深度學習

HW10

學號 : B103012002

姓名 : 林凡皓

一、 Image Feature Extraction

在 Image caption model 中，首先重要的是 encoder，此 encoder 會接收一張圖片作為輸入，並生成用於解碼的特徵。

我們使用小型的 RegNetX-400MF 作為骨幹網路，以減少訓練時間。

對於 vanilla RNN 和 LSTM，我們使用 average pooled features 來解碼，而對於 attention LSTM，我們使用 learning attention weight 來聚合空間特徵。

二、 Word embedding

在深度學習系統中，我們通常使用向量來代表字母。單字中的每個字母都會與一個向量相關，這些向量將會與系統其他部分一起學習。

這邊實作 WordEmbedding 類別來將字母轉換為向量。

- 實現方法：

  初始化部分，先創建一個參數 W_embed，其 shape = (vacab_size, embed_size)。使用 torch.randn 來初始化該矩陣，並將其除以 $\sqrt{vocab\_size}$ 來進行標準化。

  接著實作 forward 方法，這個方法主要是根據輸入的字母，從 W_embed 中找到其對應的向量，可以透過 index operation 來完成。

- 執行結果：

  ```
  out error:  2.727272753724473e-09
  ```

  Error 非常低，代表說該類別功能正確。

三、 Temporal Softmax Loss

在 RNN 語言模型中，我們在每個 timestep 產生出單字中每個字母的分數。

由於我們知道各個 timestep 的 ground-truth，因此我們在每個 timestep 採用 cross entropy loss。我們將 loss 進行時間上的總和，並在 minibatch 上進行平均。

但是這邊有一個問題，就是由於我們是對 minibatch 進行操作，而不同的 caption 可能會有不同的長度，因此我們在每個 caption 的尾端加上'<NULL>'以便它們都具有相同長度。我們不希望這些'<NULL>'也加入 loss 的計算，因此我們需要一個額外的參數 ignore_index 來告訴程式碼在計算 loss 時要忽略掉那些 index。

實作 temporal_softmax_loss 來完成 loss 的計算。

- 實現方法：

透過 torch.nn.functional.cross_entropy 來實現。

torch.nn.functional.cross_entropy 可以傳入的參數有 input、target、weight、size_average、ignore_index、reduce、reduction、label_smoothing。這邊主要為用到 input、target、ignore_index、reduction。

Input 的部分即為 x，不過關於維度的部分，我們需要將 timestep 的維度 T 放到最後，這樣才能使用 cross entropy 計算 loss，因此可以使用 permute(0, 2, 1)將 timestep 維度與最後的維度做交換。

Target 即為 y。

Ignore_index 即為呼叫函數時的輸入，主要用來在計算 loss 時忽略掉一些標籤。

Reduction 的部分要設置維'sum'，因為我們希望將計算出來的 loss 進行時間上的總合，最後再對 minibatch 取平均。

- 執行結果：

  根據不同的情況去計算 loss，結果如下

```
2.0746383666992188
20.695470809936523
2.0829384326934814
```

  這些樹直接與預期的數值接近，代表說函數功能正確。

四、 Captioning Model

我們要將所有東西封裝成一個 captioning 模組，該模組有一個通用的結構，可以根據 cell_type 參數來控制要用於 RNN、LSTM 或是 attention LSTM。目前只需要實作 cell_type = 'rnn'的部分。

- __init__：

  1. 實現方法：

     初始化部分主要是要初始化 output projection、feature projection、word_embedding 和 backbone。

     Output projection 的部分為將 RNN 的 hidden state 映射到字母機率的層，可以透過 Linear layer 來做維度的改變。

     Feature projection 的部分為從 CNN pooled feature 映射到 h0 的部分，一樣可以利用 Linear layer 來做維度的轉換。

     Word_embedding 的部分即為先前定義好的 WordEmbedding 類別。

骨幹網路的部分，可以透過創建一個已經定義好的 RNN 類別來實現。

- forward :

1. 實現方法 :

實作 RNN forward path 的部分來計算 loss，backward path 會利用 autograd 來實現。

首先要將輸入的字串分割成 captions_in 和 captions_out。

captions_in 為整個字串除了最後一個字母，而 captions_out 為整個字串除了第一個字母。captions_in 即為 RNN 之 input，而 captions_out 為 RNN 預期的輸出。

接著要將輸入的字母做 embedding，此部分利用 word_embedding 實現。

關於輸入圖片，我們需要經過 Encoder 將圖片轉換為特徵，此部分透過小型 RegNet-X 400MF 來實現。

將透過 Encoder 得到的特徵 x 送進 feature projection 來映射成 h0，並透過骨幹網路(RNN)來產生 hidden state vector。

最後再透過 output projection 將 RNN 的 hidden state 映射到字母機率，並利用先前定義好的 temporal_softmax_loss 來計算 loss。

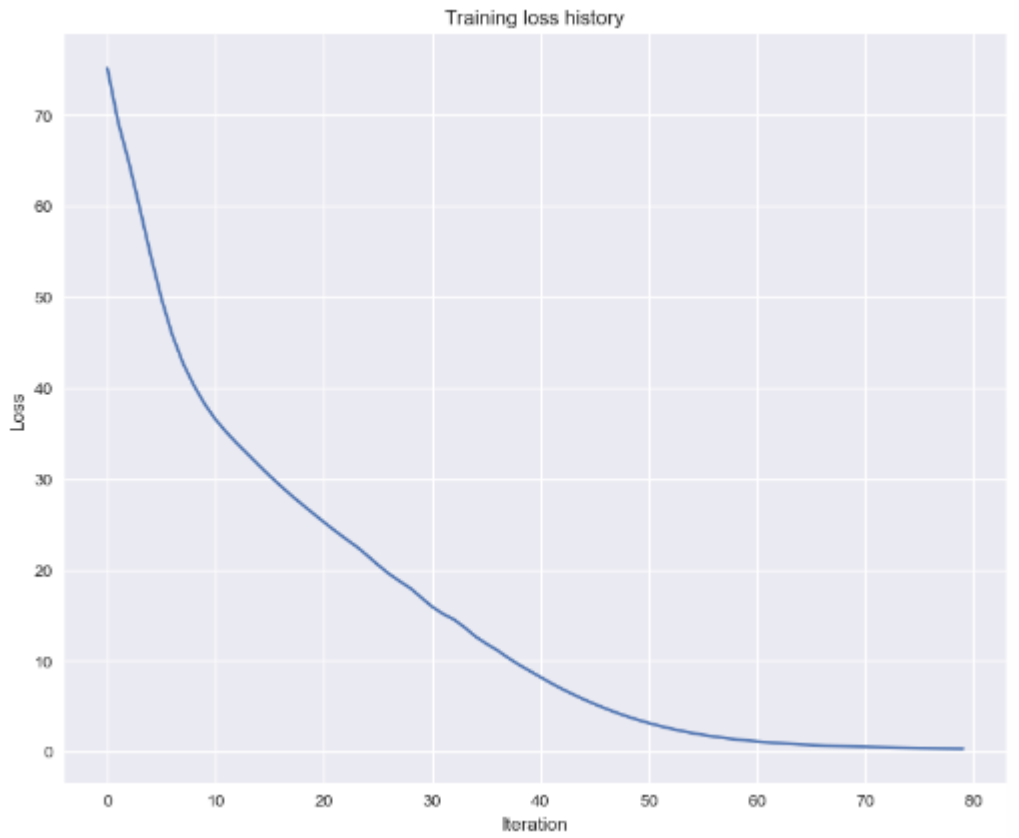
2. 執行結果 :

```
For input images in NCHW format, shape (2, 3, 224, 224)
Shape of output c5 features: torch.Size([2, 400, 7, 7])
loss:  150.60903930664062
expected loss:  150.6090393066
difference:  0.0
```

由結果可以看到，計算出來的 loss 與預期的 loss 完全相同，代表說此函數功能正確。

五、 Overfit Small Data

為了確認剛才實作的每一樣東西都可以正常運行，我們拿 50 張圖片來讓模型 overfit。訓練結果如下



可以看到經過 80 次的迭代後，loss 來到趨近於 0 的程度，即成功的 overfit 資料。

六、 Inference : Sampling Captions

Image captioning 模型在訓練與測試階段和分類器的行為模式不同。

在訓練階段，我們將 ground-truth 的 caption 在每個 timestep 餵給 RNN。

在測試階段，我們從單字的分布中採樣，並將樣本在下一個 timestep 中作為輸入餵給 RNN。

實作 CaptioningRNN.sample，並訓練模型以及對 training data 和 validation data 做採樣。

- 實現方法：

  在每個 timestep 中，我們會對當前的字母做 embedding，接著將它與先前的 hidden state 送進 RNN 已取得下一個 hidden state。利用此 hidden state 來取的每一個字母的分數，並將擁有最高分數的字母做為下一個字母。

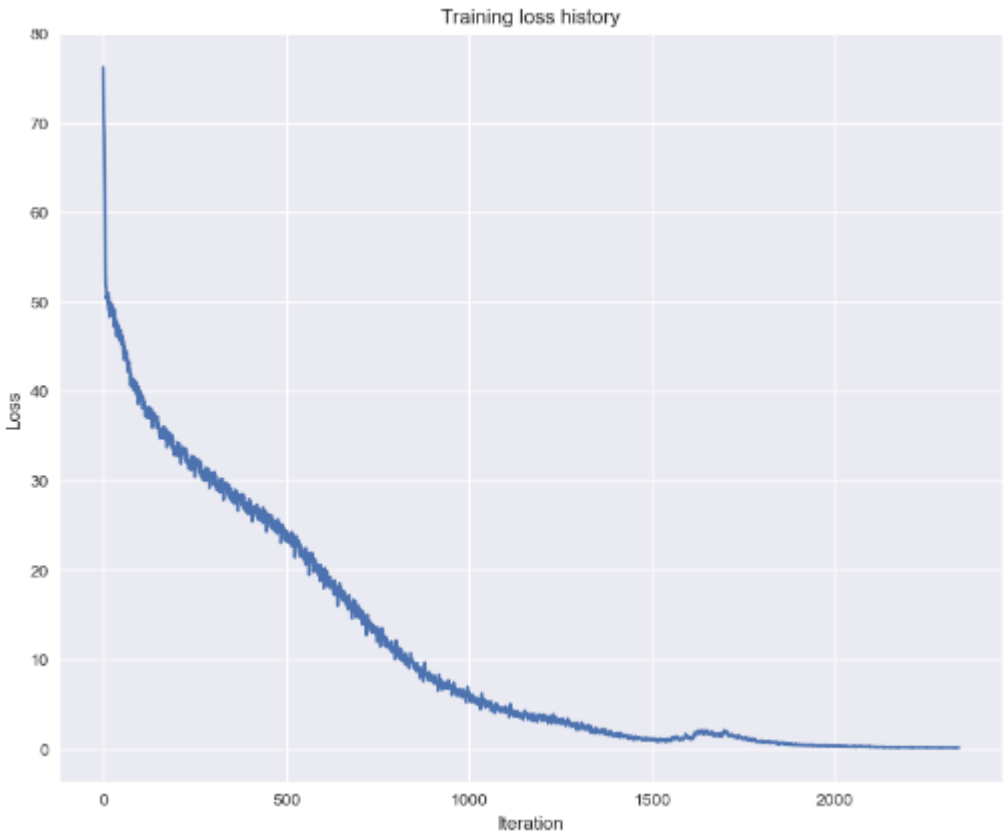
  首先將圖片經過 encoder 以取得特徵，接著利用 feature projection 將特徵映射成 hidden state。

  在每一個 timestep 中，透過先前定義好的 word_embedding 來將字母做

embedding，並將 embed 完的結果與前一個 hidden state 利用 step_forward 來產生下一個 hidden state。將 hidden state 利用 output projection 映射到分數，並從中取最高分做為下一個字母，將此字母存放到 captions 中。

為了確保每個樣本都以'<START>'作為開頭，因此要創建一個形狀為(captions.shape[0], 1)的張量，並將其與 captions 連接起來。

- 訓練過程 ：



由結果可以看出，訓練 60 個 epochs 後，loss 會降到趨近於零的程度。

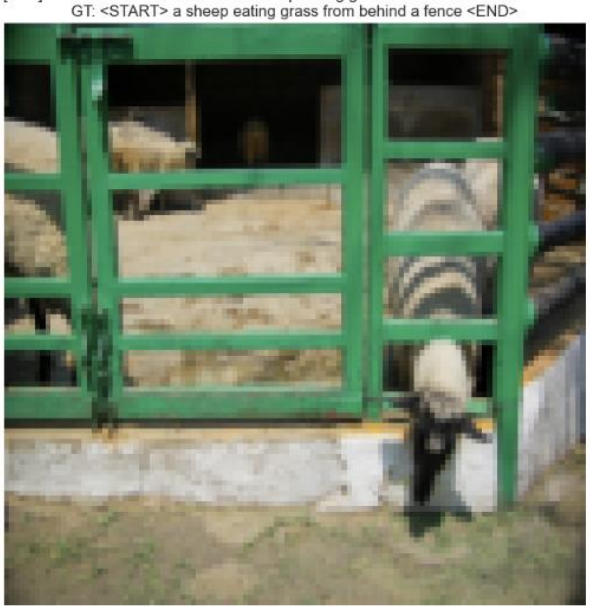
- 採樣結果 ：

接著分別查看對 training data 與 validation data 做採樣的結果。

1. Training data :

   部分結果如下

[train] RNN Generated: <START> a sheep eating grass from behind a fence <END>
GT: <START> a sheep eating grass from behind a fence <END>



[train] RNN Generated: <START> a bunch of glass <UNK> filled with yellow <UNK> and water <END>
GT: <START> a bunch of glass <UNK> filled with yellow <UNK> in water <END>

[train] RNN Generated: <START> a large group of people with some tennis <UNK> <END>
GT: <START> a large group of people with some tennis <UNK> <END>

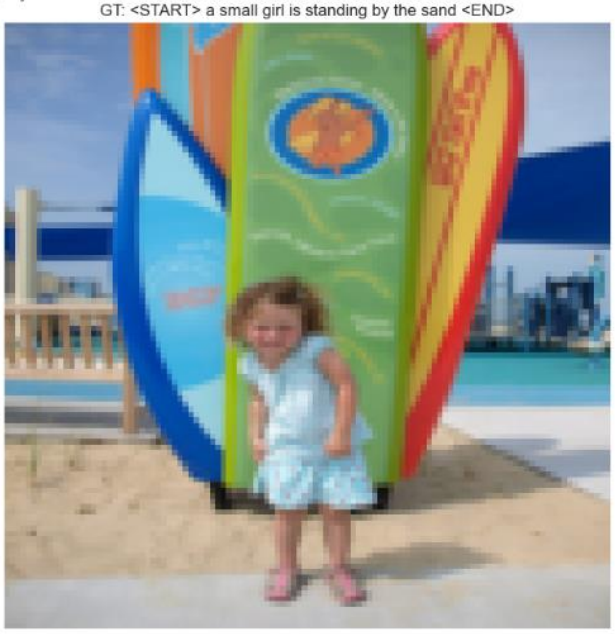
由結果可以看出，對於訓練資料來說，RNN 可以很成功的採樣出圖片的內容。

2. Validation data :

部分結果如下

[val] RNN Generated: <START> two <UNK> <UNK> on a <UNK> with a large open building <END>
GT: <START> a young boy standing next to a table near a road <END>

[val] RNN Generated: <START> a man with a <UNK> attached to a small man <END>
GT: <START> a small girl is standing by the sand <END>



由部份結果可以看到，對於 validation data 來說，RNN 基本上是無法順利採樣出圖片中的內容。

七、 LSTMs : Step Forward

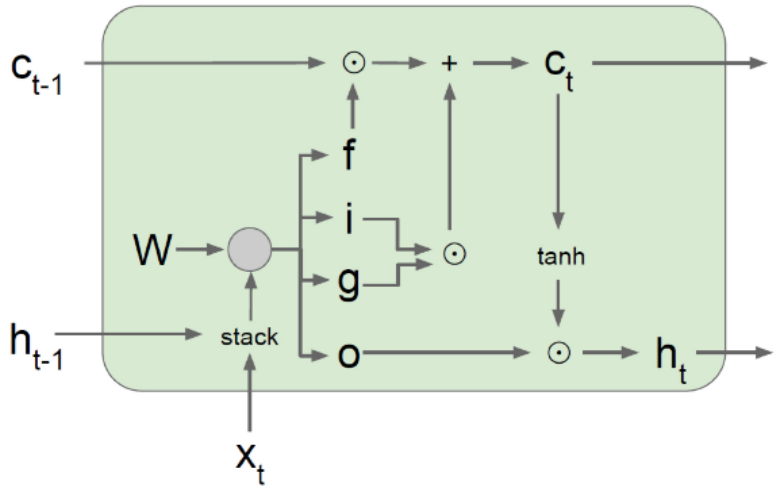
LSTM 為改良版的 RNN。一般 RNN 在處理長序列時會難以訓練，這是因為重複的矩陣乘法導致梯度消失或是梯度爆炸的問題。

LSTM 透過在 RNN 的 update rule 中引入 gate 的概念來解決這樣的問題。

與 RNN 相同，在每個 timestep 先去計算 activation vector $a = W_x x_t + W_h h_{t-1} + b$ 。接著將 a 分成四個向量a_i、a_f、a_o、$a_g \in R^H$ ，並透過 sigmoid(a_i)、sigmoid(a_f)、sigmoid(a_o)分別取得 input gate、forget gate、output gate，以及透過 tanh(a_g)取得 block gate。

最後透過$c_t = f * c_{t-1} + i * g$以及$h_t = o * \tanh(c_t)$來計算下一個 cell state 與 hidden state (*為 elementwise product)。

Computational graph 如下



接著根據以上說明完成單一 timestep 的 forward path，

LSTM.step_forward( )。執行結果如下

```
next_h error:  2.606541143878583e-09
next_c error:  1.7376745523804369e-09
```

可以看到下一個 hidden state 與 cell state 的誤差很小，代表函數功能正確。

八、 LSTM : Forward

完成 LSTM.forward( )來對整個時間序列資料執行 forward propagation。

- 實現方法 ：

  整個時間序列一共有 T 個 timestep，利用 for loop 迭代 T 次，每一次迭代都執行一次剛才定義的 LSTM.step_forward( )。將每個 timestep 所得到的 hidden state stack 在一起，得到最終 hidden state output。

- 執行結果 ：

  ```
  hn error:  2.668523515654886e-09
  ```

  Hidden state output 誤差很小，代表函數功能正確。

九、 LSTM Captioning Model

修改 CaptioningRNN 類別，將其加入 self.cell_type = 'lstm'的功能。

- 實現方法 ：

  這部分需要修改初始化與 forward 方法。

  初始化的部分，output projection、feature projection 和 encoder 接不需要修改，要修改的只有骨幹網路的部分，當 self.cell_type = 'lstm'時，骨幹網路使用 LSTM。

  Forward path 的部分，由於骨幹網路已經修改為 LSTM，因此 forward 函

數也會跟著修改成 LSTM.forward( )。

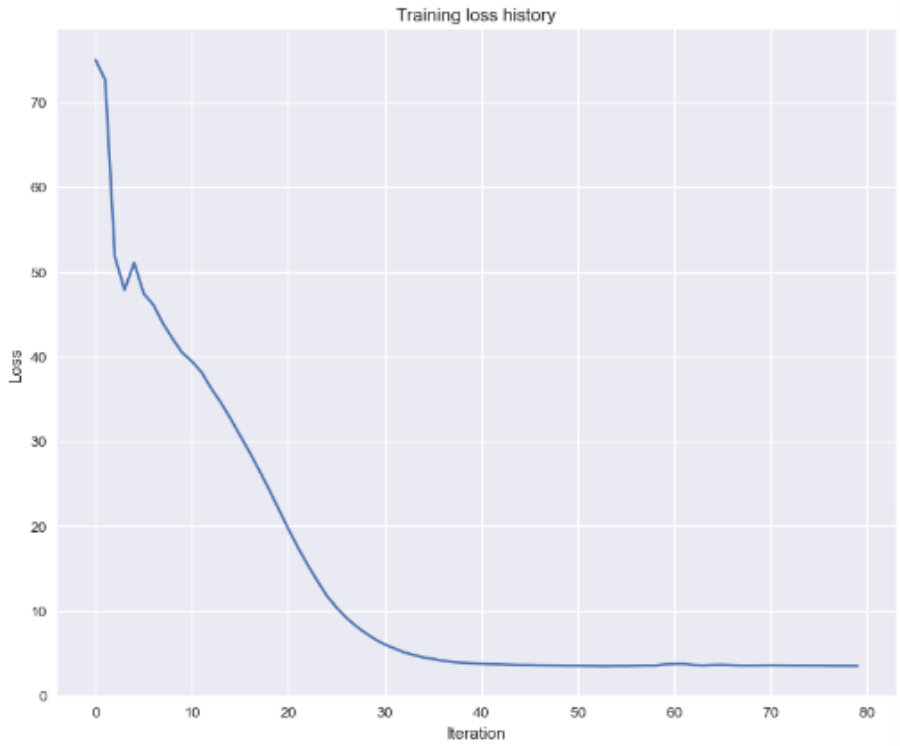
- 執行結果：

```
For input images in NCHW format, shape (2, 3, 224, 224)
Shape of output c5 features: torch.Size([2, 400, 7, 7])
loss:  146.316162109375
expected loss:  146.31614685058594
difference:  5.214321112077035e-08
```

可以看到 loss 的誤差很小，代表說此模型可以正常運作。

十、 Overfit Small Data

使用與在 RNN 時相同的資料集來讓模型 overfit，執行 80 個 epochs 後結果如下



最終的 loss 大約落在 3 左右。

十一、 Caption Sampling

修改 CaptioningRNN.sample 使得他也可以執行 LSTM。

- 實現方法：

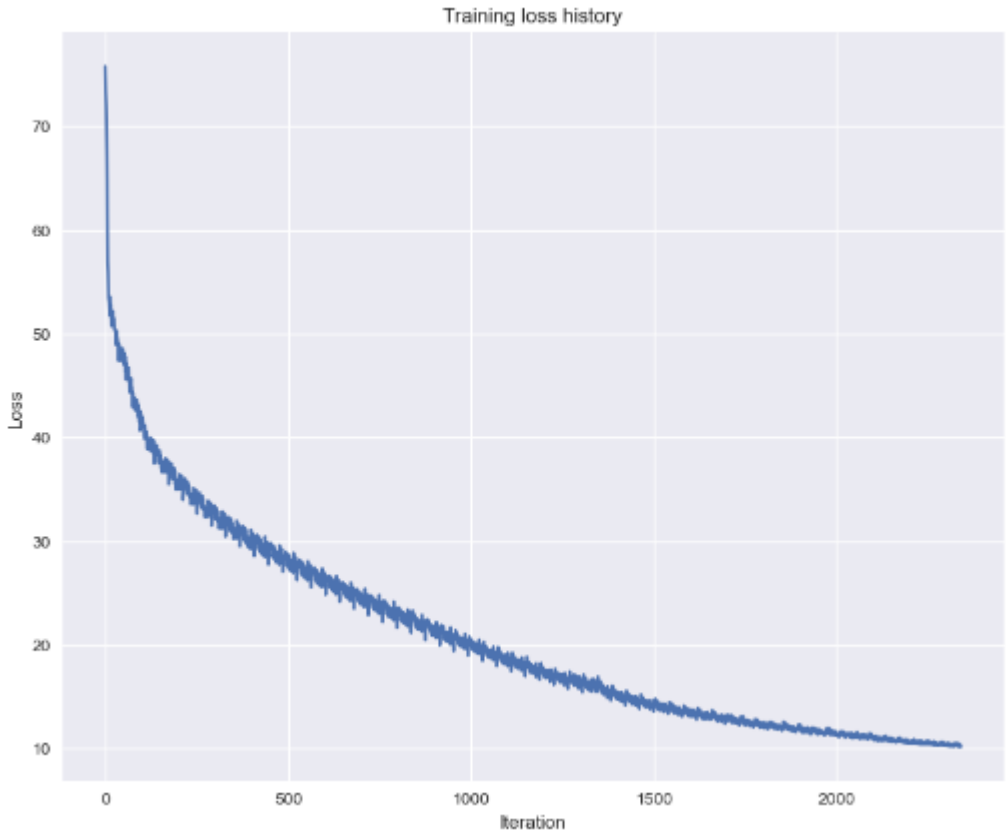
  利用 if-else 來判斷要執行 RNN 還是 LSTM。

  在 LSTM 的情況下，初始化除了要初始 hidden state 之外，還有 cell state 要做初始化。接著利用 for loop 與呼叫骨幹網路的 step_forward 來取得下一個 timestep 的 hidden state 與 cell state。

利用 output projection 來將 hidden state 映射到分數，並取分數最高的字母作為該 timestep 的最終結果。

最後再將每個 timestep 所得到的字母合併成最終結果。

- 執行結果 ：



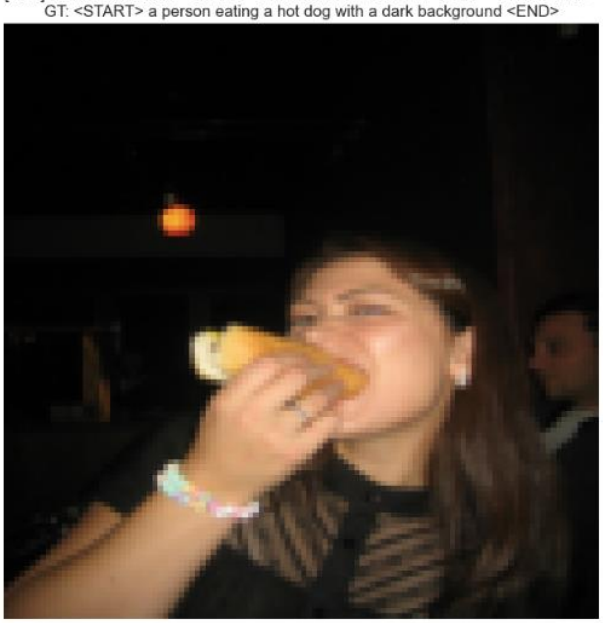
在訓練 60 個 epochs 之後，loss 降到 10 左右。

十二、 Test-time Sampling

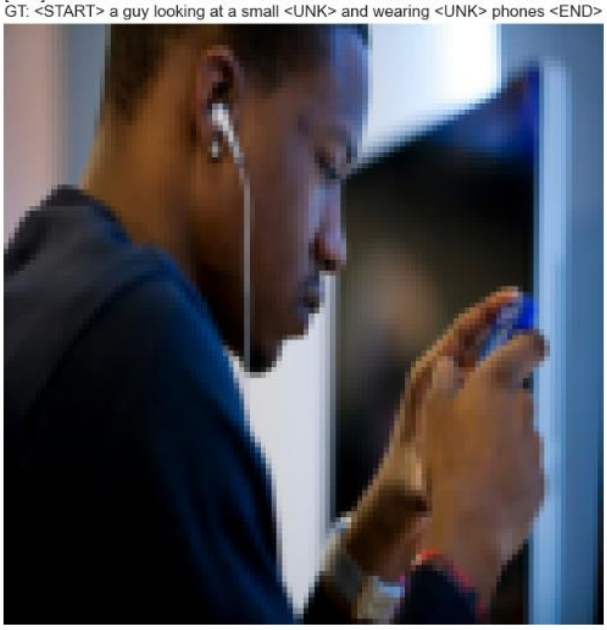
查看訓練完成的模型對 training data 與 validation data 的效果。

- Training data :

[train] LSTM Generated: <START> a man is on a <UNK> in <UNK> <UNK> <END>
GT: <START> a person eating a hot dog with a dark background <END>



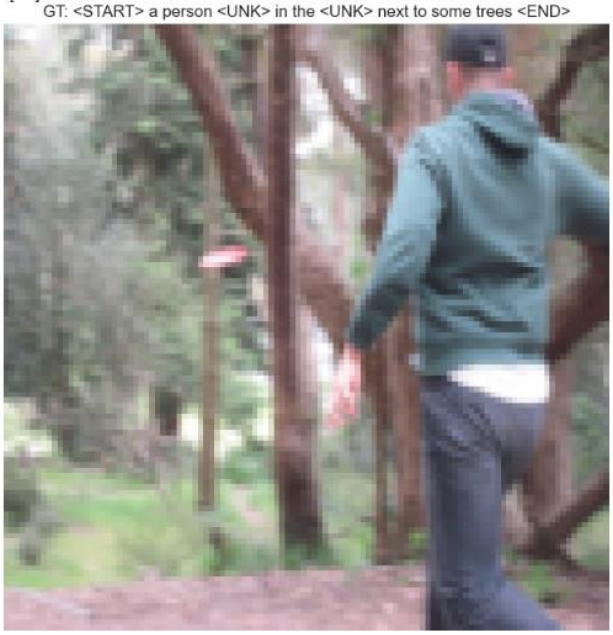
[train] LSTM Generated: <START> a man is on a <UNK> in <UNK> <UNK> <END>
GT: <START> a guy looking at a small <UNK> and wearing <UNK> phones <END>



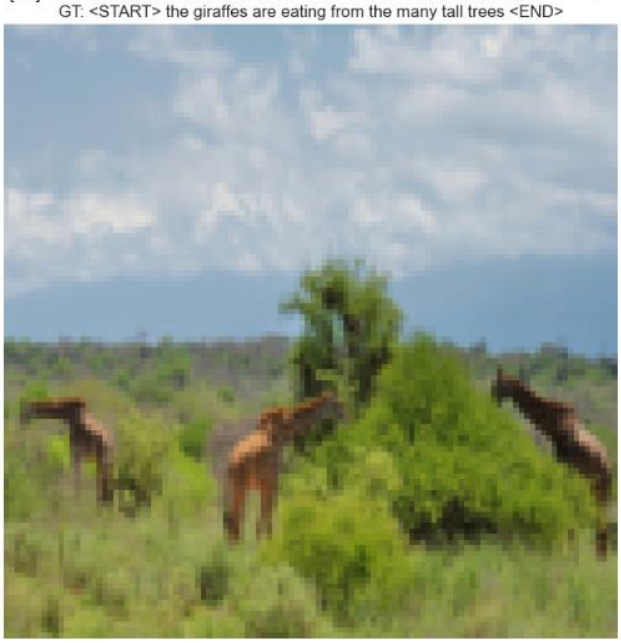
由部份結果可以看到，訓練 60 個 epochs 所得到的模型在 training data 上的採樣效果不太好。

- Validation data :

[val] LSTM Generated: <START> a man is on a <UNK> in <UNK> <UNK> <END>
GT: <START> a person <UNK> in the <UNK> next to some trees <END>



[val] LSTM Generated: <START> a man is on a <UNK> in <UNK> <UNK> <END>
GT: <START> the giraffes are eating from the many tall trees <END>



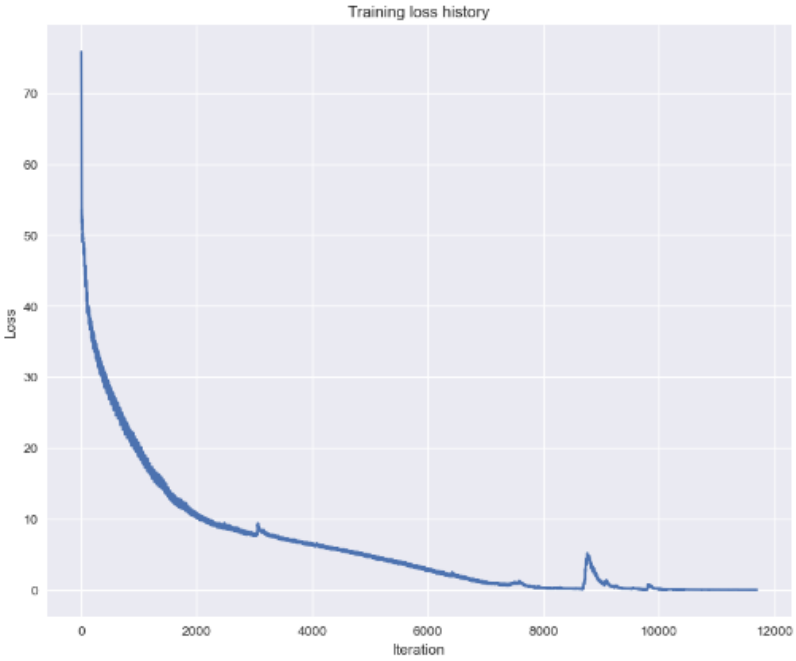
對於 validation data 也一樣，效果不太好。

十三、 額外嘗試

- 重新訓練 LSTM :

  剛才訓練出來的 LSTM 表現不太好，因此我嘗試調整參數，重新訓練 LSTM。

  根據剛才的 loss curve 可以發現到，在訓練完成附近 loss 還有在下降，因此我猜測加大 epochs 數量可以有效提升模型表現。我將 epochs 調整成 300 後重新訓練，新的 loss curve 如下



可以看到經過調整後，loss 可以來到趨近於 0 的程度(實際值為 0.03)。

查看對 training data 以及 validation data 的表現

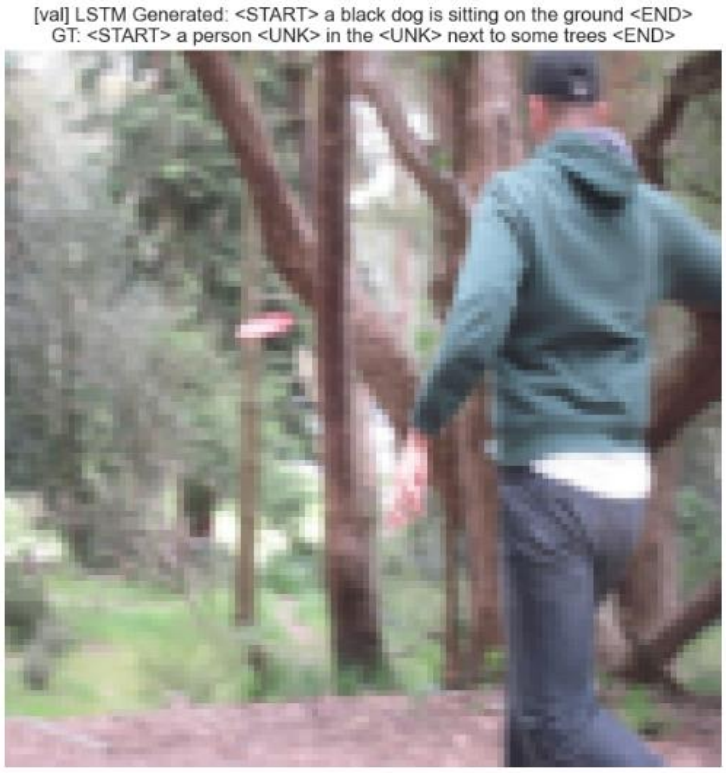
1. Training data :

[train] LSTM Generated: <START> a guy looking at a small <UNK> and wearing <UNK> phones <END>
GT: <START> a guy looking at a small <UNK> and wearing <UNK> phones <END>

由部份結果可以看出，對於 training data 來說，模型可以很準確的採樣。

2. Validation data :

[val] LSTM Generated: <START> a train on a train track <UNK> <UNK> sky background <END>
GT: <START> a couple of blue street signs sitting on the side of a road <END>



對於 validation data 來說，模型表現就比較差了，可能有 overfitting 的狀況發生。

- 時間序列 AI :

  由於這是我第一次接觸時間序列的模型，因此我希望可以多加熟悉相關概念。

  首先先了解神經網路與地回神經網路的差別。

  神經網路中會儲存關於特定問題的許多特徵(權重)。當我們要使用訓練好的神經網路來預測時，只要把一筆新的資料溜進去，神經網路會把資料分解成包含特徵的訊息。這些訊息會在神經網路內部傳遞並刺激神經網路，而神經網路會透過這些刺激來分析訊息所包含特徵，並得到答案。但是這個答案與時間無關，因為訓練好的神經網路每個神經元儲存的權重都已經固定住了，因此相同資料不論丟多少次都會得到相同答案。

  RNN 在這之上做出改進。每個神經元除了儲存特徵之外，還有一個儲存歷史資訊的 hidden state。當我們進行預測時，，每個神經元的權重依然被固定住，但是 hidden state 卻沒有被固定住，讓神經網路可以根據輸入資料的變化調整輸出，得到一個相關的結果。

  RNN 的致命缺點在於很容易梯度消失，主要原因在於歷史資訊過長。

因此發展出 LSTM 與 GRU。這兩種模型都是在 hidden state 上動手腳來解決梯度消失的問題。

LSTM (Long short-term memory)多了 input gate、forget gate、output gate。Input gate 決定當前輸入是否要被記憶，forget gate 決定是否遺忘先前的 hidden state，output state 決定當前所得之輸出要放多少進到 hidden state。這三個 gate 的加入可以讓一些不重要的資訊被忽略，進而讓梯度可以順利進行遠距離傳播。

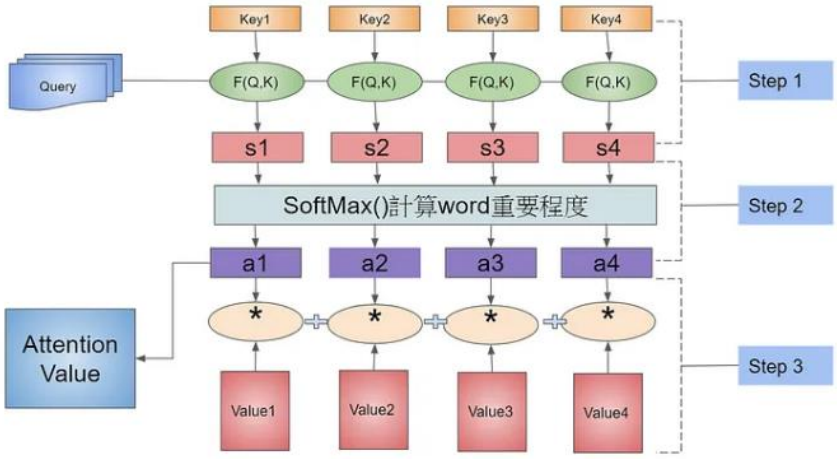
GRU (Gated recurrent unit)有 reset gate 與 update gate。Reset gate 決定是否要遺忘先前的 hidden state，update gate 決定先前的 hidden state 留下來的比例。

- Attention :

  雖然說這次作業沒有實做到 attention 的部分，但是 attention 也是一種優化 RNN 的概念。

  Attention-based model 其實就是一個相似性的度量，當前的輸入與目標狀態越相近，那當前入的權重就會越大，代表說當前的輸出更加依賴當前的輸入。

  attention 架構大致如下



計算 Query 和 key 的相似度，常見的計算相似度的方法如下 ：

1. 求兩者向量 dot product

$$Similarity(Query, Key_i) = Query \cdot Key_i$$

2. 求兩者向量 cosine 相似性

$$Similarity(Query, Key_i) = \frac{Query \cdot Key_i}{\|Query\| \cdot \|Key_i\|}$$

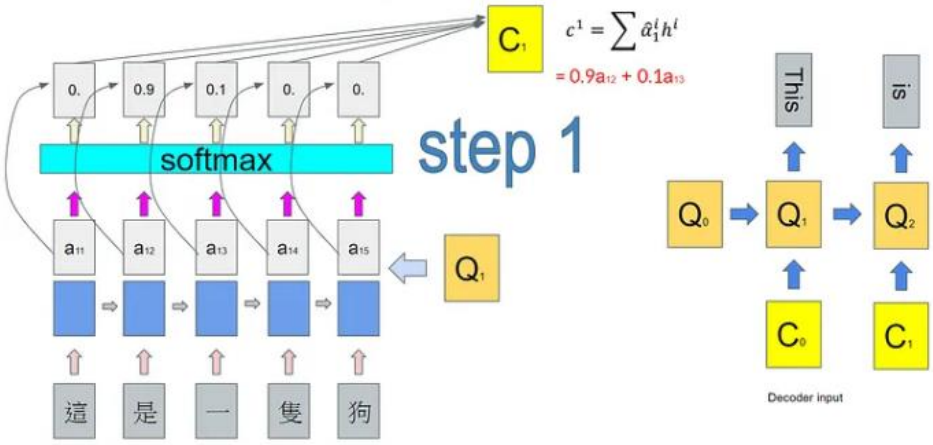
3. 引入額外神經網路求值

$$Similarity(Query, Key_i) = MLP(Query \cdot Key_i)$$

然後將所有對應的相似度與 word value 做相乘後相加，得到 attention

$$Attention(Query, Source) = \sum_{i=1}^{L_x} Similatiry(Query, Key_i) * Value_i$$

引入 attention 的 RNN 架構如下



十四、 Reference

[1] Fredrick Lee “Attention in Text： 注意力機制”
https://medium.com/@fredericklee_73485/attention-in-text-%E6%B3%A8%E6%84%8F%E5%8A%9B%E6%A9%9F%E5%88%B6-bc12e88f6c26

[2] Rice Yang “RNN, LSTM, GRU 之間的原理與差異”
https://u9534056.medium.com/rnn-lstm-gru%E4%B9%8B%E9%96%93%E7%9A%84%E5%8E%9F%E7%90%86%E8%88%87%E5%B7%AE%E7%95%B0-23eba88afa1e

[3] OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model].
https://chat.openai.com/