf.device("/cpu:0"): # word embedding
        embed_result = tf.nn.embedding_lookup(embeddings['emb'], feed_in)
```

```
        with tf.variable_scope("LSTM2"):
          con = tf.concat([embed_result, output_red], axis=1)
          output_gre, state_gre = lstm_gre(con, state_gre)
```

在做完 RNN 的操作後，我們就用 `logit_words = tf.matmul(output_gre, weights['W_dec']) + biases['b_dec']` 把 decoder 的 output 乘進去之後，就能拿到 2891 dim 的單字 output ，最後再用 cross entropy 算每一句總和的 sequence loss ，比較重要的是要除以該句長度、而且要蓋上 sequence mask。

除了 for loop 單元操作的版本之外，我們也有實作 `dynamic_rnn()` 以及 `raw_rnn()` 版本的 seq2seq，再搭配 tf 提供的 `bidirectional_dynamic_rnn()` 取出 forward state 以及 backward state，再將兩者加總，但最後結果和原本差不多。如果有興趣請換到 adrian-hw2 branch 的 commit 以查看。

commit: 5a7c74d61edc6012353f930f018640f91d131069

BLEU Score : `0.72434`

在此參數設定下（bahdanau attention + schedule sampling） ，沒有任何 rule-based 的微調，我們達到了 **BLEU =** `0.72434` 的分數，通過 baseline = `0.6` 。

```
(python3) adrianhsu:~/Desktop/S2VT-seq2seq-video-captioning-attention (master)
$ python3 bleu_eval.py output.txt
By another method, average bleu score is 0.7243460350812861
```

關於每個 epoch 對應的 BLEU score，**請參照 report 後段的實驗**；簡單來說我們發現 train 得越久、雖然 schesule sampling 的擲硬幣擲出的機率越來越低，但當時 model 已經學得夠好了， training loss 仍然能維持很低、而且有自己的 output 更能模擬 testing 的實際狀況，所以 train 得蠻好的。

Experiment Setup

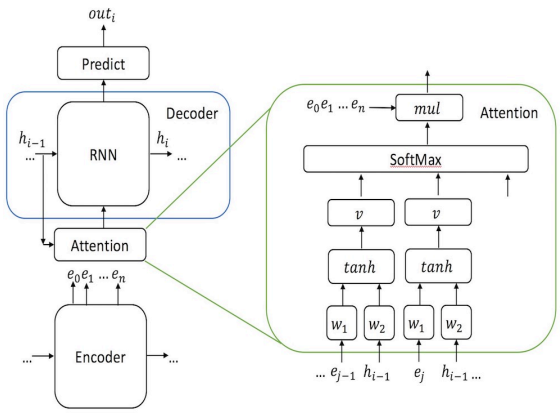
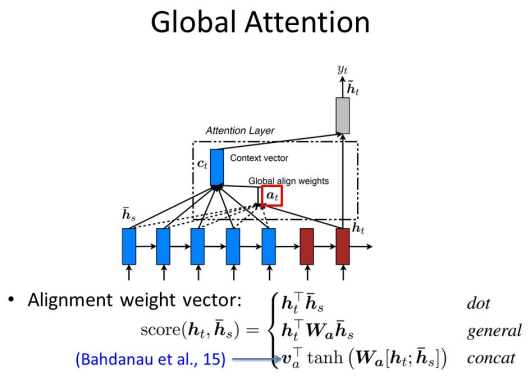
Global Parameters:

```
n_inputs        = 4096
n_hidden        = 600
val_batch_size  = 100 # validation batch size
n_frames        = 80 # .npy (80, 4096)
max_caption_len = 50
forget_bias_red = 1.0
forget_bias_gre = 1.0
dropout_prob    = 0.5
learning_rate   = 1e-4
num_epochs      = 100
batch_size      = 250
```

How to improve your performance (3%)

Write down the method that makes you outstanding (1%)

& Why do you use it (1%)

Bahdanau Attention	Luong Attention
	

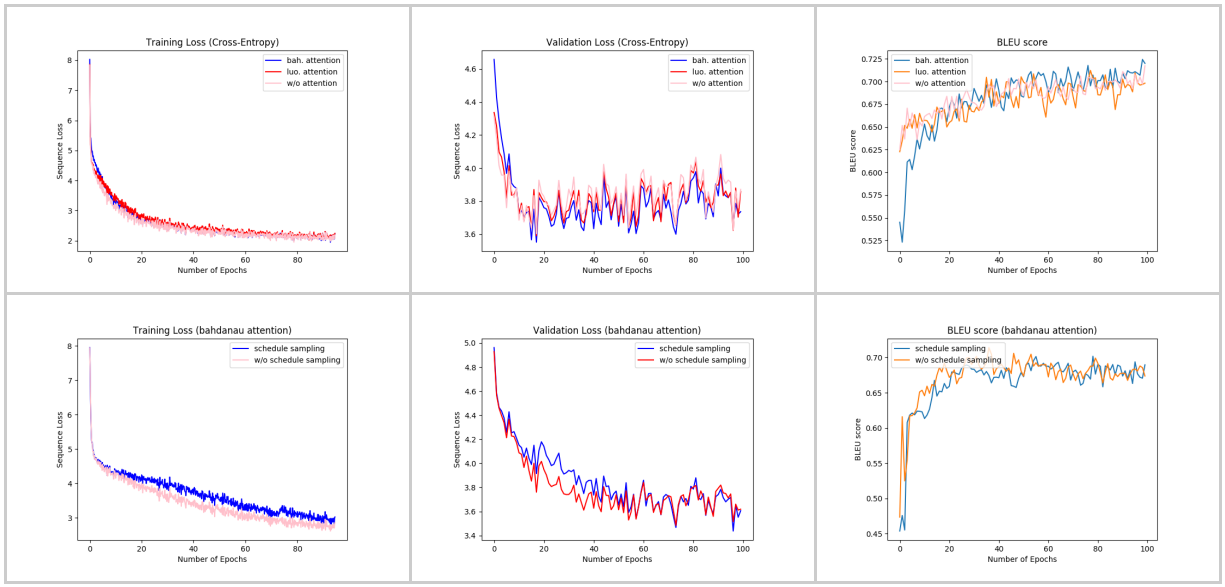
1. **我們實作了 Bahdanau Attention，按照公式：** $score(\mathbf{h_t}, \mathbf{h_s}) = \mathbf{v}_\alpha{}^T tanh[\mathbf{W}_\alpha; \mathbf{h_s}]$，其中 W 拆成兩個，一個 W1 和 encoded

input 的每個 output vector 相乘、另一個 W2 則和 decoder 的 hidden state（取出 state tuple 的 `(state_c, state_h)` 的後者）相乘，過一個 `tanh()` 之後再乘上 V。另外，我們把最後的 encoder output 當作 z_0 。這三個 W1, W2, V 都是 trainable parameters。

2. **我們也實作 Luong Attention，我選的是 General 的 Mutiplicative 版本，按照公式**：$score(\mathbf{h_t}, \mathbf{h_s}) = \mathbf{h_t}^T \mathbf{W}_\alpha \mathbf{h_s}$，雖然可以設定 window 做成 Local Attention，但我們選用 Global 的實作才能和前者 Global 比較。這版本只有 W 是 trainable 的，參數量較少，但我們猜測 model capacity 也較小。
3. **我們使用了 schedule sampling** ，並比較了 inverse-sigmoid 和 linear decay 的效果，發現前者較好。我們的 inverse-sigmoid 是從 0.88 取到 0.12，而不是常見的 1 取到 0 ，以對抗 overfit, underfit 的發生可能性。Schedule sampling 我是**讓 batch 裡面的每一句都同時是 T 或同時是 F**，其中 T 代表取 ground truth、而 F 則是取 decoder 自己的 output。
4. **encoder 階段的 input frame、以及 decoder 階段的 output words 傳進去作為 input 時，都應該要先做 embedding。**前者是我們實驗後發現先過一層 fc 再傳入 500 dim 會比直接傳 4096 dim 的結果好；後者則是一定要做 embedding，否則維度不對。
5. **decoder 階段的 output words 有三種做法做 embedding**：(1) 用 GloVe 的 pretrained vector (2) 用 end-to-end train 的 embedding (3) 用 one-hot，也就是 `(2891, 2891)` 維度的矩陣。後來發現 (2) 的結果就很好，而且原本 paper 也是用這個方法，於是採用 **(2)**。
6. **使用 Schedule Sampling 採用 inverse sigmoid， 0.88 下降到 0.12。**我們發現從 1.0 下降到 0.0 的效果並不夠好，因為一開始的 teacher learning 太容易、而最後逼近 0 的 learning 又太難、很容易錯，導致中間 train 得很好、但到最後爛掉的結果。
7. **在 training 階段把 input frames 個別加上 random noise**。因為每部 videos 對應到很多句 captions，可能會讓 model 混淆、不知道這個 input 應該生成哪個 caption 才正確。
8. **以及 不需要做 embedding，而應該直接 concat 。**
9. 設定 `min counts = 3` ，然後存成 `.pkl` ，把出現太少次的字當成 `<UNK>` 以簡化。
10. dataset 有兩種取法，一種是從 training 的 1450 部影片中，每部隨機取一個 caption，這樣一個 epoch 的大小是 1450；另一種則是**把每部影片對應到 caption 都當成一筆 data，這樣共有 24232 筆**、資料量比較多，而且也能確保每個 caption 都被採用過，因此我們使用的是後者。

Link Link

Analysis and compare your model without the method. (1%)



上半部 fig. 1~3

我們比較了 without attention, bahdanau attention, luong attention 這三者的效果。可以發現 training loss 大家差不多、 validation loss 則是 w/o attention 的表現最差（粉紅色）、而 bah. attention 的 loss （藍色）則是一直保持最低，最後 BLEU score 可以發現 bah. attention 在剛開始時很爛、應該是因為 W1, W2, V 這三個 trainable parameters 都才剛創建，數字還很差；最後的結果，比較奇怪的是 bah. 雖然是最好的，但 luo. attention 卻是比 w/o 的差一些，我猜測可能是只有 train 100 epochs 還不太夠，或是我的 attention 實作有些地方有 bug 或需要優化。

下半部 fig. 4~6

我們比較了在 bah. attention 給定之下（`batch_size=250, lr=1e-4`） ， with schedule sampling v.s. w/o schedule sampling 的結果。從結果來看，training loss 因為有 schedule samp. 的關係所以藍色線表現較差，符合猜想；至於 validation loss 也很符合預期結果：一開始 schedule samp. 表現較差，但是 train 越久之後兩者表現差不多，然後再接近 100 epochs 時超越 w/o 的表現，因為已經不需要 teacher learning、可以自主學習；最後則是 BLEU Score，這部分比較看不出好壞，但大致上也是後期 w/o 的表現較差一些（橘色線）。

Experimental results and settings (1%)

Batch size, Model Capacity & Overfitting

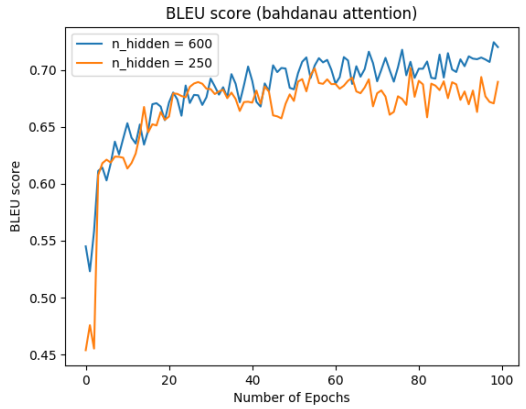


fig 7.

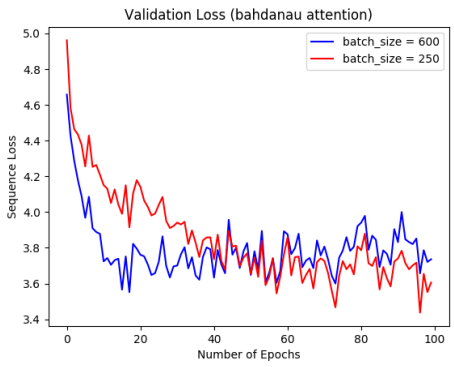


fig 8.

fig 9.

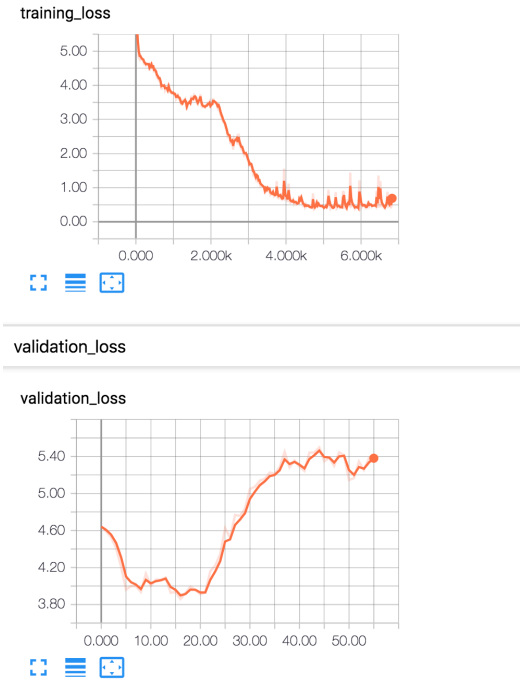


fig 10.

上面的 fig 7. 和 fig 8. 是比較 `n_hidden` 的影響。我的 n_hidden 就是兩個 LSTM cell 的 hidden state 數量，理論上來說這個值越大、model capacity 越大、越能學得好。實驗結果發現 BLEU 確實變得很好，但奇怪的是 Validation Loss 在前半段是 `n_hidden=600` 的領先、但到後面卻被 `n_hidden=250` 的超越了，我猜測可能是 overfitting，因為學得太好、所以無法應對 validation 的沒看過的 data。另一個原因可能是我的 **Validation Loss 計算方式**，我覺得這樣的計算方式有些問題：拿 testing set 任一句 caption 和結果對答案，而不是 1-to-1 的關係，而且每輪可能不同，這是我們可以改進的地方。

下面的 fig 9. 以及 fig 10. 則是兩個極端的例子：我想測試 model capacity 很大時會不會 overfitting。 fig 9. 是 `lr=1e-4, batch_size=60, dropout=0.5, n_hidden=1000, w/o schedule sampling` ， fig.10 則是 `lr=1e-3, batch_size=100, dropout=1.0, n_hidden=1000, w/o schedule sampling` ，可以明顯發現兩者登在中後段之後、 training loss 降到接近 0 、與此同時 validation loss 開始急速上升、明顯的 overfitting 。後來把 n_hidden 降到 600 就好很多。

Appendix

Correct descriptions

`TZ86OP4iTaM_15_28.avi`	**a cat is playing the piano**
`Olh_UWF9ZP4_62_69.avi`	**a woman is mixing ingredients**

容易辨認的「貓」、「鋼琴」，或是「女性」甚至是 mixing ingredients 這樣的動作，都能夠順利的辨認。而且只要在 epoch = 50 左右就能達到。

Relevant but incorrect descriptions

`778mkceEOUQ_40_46.avi`	**a car is driving a a car**
`PeUHyOAlGFO_114_121.avi`	**a woman is the shrimp**
`ufFT2BWh3BQ_0_8.avi`	**a panda panda is**

在這三個例子內，第一個從頭到尾只有空中拍到車、因此被誤認為 car is driving a car；第二個則是只拍到蝦子、還有剝蝦子的手指、並沒有拍到人物；第三個則是兩隻熊貓疊在一起，結果印出兩次 panda 而不是複數名詞。還有很多沒列出的 `a man is a a` 這樣的句型，讀來不知所云。我猜測是因為資料不夠造成學得不夠好，training set 沒有教到這些情況應該如何應對，所以才會出錯。如果再加上 beam search 或是更多的 model 優化技巧相信能做到更好。

Implementation Details & Difficulties

1. **計算 sequence loss 時，要把後面的 padding 都用 cap_len 蓋掉，才不會多算到。**
2. **要把 caption 都補齊到相同長度，才能餵進去 model**，並且記錄 cap_len，因為算 cross-entropy loss 需要。
3. 一開始我以為 LSTM cell 的 num_units 要和 input dim 相等，後來發現不用。但是最後的 output dim 會等於 num_units。這個 num_units 真正的意思是有幾個 hidden-state，也就是 `n_hidden`，可以想成是經過這麼多的時間段。 Link
4. 不能每個 epoch 都 random 亂取（會取到重複的）而是要一個 epoch 給一個 permutation 順序，然後照著這個取，否則會發生 birthday paradox：發生 collision 的機率比認知的高很多。
5. Gate 的初始化以及不僅僅影響收斂速度，還影響最後的 testing。我們使用 `orthogonal_initializer()`，然而、效果和沒有特別指定 initializer 差不多。
6. （此段 和 performance 沒有關係），我們使用兩個 graph 分別存 train, test model ， `sess.run()` 的對象就跟根據 graph 實作，TF nmt tutorial 提出這樣的設計模式會有助於機器學習模型的維護。