ADLXMLDS Homework 2 Video Captioning

B03705013 資管四 徐逸然

- ## Model description (2%)

我的模型分成兩個 input，第一個 video_model_input 為助教提供經過 VGG19 取出 feature 後的影片 ，shape 為 sample * timestep * 4096。第二部分 caption_model_input 為輸入第 0 至第 t-1 timestep 的 caption。舉例來說句字若是[<bos>, hi, <eos>]，則 input [<bos>, hi]。video_model_input 經過 LSTM 會到一個 state，將此 state 作為 decoder 的 initial_state，並把 decoder output 與我自己嘗試做出來的 Attention concat 起來最後通過 Dense 來預測結果。模型最後的 output 為第 1 至第 t 個 timestep 之 caption，即[hi, <eos>]。

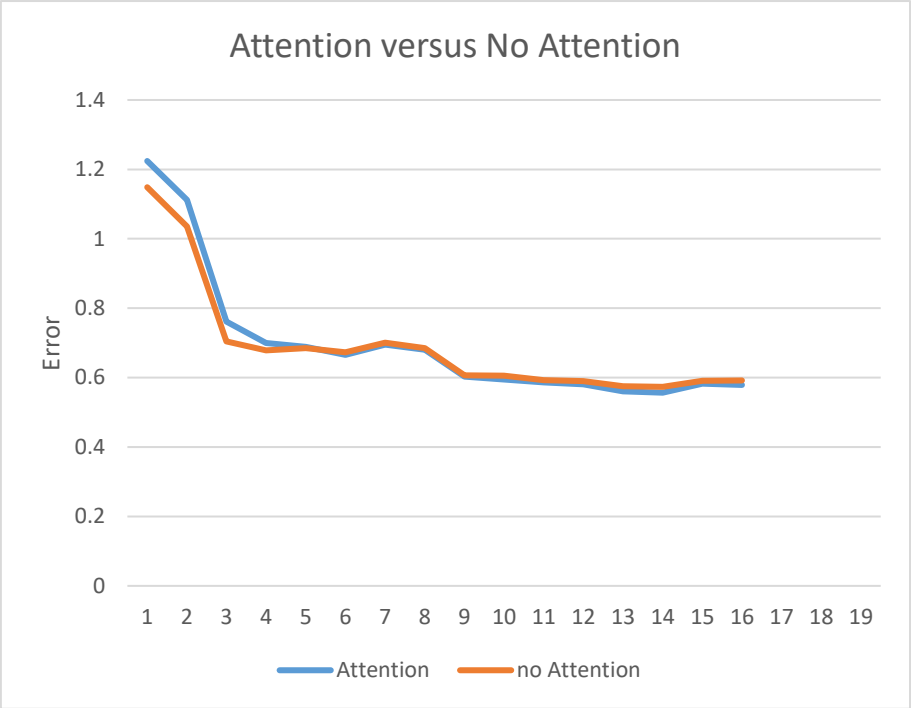
```
video_model_input = Input(shape=(80, 4096))
video_model, h1,h2 = LSTM(4096, return_sequences=True, return_state=True, activation="relu")(video_model_input)

caption_model_input = Input(shape=(None,))
caption_embed = Embedding(num_decoder_tokens, 300)(caption_model_input)
decoder = LSTM(4096,return_sequences=True, stateful=False, activation="relu")(caption_embed, initial_state=[h1,h2])

score = dot([decoder,video_model],2)
score = Activation('softmax')(score)
score = Dropout(0.35)(score)
permute = Permute((2,1))(video_model_input)
attention = dot([score,permute],2)

final = concatenate([attention, decoder])
final = Activation('tanh')(final)
final = Dropout(0.35)(final)
final = Dense(num_decoder_tokens, activation='softmax')(final)

model = Model([video_model_input,caption_model_input],final)
model.compile(optimizer=Adam(lr=0.0001, clipnorm=1.), loss='categorical_crossentropy')
```

而在預測是我採用以下做法。將影片 feature 與一個 caption [<bos>]讀入。此時會預測出一個新的句子(假設為 a)。下一次 predict 即丟進[<bos>, a]。此時會預測出兩個字(也就是下一個 timestep 應該是甚麼句子，假設為 a man)。把最後的字(man)當加入剛剛的 input ➔ [<bos>, a, man]，接著再將此 input 丟進去 predict。不斷重複以上步驟直到 eos 被預測出來，或是句子長度超過 42(training data 最大句子字數)為止。

- ## Attention mechanism(2%)

我的 Attention 機制是參照這篇論文 http://www.aclweb.org/anthology/D15-1166 而來。首先我先把 decoder 輸出與 encoder 輸出做 dot，並且 activation 設為 softmax。此時稱作 alignment。接著將讀進來的影片 feature 做 permute 後與 alignment 做 dot。此時即得到 context vector(shape 為 None, None, 4096)。最後再把 context vector 與 decoder 輸出 concatenate 並接上模型的 output。

然而做出來的結果並沒有想像中的理想。從 Validation Error 來看，有沒有加 Attention 幾乎完全無差別，Attention 最後僅比普通 encoder decoder 模型好上百分之三左右。我的猜想是我沒有將有將上一個 timestep 的 alignment 納入這一次 alignmnet 的計算中。然而若想用 keras 時做的話必須手動刻出一個新的 layer 才有辦法。這次由於卡在期中考故無時間做詳細的研究。



- # How to improve your performance (1%)

1. Word Embedding:

Word embedding 相較 one-hot encoding 有幾個好處:

1. One-hot 會變成很大的稀疏矩陣，training set 的字數大概有 6000 多字，因此 One-hot 出來會有 6000 多個 feature，訓練時間很長。
2. Word embedding 後相同語意的字或是在句子中常出現在類似位置的字會算出比較相近的 vector，也就是考慮了字詞先後順序的關係。One hot 則沒辦法，兩個字的 vector 之 cosine-similarity 永遠是零，即便他們是很類似的字(如 man 與 woman)

2. Stateful RNN:

這次的訓練影片只有 1450 個，而當中其實有許多影片都描述些許類似的情景，因此每個 batch 之間我認為 state 可以保留，而不是重新 initial state 之後再訓練。調成 stateful 之後收斂速度以及精準度都有些許提升。

- # Experimental results and settings (1%)

由於這次作業跟期中考撞期，我被迫提早一周寫完作業去念其他科的考試，所以沒有餘裕做很多實驗，請助教見諒，以下針對兩個參數做調整並說明結果。

第一個測試為不同 embedding size 之調整。可以看見兩條 training curve 非常類似，然而增加 embedding size 可以使整體 error 下降。這邊用的 embedding 是 Keras 內建的 embedding，若能引用其他的 embedding 方法如 word2vec，應能達到更好的效果。

	New Bleu Score	Origin Bleu Score
Embedding size 50	0.2817	0.299083
Embedding size 300	0.682992	0.693190563

第二個測試為比較 LSTM 與 GRU 之效果，由於 GRU 將 forget gate 和 input gate 合再一起，因此參數量比起 LSTM 的少了許多。以往在訓練 RNN 模型時，包況上學期 ML 作業以及上一次 HW1 等等，都是 GRU 的結果較好，訓練時間也較短。然而參考了幾篇關於 Attention 機制的論文，很多都是 LSTM 勝出。因此我測試 encoder 和 decoder 分別都用 LSTM 或 GRU。結果入下圖，LSTM 的結果稍微好一些，error 整體較 GRU 下降百分之四到百分之五左右。然而 LSTM 訓練時間約略是 GRU 1.5 倍，若沒有足夠運算資源則使用 GRU 即可。