Introduction

1. Model description

- 每一個 video 在 preprocess 時，會先平均的選擇 80 個 frame，再把這些 frame 丟進 pre-trained 好的 VGG-16 model 中，然後抽取 VGG-16 的 fc7 這層當成這個 frame 的 feature。最終每個 video 都可以表示成一個維度為 80 x 4096 的 feature。
- 我們的 model 主要是參考 <sequence to sequence - video to text> 這篇 paper，它提出的 model 採用 encoder-decoder 架構，由兩個 hidden layer size 為 1000 的 LSTM 組成，在 encoder 以及 decode 階段都是使用同一個 model (share weights)，而不是分別 train 一個 encdoer 以及 decoder。我們實作了這篇 paper 提出的 model，並在這個基礎上加上 attention mechanism。
- 在 encoder 階段，每次把 video feature 中的 4096 維丟進第一個 LSTM，接著把第一個 LSTM 的 output 丟進第二個 LSTM。當 80 個 4096 維的 feature 都丟進 model 中，encoder 階段就算是結束。
- 在 decoder 階段，我們會先把 <BOS> 丟進第二個 LSTM，然後產出的字在當成下一個 timestamp 中第二個 LSTM 的 input，直到產出 <EOS> 為止。
- 我們的 attention mechanism 使用 soft-alignment，把 encode 階段，每一個 timestamp 中第二個 LSTM 的 output 記錄下來，然後讓 neural network 去學這 80 個 output 的權重，然後把加權過後的 vector 當成 decoder 階段第一個 LSTM 的 input。
- 在 encode 階段，因為沒有 attention 的值，第一個 LSTM 的 input 會 concatenate 一個 padding vector，此外，因為沒有 generate word，第二個 LSTM 的 input 會 conacatenate 一個 padding vector。而在 decode 階段，因為沒有 video feature，所以第一個 LSTM 的 input 也會 concatenate 一個 padding vector。

3. Improvements

- 我們有試過在 decode 階段，attention vector 要加在第一個 LSTM 的 input 還是第二個 LSTM 的 input。後來覺得這個 attention vector 儲存了 video embedded 後的資訊，在 encode 階段 video 的 feature 是餵給第一個 LSTM，因此把 attention vector 串在第一個 LSTM 的 input 比較合理。
- 我們的 corpus 是 training data 以及 testing data 的所有 caption label 的集合。我們發現在 corpus 中出現次數過少的字，必須濾掉，這個步驟會讓結果更好。因為我們 model 預測出來的 word 會以 one-hot encoding 來表示，如果不做這個動作，word

的數量會太多，one-hot vector 的維度太大，caption 出來的結果通常不會太好。因此我們設了一個 word_count_threshold，把低於 threshold 的 word 濾掉。

- 在 training 的 decode 階段，我們最初是把每一次產生的 word 當成下一個 timestamp 中第二個 LSTM 的 input，但這個 train 法有缺點，如果前幾個產生的字偏離了 ground true，那後面的字會越來越偏離 ground true，這會導致 training 的不穩定。所以在 training 過程中，假設你餵了 <BOS> 進去，產生了句子的第一個字 w1，在下一個 timestamp 中，不是把 w1 餵進第二個 LSTM，而是把 ground true 的第一個字 g1 餵給第二個 LSTM，直到把 ground true 的每一個字都餵進去後，decode 階段就結束。
- 使用上述的 train 法後，train 出來的 model 會有不錯的 caption 能力。不過這裡會有另一個問題，就是 training 與 testing 的不一致。在 testing 階段，因為沒有 ground true，所以當產出一個 word 後，會把該 word 當成下一個 timestamp 中第二個 LSTM 的 input，但 training 時是把 ground true 中的字餵到第二個 LSTM。這個問題或許可以透過 <Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks> 這篇論文提出的 scheduled sampling 來得到些許的改善，這個部分可以當成未來改進的方向。

4. Experiment observations and settings

- Experiment observations
  - 如果實際去看 model 產生的句子，用 Adam optimizer 的話，差不多 1000 epoch 出頭，產生的句子就不會再有大變化了。用 RMSProp optimizer 的話，差不多 2000 epoch 左右產生的句子就不會再有大變化。用 SGD optimizer 的話，到了 3000 epoch 左右產生的句子仍不穩定。
  - 以上三種優化器，Adam 收斂最快，不過以產出的句子質量來說的話，RMSProp 會稍優一點 (部分產生的句子跟 ground true 一模一樣，頗驚人)。
  - word_count_threshold 如果太小，會導致產生 <unk> 的機率變高，word_count_threshold 如果太高，產出的句子會包含更多高頻率的字，這會讓 BLEU@1 score 稍微上升，但仔細去看產生的句子會發現沒有特別好。最後找到最佳的 word_count_threshold 為 6，依照這個 threshold 過濾出來的 word 大約有 3100 個。
  - batch size 只要別太小，對結果沒有太大的影響。目前試過 batch size 從 20 到 50，如果其他設定不變，產出的句子質量以及 BLEU@1 score 不會差很多。
  - 加上 attention 後，得到的 BLEU@1 score 會上升，而產出的句子多樣性會下降，如果想衝分數，有 attention mechanism 會更好，如果純以產出的句子質量而論，原先的 2 layer LSTM model 會比較理想。

- Experiment settings
  - epochs = 1880
  - learning_rate = 0.0001
  - batch_size = 50
  - word_count_threshold = 6
  - RNN model: 2 layer LSTM (hidden layer size = 1000)
  - loss function: tf.nn.softmax_cross_entropy_with_logits
  - optimizer: tf.train.RMSPropOptimizer
  - training time: 1.5 days
  - max accuracy: 0.29 in BLEU@1
- Experiment environments
  - max accuracy: 0.29 in BLEU@1
  - OS: CentOS Linux release 7.3.1611 (Core)
  - CPU: Intel(R) Xeon(R) CPU E3-1230 v3 @ 3.30GHz
  - GPU: GeForce GTX 1070
  - Memory: 16GB DDR3
  - Python2.7.5
  - Libraries:
    - numpy 1.12.0
    - pandas 0.19.1
    - tensorflow 1.0.1
    - tensorflow-gpu 1.0.1