# MLDS HW2 Seq2Seq & Attention -- Report

物理三 B03202017 李漪莛 | 物理三 B03202047 王昱翔

# 1 Environment

- OS: Ubuntu 16.04.2 LTS
- CPU: Xeon E5-2630V3
- GPU: NVIDIA GTX 970
- Memory: 4G
- Libraries: Tensorflow r1.0, numpy 1.12.0, CUDA8.0

# 2 Model descriptions

共兩層 LSTM，中間由 fully-connected 改變矩陣的維度，詳情如下：

## 2.1 預處理

- 除了「'」（一撇）留著，其他所有不是英文字母的字元都用空白代替。
- 把所有字母轉成小寫。
- 在每句的頭尾分別加上 BOS、EOS，代表句子的開頭跟結尾。
- 把句子 tokenize，vocabulary size = 5995（含 BOS、EOS）。

## 2.2 Encoding LSTM

- CNN feature (dim = 80*4096) → fully-connected layer → ReLU → X1 (80*300)
- X1 (80*300) → encoding LSTM: tf.nn.dynamic_rnn (BasicLSTMCell) → 得到 80 個 time step 的結果，令為 hidden_pattern，儲存做 attention。
  註：等到 encoding 全部結束後，decoding 才會開始運作。

## 2.3 hidden_pattern

- 取出 hidden_pattern 分別對應到第[10, 20, 30, 40, 50, 60, 65, 70, 75, 79] time step 的十筆，分別和 $y_{t-1}$ 去做 attention。
  （只取 10 筆是因為若 80 個全取的話，之後的運算會使我們的 gpu 記憶體不夠。）

## 2.4 Decoding LSTM

共 44 個 time step（label 中最長句子有 44 個字），每個 time step 都做以下步驟:

- 前一個輸出（$y_{t-1}$）的 one hot vector (dim = 5995) → fully-connected → last_word (dim = 300)
  註：以上這層 fully-connected 是整個 model 中唯一沒有加上 ReLU 的
- 令 alpha (dim = 10) = hidden_pattern (10*300) * weight (300*300) * last_word (300)
  找出 alpha 最大的值的 index 令為 idx
- concatenate hidden_pattern[idx] 和 last_word 成 X3 (dim = 600)

- X3 (600) → fully-connected → ReLU → X4 (400)
- X4 (400) → LSTM → X5 (400) → fully-connected → ReLU → outputs (5995)

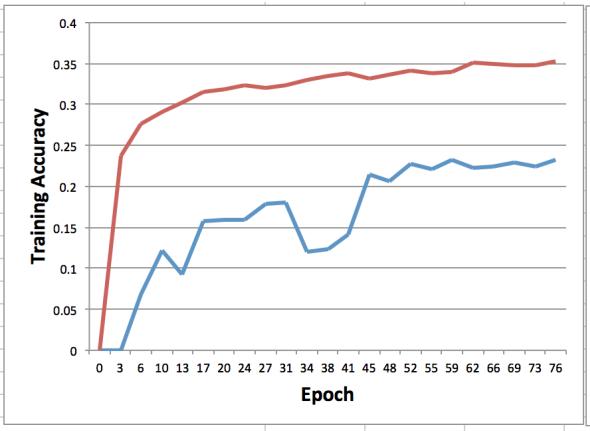
outputs 做 softmax 和 cross entropy

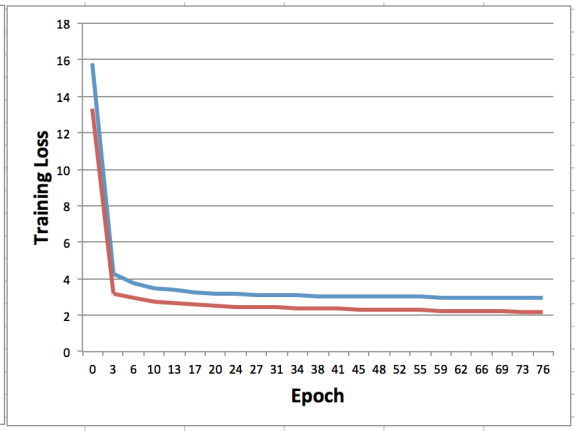
# 3 How to improve performance

## 3.1 拿掉 Encoding LSTM 輸出時的 fully-connected + ReLU

在 encoding LSTM 結束時，原本有再接上一層 fully connected + ReLU。

發現 training acc 和 loss 結果不太好，訓練初期如下圖：藍色表示有這層 layer，紅色表示拿掉 layer，此時兩層 LSTM units 數目都是 256。




## 3.2 加入 attention

在有 attention 但還沒加入 schedule sampling 情況下，在 training 的 acc 和 loss 都能得到較好的結果：

| LSTM units = 300+400, epoch = 300 | no attention （直接使用最後 time step 的 hidden_pattern） | | attention（如 model description 所示） | |
|---|---|---|---|---|
| | acc | loss | acc | loss |
| training | 0.473 | 1.631 | 0.523 | 1.458 |

不過，在 validation 和 bleu 都只進步一點，bleu 在 attention 時只進步約 0.2%。

## 3.3 關於 attention 中的 alpha

我的 attention 是照李宏毅老師在 attention 投影片 P14 的架構做的，其中關於 alpha 的設定對於成果影響非常大，以下比較三種情況：

(1) 做 attention 得到 alpha，直接把各 time step 的 alpha 和 hidden_pattern 做 weighted sum，會發現幾乎訓練不起來，training acc 沒有超過 5%過。

(2) 將上述的 alpha 先經過 softmax，再和 hidden_pattern 做 weighted sum，training acc 可到 20~30%。

(3) 最後決定只取 alpha 中最大的值，直接把對應到這個 time step 的 hidden_pattern

傳到下一層，會發現 training acc 可超過六成（到這邊 over fitting 所以沒有繼續訓練下去）

由以上推估，hidden pattern 直接做相加反而會喪失他已訓練好的性質。

### 3.4 從 training data 中選句子

在 training label 中每筆都有 18~20 個句子，我利用幾個方式篩選，條列如下：

- 篩選長度：句子少於 6 個詞的直接踢除，句長超過 12 個詞的直接加入。
- 句首是 ”a”, “an”, “the”，搭配句子的第 3 或 4 個字是”is”。
- 句首是複數詞，搭配句子的第 2 或 3 個字是 ”are”。
- 句中有連接詞 "and" 或副詞，如 “on”, “into”, “at”, “with”…。

某影片若符合以上條件的句子 > 3 句，則會用篩選過的句子去訓練 model，若 <= 3 句，則使用原先所有 label。統計後，1450 筆 training data 中，有 1435 筆左右，能夠找到符合以上的判別條件，作出篩選。

加了這步，在其他條件相同情況下，bleu 可從 26.1%進步到 29.0%，其他比較在 4.4。

### 3.5 計算 validation loss，防止 over-fitting

訓練時把最後 50 筆 training data 當作 validation data，並計算他在兩種情況的 loss：

1. sampling1：last_word 利用正確答案當作前一個字
2. sampling2：last_word 使用自己預測的前一個字

觀察這兩種 loss，可作為調整 schedule sampling 在兩種方法各需多少 epoch 的依據。

### 3.6 Schedule Sampling

前 30~50 個 epoch 全採用 sampling1，之後隨 epoch 線性增加（成正比）sampling2 的比例，增加幅度由 validation loss 做調整，最後 30~50 個 epoch，全採用 sampling2。

加入 schedule sampling，在 sampling1 結果會稍差，但在 sampling2 的 loss 結果會好許多！也因為 testing 是用 sampling2 的方法計算，因此我相信 sampling2 計算的 loss 是較有指標性的。

| LSTM units = 300+400, epoch = 100 | 用 sampling1 計算結果 | | | | 用 sampling2 計算結果 |
|---|---|---|---|---|---|
| | training acc | training acc | validation acc | validation loss | validation loss |
| no sampling | 0.374 | 1.966 | 0.321 | 2.302 | 5.834 |
| schedule sampling | 0.330 | 2.294 | 0.305 | 2.506 | 3.082 |

## 4 Experiments settings and observation

### 4.1 Settings

- Learning rate = 0.001
- Epoch = 300
- Batch size = 50
- Units of LSTM = （Encoding + Decoding）= 300+400
- Optimizer = Adam

### 4.2 LSTM units 的數目

因為我們所使用的 GPU 限制，兩層 LSTM units 數目合計不可超過 800，否則記憶體會不夠，測試後發現收斂速度：

（Encoding + Decoding）= 300+400 > 400+300 > 300+300 > 256+256。

由上可知，總體 units 數目越多，效果越佳；若在總 units 數目固定的情況下，decoding LSTM 比 encoding 需要更多的 units 來提昇 performance。

### 4.3 Activation Function：是否加入 ReLU

相同情況下，拔掉所有 ReLU 的比較如下；可發現有 ReLU 的收斂速度稍快一些（同樣 epoch 下結果稍好）。

| LSTM units = 256+256, epoch = 80 | 全部 ReLU 拔掉 | | 加 ReLU（如 2 所示，共三層） | |
|---|---|---|---|---|
| | acc | loss | acc | loss |
| training | 0.331 | 2.114 | 0.353 | 2.091 |

### 4.4 Training label 的結果比較

由下表可發現：只取一句時，training 的結果非常好，但 validation 較差，表示 over-fitting；而從篩選過的句子隨機訓練，validation 結果最好。

| LSTM units = 256+256, epoch = 75 | 只取影片中固定的其中一句做訓練 | | 從所有句子裡隨機挑選做訓練 | | 從篩選過的句子隨機挑選做訓練 | |
|---|---|---|---|---|---|---|
| | acc | loss | acc | loss | acc | loss |
| training | 0.728 | 0.692 | 0.331 | 0.2114 | 0.411 | 1.909 |
| validation | 0.314 | 3.032 | 0.307 | 2.378 | 0.346 | 2.314 |

### 4.5 BOS 的重要性

在做 testing 時，一開始忘記在第一個字先給 BOS 的 one hot vector（只輸入了一個全部是零的向量），發現會出現一堆較少出現的字當開頭，比如：chair, plastic，等等；直到加入 BOS 就幾乎都會出現 "a man is..." 開頭等的通順句子了。

## 5 Team divisions

李漪莛：model、post-process、report

王昱翔：preprocess、model、彙整並上傳