Report

R04922108、R04922098、R04922086、R04922065 Master Best Bleu: 0.289

Environment

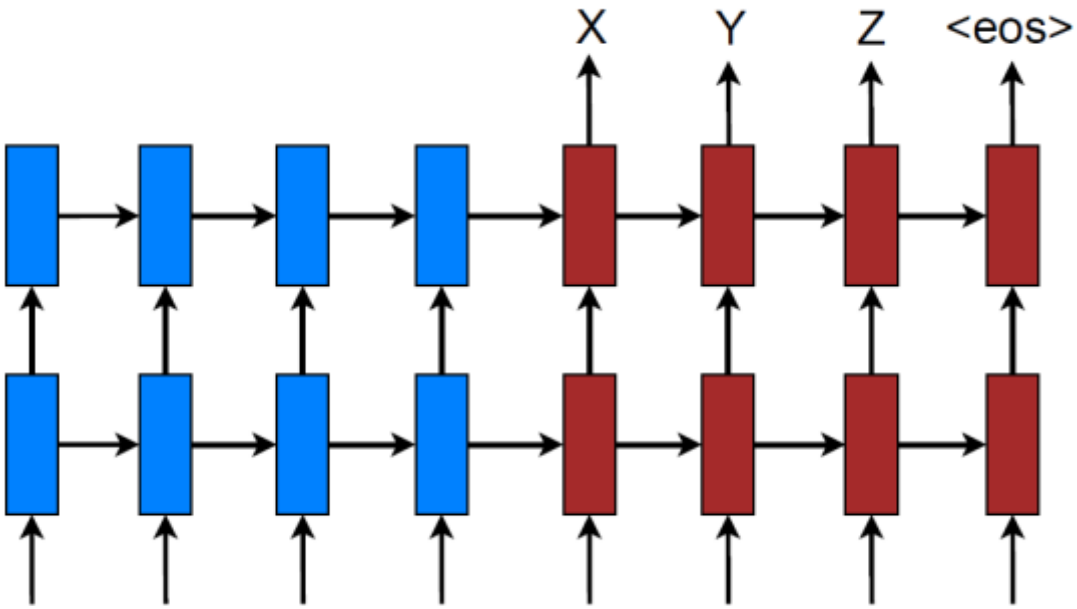
OS	=	Ubuntu 14.04	CPU	=	i76700
Lib	=	Tensorflow 1.0	GPU	=	1080
Python	=	2.7	CUDA	=	8.0

上圖為最終測試的環境，開發時有各自的環境，因此不一一列舉。

Model Description and Improvement

接下來會分別介紹我們使用的各種 Improvement 方式，以及其使用的 Model。

(1) S2VT

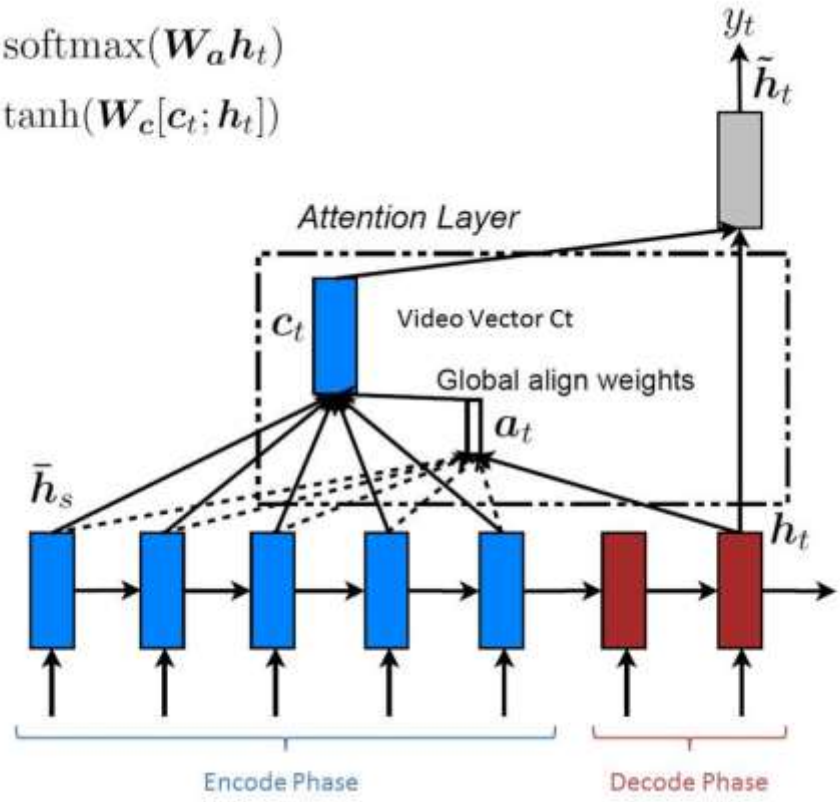


這次作業，我們使用 S2VT model，首先，建立兩層獨立的 lstm layer(weight 無共享)，第一層的 output 為 256 維，在 encoding stage，第一層的 lstm cell 的 input 為 4096 維的 frame feature vector，第二層的 input 為第一層的 output 接上 256 維的 0 向量，所以是 512 維，第二層的 output 與第一層相同，為 256 維，值得注意的是，我們在 encoding 與 decoding stage 同一層的 weight 是共享的，意味著兩個階段的 input 和 output 的維度必須相同，所以在 decoding stage，第一層的 input 為 4096 維的 0 向量，第二層的 input 則為上一層的 output 接上 256 維 target caption 的 word embedding，這裡的 word embedding 並非事先 train 好的，我們一邊 train frame feature->video caption 的過程中，一邊也讓 word embedding 做更新的動作。我們

使用 stocastic gradient decent 作為更新 model 的方法，並使用 adam 作為 optimizer。

(2) S2VT with Global Attention

$$a_t = \text{softmax}(W_a h_t)$$
$$\tilde{h}_t = \tanh(W_c[c_t; h_t])$$



接著，我們加入使用 Attention，在此我們採取 Global Attention，架構如上圖所示。其目標在於，在輸出每一個 Output 時，返回觀察 Encoding 階段的資訊，並且利用權重的方式，去決定這時間的 Output 輸出，應該對那些 Encode Frame 有更多的關注。在此我們採用 Minh-Thang Luong 的 2015 論文，先把 Decode 階段的 Output 乘上 Weight 後，再經過 Softmax，並且已知 Training 及 Testing Data 都剛好為 80 個 Frame Feature，所以我們得到 80 個，對應 80 個 Frame 的 Weight。接著經過 Weighted Sum，我們又得到 Video Vector C，並與原本的 Decode Output 進行疊合(Concat)並通過 Tanh Function，最後在乘上 Weight 使其轉化為 Vocabulary Size 的維度且經過 Softmax，得到最後的單字機率。依循此步驟，最後求得整個 Caption。

(3) Improvement

除了上述的 Model，我們還有額外使用增進效能的方法，第一個為 Linear Decay 的 Sample Scheduling，這是為了避免取 Target 進行 Training 時，Testing 卻是取 Predict 進行 Testing 的不同調行為。藉由 Linear Decay，在每一個 Training Epoch 我們都會得到逐步下降的機率，此機率是用來決定這次是取用 Target 還是 Predict 作為 Training 的依據，此方法我們任位能夠避免 Overfitting

第二個方式是肇因於前述，我們在 Training 只針對每一個影片取一個描述句子，這很明顯會使得整個 Training Data 只有少少的 1450 個，而容易使得整個架構只訓練到 Decode 階段(因為只有 1450 個句子)，所以我們改採把所有句子一起 Training，並且運用 Dropout 以及 Attention，來減低 Same Input Different Output 的困擾。

Experiment and Performance

(1) Experiment Setting

以下圖為固定的參數設定。

Cost Function	=	sequence_loss_by_example			
Cell	=	BasicLSTMCell	Learning Rate	=	Adam
RNN Layers	=	2	Drop-out Rate	=	0.5
Initial	=	Uniform[-0.05, 0.05]	Batch Size	=	50

接下我們會分別比較，上述的不同 Model 所對應的 Bleu 的分數。

因為時間的關係，我們 Training Data 為單影片單句子時 Epoch 為 1000；Training Data 為單影片多句子，則 Epoch 為 160。以下 Model 代稱為，

S2VT：使用 S2VT Model、無 Sample Schedule，並且使用單影片單句子

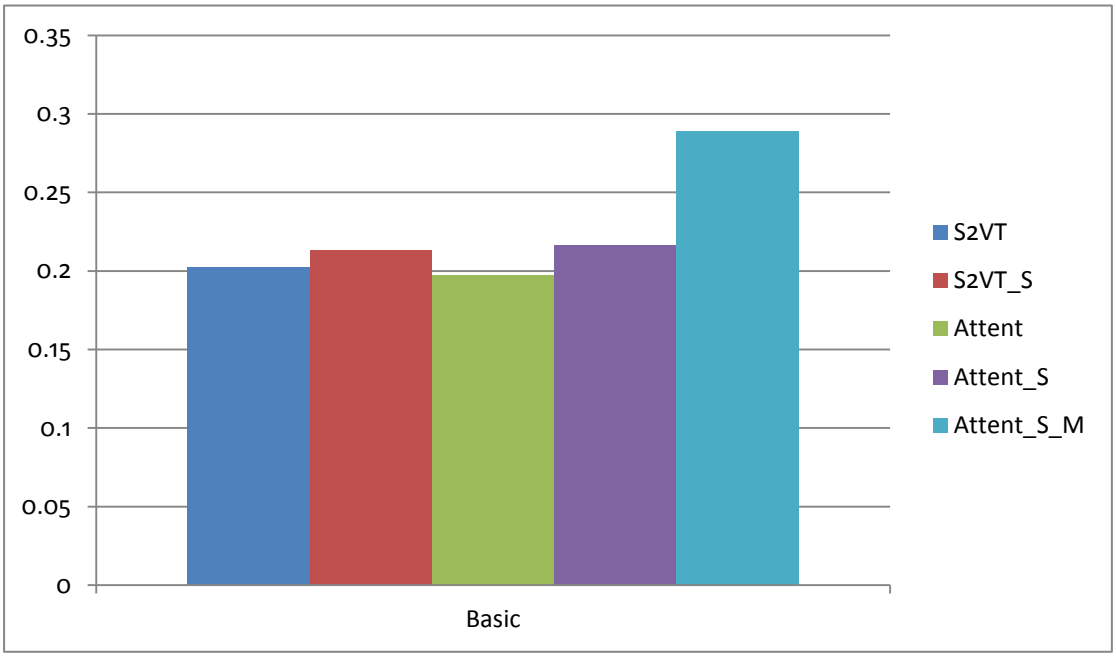
S2VT_S：使用 S2VT Model、有 Sample Schedule，並且使用單影片單句子

Attent：使用 S2VT with Attention Model、無 Sample Schedule，並且使用單影片單句子

Attent_S：使用 S2VT with Attention Model、有 Sample Schedule，並且使用單影片單句子

Attent_S_M：使用 S2VT with Attention Model、無 Sample Schedule，並且使用單影片多句子。

(2) Performance



我們可以發現，只有在 Attent_S_M 時才有顯著的進步，其他幾乎沒有差異太多，很明顯的，這是因為 Training Data 的不足，所以造成即便使用更複雜的 Model，也只會因為 Training 過後造成 Decode 階段 Overfitting，也就是單純看 Input Text 而決定 Output Text 而少考慮了前面 Encode 的資訊。

這在訓練的過程中也可以發現這個現象，Training Epoch 逐步上升時，Loss 會持續下降，而且明顯看見 Predict 幾乎與 Target 同步。但是 Testing Data 卻仍然沒有好的結果。

另外有可能是 Bleu 算法的問題，我們也發現在訓練初期時 Bleu 分數卻是蠻高的。即便 Predict 出的句子多偏向於 A man a a ...，此種通用的句子。不過隨著 Training Epoch 上升，Bleu 分數就會慢慢下降，但 Predict 出的句子就有貼近影片描述的傾向。

Team Division

R04922108	R04922098	R04922086	R04922065
S2VT Report	Attention Report	Report	Report