MLDS HW 2-2 Chinese Chatbot

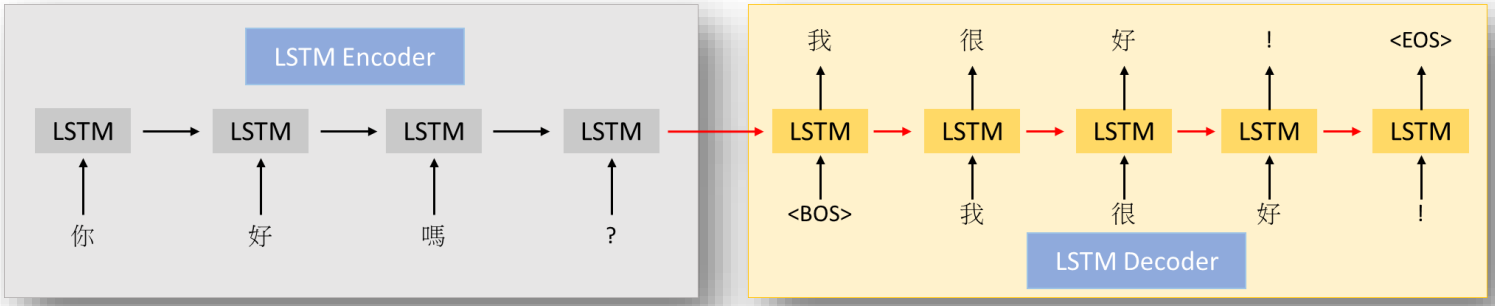
A. Model description

Data Preprocess:

a. Build word dictionary (including word2ix, ix2word) :
首先從 clr_conversation.txt 中收集詞彙， 共有 6347 個詞彙，並依照所設的詞彙出現次數 threshold(2)，將總詞彙壓縮為 3870 個；根據所收集的詞彙建立 word2ix, ix2word 字典，以提供模型訓練及預測使用。

b. Training data preprocess:
clr_conversation.txt 中每一行為一句話，對話和對話之間以+++$+++做分隔。首先利用+++$+++收集每段對話 conv 到對話清單 convs 中，再從每段對話 conv 中，根據前後兩個句子 sent 製作 [question, answer] pair 的清單，形成 training samples。
將 training samples 分成多個大小為 batch_size 的 batch；而在每個 batch 裡將每個 question 做 padding 至該 batch 中 question 句子的最大長度並 reverse，形成 encoder input ；在每個 answer 前加上<eos>後，再做 padding 至 batch 中 answer 句子的最大長度，形成 decoder input； 最後將每句話中的每個詞彙根據字典 word2ix 轉成相對應的 ID vector，生成訓練資料。

Model Structure



Model 架構為 encoder-decoder 架構，由兩層 size 為 512 維的 LSTM 所組成並共享權重；一個單層的 LSTM 作為 encoder，encode 在 data preprocess 步驟中所產生的 encoder input (question)，另一個 LSTM 則作為 decoder，decoder 的 target 為 decoder input shift by 1; 吃入 encoder output 及前一次 decoder 的輸出(answer)作為 input。Loss function 使用 sequence_loss 計算，而 back propagation 則選用 AdamOptimizer 來更新模型的參數。

Inference:

讀取 input test data，將每一句話根據 word2ix 轉成相對應的 ID vector，接著對每個句子做 padding 至 test data 中的最大長度，餵進 seq2seq 的模型中；當 decoder 讀到<BOS>時便開始運作: 在 time step t 輸出經 softmax 後機率最大(argmax)的詞彙(greedy decoding)；並將此詞彙作為 time step t+1 的 decoder input ，持續重複此步驟直到 deocder 輸出<EOS>後結束，最終產生與該 test data 相對應的答句 。另外，除了 greedy decoding 外，也有使用 beam search 來實現 inference 時 decode 的 model。

Model details

Parameters

Input dimension = 4096;

LSTM hidden dimension = 256;

#video LSTM steps = 80;

#caption LSTM steps = 16;

Optimizer = Adam optimizer

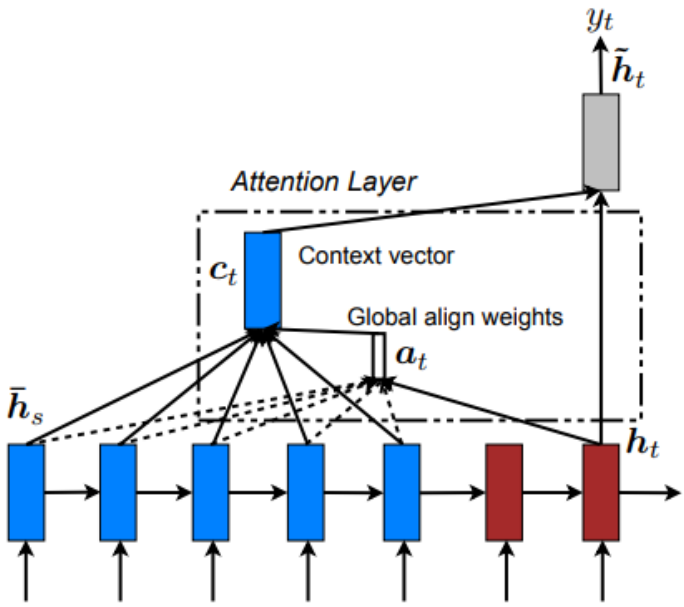
Learning rate = 0.001

Batch size = 32
Epoch = 205

B. How to improve your performance

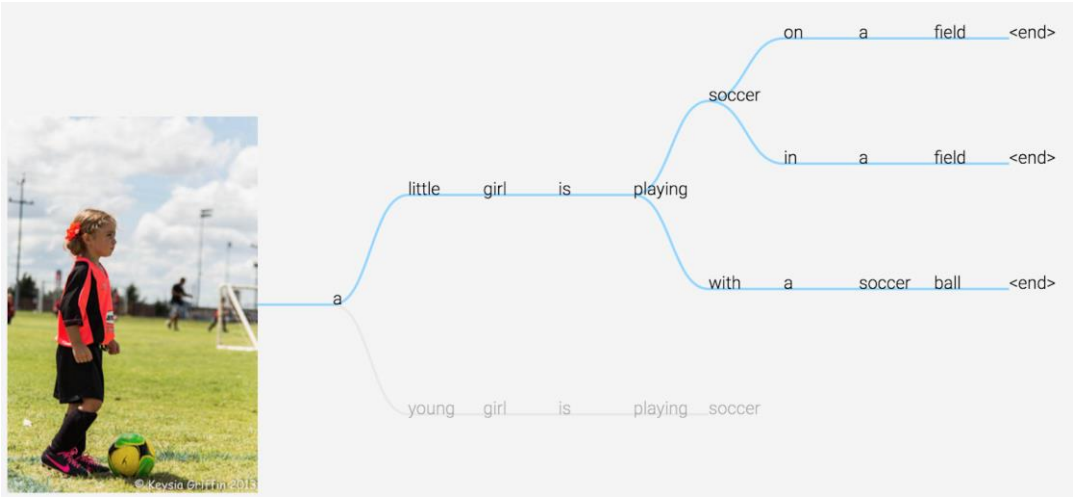
Write down the method that makes you outstanding

a. Attention mechanism



Attention mechanism 的實作是採用 tensorflow 中的 LoungAttention layer，將所有的 encoder output 作為 memory 傳進 attention layer, 每個 decoder output 會經由 context base function 對每一個 encoder output 計算 alignment score； Normalize 所有 scores 後，將每個 encoder output 與相對的 score 相乘後加總(weighted sum)以得到 context vector. 最後同時考慮 context vector 與 decoder output 以得到最後的結果。

b. Beam Search



Beam search 的實作是採用 tensorflow 中的 BeamSearchDecoder，首先對 encoder output 進行 tile_batch，也就是複製 beam_size 份；接著在 decode 的過程中每次皆保留概率前 beam_size 高的詞彙組合，捨棄其餘組合，重複此步驟至遇見<EOS>結束為止。最終將這些句子依照得分排序，得分最高者作為最終輸出。

Why do you use it

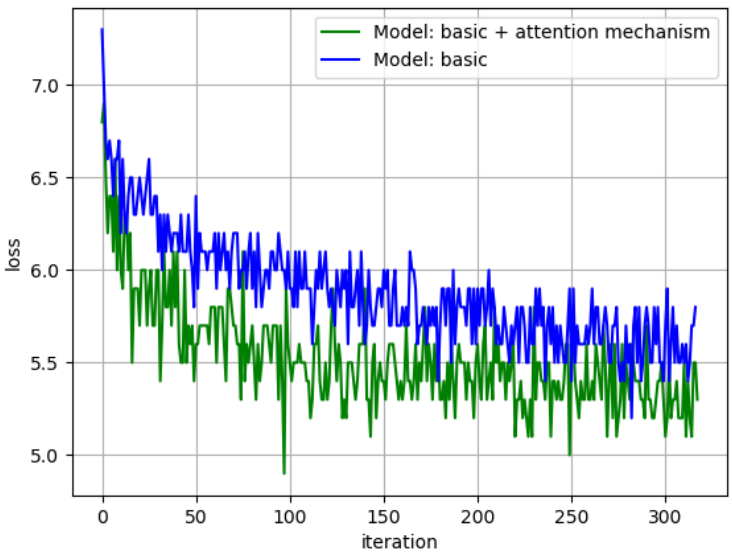
a. Attention mechanism

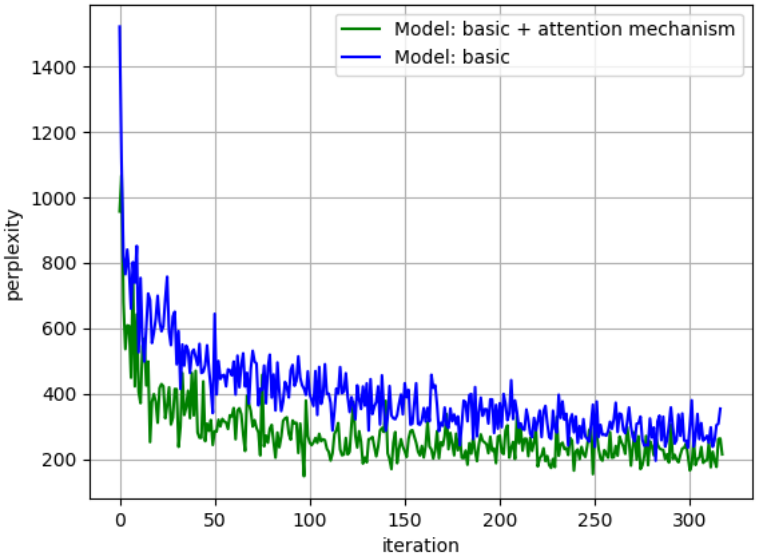
在 Encoder-Decoder 的結構中，輸入序列無論長短都會被 encode 成一固定長度的向量，decode 時則受限於該固定長度的向量；當輸入序列較長時難以保留全部的資訊，使 model 的性能變得很差。Attention mechanism 可以幫助 model 對輸入的每個部份賦予不同的權重，抽取出更加關鍵且重要的資訊，使模型能做出更加準確的判斷。

b. Beam Search

當 model 在 inference 時若採用 greedy search，則 decoder 取概率最高(argmax)的詞彙作為當前輸出，並將此詞彙作為預測下個詞彙的 decoder input，重複此步驟直到 model 輸出<EOS>結束。很顯然在每一步都取概率最大的詞並不能保證最後生成的句子概率是最大的，因此可以利用 beam search 來做動態規劃、全局的 decode，每次皆留下整體概率 top-N 的詞彙組合，捨棄概率低的結果；如此比 greedy search 更能獲得接近最佳解的解。

Analysis and compare your model without the method (比較前 325 個 iteration)





a. Attention mechanism

從上兩張圖觀察可知，在加入 attention mechanism 後確實能有效降低 loss，而在 perplexity 的結果上，也可發現加入 attention mechanism 後的 model 其 perplexity 確實較低，顯示 attention mechanism 所抽取的資訊確實能幫助 model 的 training。

b. Beam search

由下列例子中可看出，經由 Beam search 後所得的 Output sentence 更合乎語意，且也更像對話

Example case:

	Input Sentence	Output Sentence
Directly Inference	真是個乖孩子	我不懂，你
Inference by Beam Search	真是個乖孩子	我不知道你的意思

C. Experimental results and settings

實驗中發現，loss 會大約卡在 4.7 左右無法降下，增加 layer 的深度反而更糟；也有試過 SGD optimizer ，也是會存在一樣的情形。最終經過 tradeoff 後使用以下的設定，以得到最佳的 Chatbot

hidden_dim = 512
num_of_layers = 2
embedding_size = 128
learning_rate = 0.0005
batch_size = 128
num_of_epochs = 50
steps_per_checkpoint = 100

D. 分工

由於另兩位同學決定退選，故此次作業皆由一人 b03901156 完成