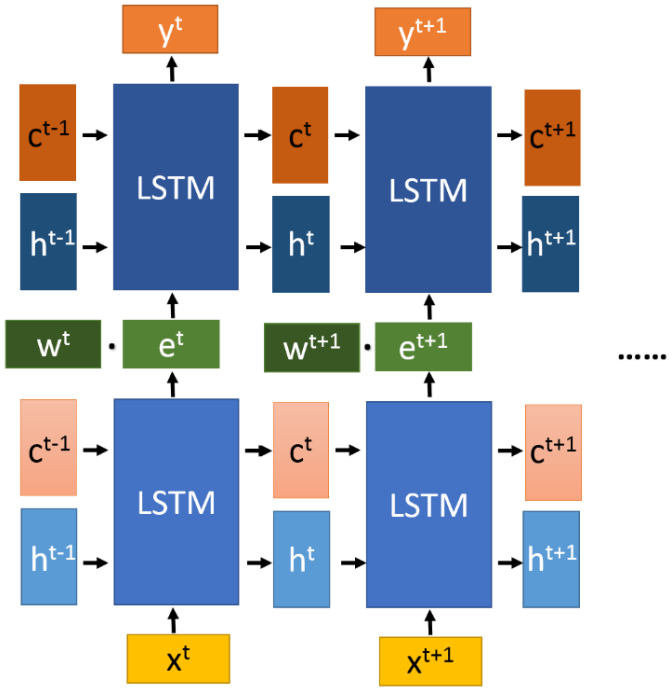
Homework2 Report

環境設定

- OS - `4.4.0-72-generic #93-Ubuntu x86_64 GNU/Linux`
- CPU - `Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz`
- GPI - `GeForce GTX 1080 8112MiB`
- Memory - `62GiB`
- Libraires - `python v3.5.2`, `tensorflow v0.11.0`, `numpy v1.12.1`

模型簡介

1. Sequence to sequence



Output - softmax probability distributions (n-words d)

Layer2 - weight matrix (256d, n-words d) and bias (n-words d)
Activation: softmax

LSTM2 [Encoding Stage]
- input the information of LSTM1 only (512d)
- reset the memory cells to zeros (256d)

LSTM2 [Decoding Stage]
- input the word embedding and LSTM1's info (512d)
- output the info required to predict the current word (256d)

Hidden Layer - concatenate the followings (512d)
1. word embedding of the previous word or padding (256d)
2. output of LSTM1 (256d)

LSTM1 [Encoding Stage]
- input the latent vector of the image (256d)
- reset the memory cells to zeros (256d)
- combine the informantion of image and memory (256d)

LSTM1 [Decoding Stage]
- input padding (256d)
- output the information based on memory only (256d)

Layer1 - weight matrix (4096d, 256d) and bias (256d)

Input - image features for each frame (4096d)

- 模型的輸入，是80個frame的圖像features
- 答案則是長度為20字的描述，不足則補上自定義的空白字元
- 損失函數以這20個字的「模型預測與答案的交叉熵」之和來衡量
- 需要訓練的參數包含以下幾項
  - 把圖像4096維的特徵，轉成256維潛在向量所需的weight和bias
  - 依序讀取圖像潛在向量的LSTM
  - 大小為「詞彙量×256」的emedding matrix
  - 依序產生詞彙訊息向量的LSTM
  - 把256維的辭彙訊息向量，轉成「維度等於詞彙量大小」的機率分布，所需的weight和bias

- 模型的前段在讀取圖片特徵，因此第二個LSTM所需的word embedding會先補上全零的padding，此時的輸出也沒有用途，重要的是兩個LSTM向後傳遞的記憶向量
- 模型的後段則用於產生詞彙，因為不再有圖片，故第一個LSTM的輸入則會是全零的padding，而第二個LSTM則會同時處理「來自第一個LSTM的輸出、前一個字embedding以及前面傳遞過來的記憶訊息」，並用來預測當前詞彙的機率分布

2. Attention

- 為了讓產生句子的某些詞彙時，可以專注於某幾張圖片，因此加入attention
- 主要調整的地方，在圖中的w^t及e^t，並只限於decoding階段
- 首先，將原本各為256維的w^t, e^t整合、投影到另一個256維的向量上面
  $p^t = \text{concat}(w^t, e^t) \cdot W_p + b_p$，其中，$W_p \in \mathbb{R}^{512\times256}$且$b_p \in \mathbb{R}^{256}$，都是需要訓練的參數
- 此外，利用第二個LSTM前一步的記憶向量h^{t-1}，試著生出一個大小為80維(即frame數量)的遮罩，每一維代表的該frame的權重，總合為一
  $m^t = \text{softmax}(h^{t-1} \cdot W_m + b_m), \text{where } W_m \in \mathbb{R}^{256\times80} \text{ and } b_m \in \mathbb{R}^{80}$
- 有了遮罩m^t，以及在encoding階段中原本沒有用到的80個frame之輸出，即
  $y^m = [y_1, \dots, y_{20}]$，即可得$q^t = \sum_{i=1}^{80}(m^t)_i(y^m)_i \in \mathbb{R}^{256}$
- 最後，把p^t和q^t接在一起變成512維的向量，取代原本的w^t和e^t，作為第二個LSTM的輸入，即完成了attention的實作

優化概述

1. Beam Search

- 在預測的時候，每一個Step我們可以保存當前最好的k個句子。在下一個Step時，再把前一次保存的k個句子，都加上新的單詞（總共就有$k \times |V|$個新句子和機率），從中再選出最好的k個保存，以此類推
- 最後可以拿到k個結果，選機率最大的那一個並一步步往回尋找parent beam，把句子完整生出來
- 實作內容主要如下

```
probs = tf.reshape(probs + log_beam_probs[-1],
    [-1, self.beam_size * self.n_words])
best_probs, indices = tf.nn.top_k(probs, k=self.beam_size)
words = indices % self.n_words
beam_parent = indices // self.n_words
...
```

2. Schedule Sample

- 考慮訓練與測試時，最大的不同就是正確答案的有無
  - 訓練時，decoding階段可以直接拿前一個字的正確答案拿來當作第二個LSTM的輸入
  - 測試時，則只能從預測結果中，選機率最大的那一個字，當作是答案

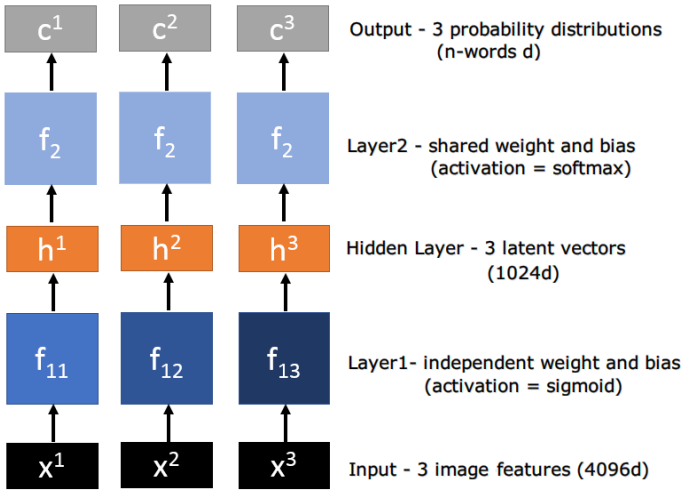
- 為了縮小這個差距，或者說讓機器認知這一件事，我們在訓練時，有一定的機率，不直接拿取正確答案，而是拿前一步預測中機率最大的字，餵給第二個LSTM當輸入
- 實作內容主要如下

```
ss_list = [(np.random.uniform(0, 1) > prob) for i in range(20)]
...
for i in range(20):
    current_embed = tf.cond(ss_list[i],
        lambda: tf.nn.embedding_lookup(self.Wemb, caption[:, i]),
        lambda: tf.nn.embedding_lookup(self.Wemb, max_prob_index))
        ...
```

實驗與觀察

1. 與DNN比較

- DNN的架構



- 輸入為第$20/40/60$個frame的圖像features
- 答案是caption的句子中，文件頻率(document frequency)最低之三個字的one-hot向量
- 損失函數以「三個交叉熵的加總」來衡量，因為答案的三個字並沒有排列順序，所以在$3! = 6$種「交叉熵之和」當中，取最小的那一個
- 在測試、產生句子的時候，除了c^1, c^2, c^3三個機率分布中最大的三個字之外，在加上「is, the, a」這些常用字，組成最終的句子(順序是亂的)
- 實驗結果 - `n_epoch=600, BLEU@1=0.252`
- 總體來說，因為 `bleu@1` 並不考慮字的順序，DNN的表現很好，用極少量的參數和簡單的模型，就能達到baseline的要求

2. seq2seq 的不同架構比較

- Attention
  僅比較基本模型和加上attention layer後的表現，實驗結果沒有非常明顯的差異，不過大致上加上attention layer後表現有變好

model	avg. bleu score
basic	0.2847
attention	0.2989

- Schedule Sampling
  我們Schedule sampling的機率會隨著訓練過程線性上升，然而實驗結果難以看出差異

model	avg. bleu score
basic	0.2847
schedule sampling	0.2880

- Beam Search
  我們比較不同數量的beams的差異，但差異並不明顯，推測是因為預測結果中不同字的機率差異太大，因此使用Beam Search不太影響結果

beam_size	1	2	3	5
basic	0.2847	0.2848	0.2854	0.2849
attention	0.2970	0.2976	0.3003	0.3010

組內分工

- 簡瑋德 - 實驗、報告
- 黃兆緯 - 模型實作、優化
- 劉岳承 - 優化、分數計算
- 李承軒 - 優化、前處理