HW 8 成果分析

資料及狀況

使用 COCO dataset 來進行 image caption 的訓練

```
Train images shape:  torch.Size([10000, 3, 112, 112])
Train caption tokens shape:  torch.Size([10000, 17])
Validation images shape:  torch.Size([500, 3, 112, 112])
Validation caption tokens shape:  torch.Size([500, 17])
total number of caption tokens:  864
```

<START> a man is walking on the sidewalk holding an umbrella <END>

RNN 網路模型

RNN神經網路不只會讀入目前的 x_n 資料，還會考慮到上一個 x_{n-1} 的隱藏層資料。

Step RNN

我們先完成單一 sequence 的程式：

```
# Step forward

next_h = torch.tanh(torch.matmul(x, Wx) + torch.matmul(prev_h, Wh) + b)
cache = (x, prev_h, Wx, Wh, b)
```

```
# Step backward

x, prev_h, Wx, Wh, b = cache
dtanh = (1 - torch.tanh(torch.mm(x, Wx) + torch.mm(prev_h, Wh) + b) ** 2) *
dnext_h
dprev_h = torch.mm(dtanh, Wh.t())
dx = torch.mm(dtanh, Wx.t())
dWx = torch.mm(x.t(), dtanh)
dWh = torch.mm(prev_h.t(), dtanh)
db = torch.sum(dtanh, dim=0)
```

Whole Sequence RNN

跑遍整個序列：

我這邊用 `caches` 存放每次的 cache。

```
# RNN forward

N, T, D = x.shape
_, H = h0.shape
h = torch.zeros(N, T, H, device=x.device, dtype=x.dtype)
caches = []
for t in range(T):
  if t == 0:
    # use clone to aviod error
    h[:, t, :], cache = rnn_step_forward(x[:, t, :].clone(), h0, Wx, Wh, b)
  else:
    h[:, t, :], cache = rnn_step_forward(x[:, t, :].clone(), h[:, t - 1,
:].clone(), Wx, Wh, b)
  caches.append(cache)
cache = caches



# RNN backward

N, T, H = dh.shape
_, D = cache[0][0].shape

for t in reversed(range(T)):
  dx[:, t, :], dprev_h, dWx_tmp, dWh_tmp, db_tmp = rnn_step_backward(dh[:, t,
:].clone() + dprev_h, cache[t])
  dWx += dWx_tmp
  dWh += dWh_tmp
  db += db_tmp
  if t == 0:
    dh0 = dprev_h
```

RNN with Module API

我們也可以使用 tensor 的 API 來建立 RNN 模型

```
# in class RNN(nn.Module)

# def __init__

# Initialize parameters
self.Wx = Parameter(torch.randn(input_size, hidden_size,
                  device=device, dtype=dtype).div(math.sqrt(input_size)))
self.Wh = Parameter(torch.randn(hidden_size, hidden_size,
                  device=device, dtype=dtype).div(math.sqrt(hidden_size)))
self.b = Parameter(torch.zeros(hidden_size,
                  device=device, dtype=dtype))

# def forward(self, x, h0)

hn, _ = rnn_forward(x, h0, self.Wx, self.Wh, self.b)
return hn

# def step_forward(self, x, prev_h)

next_h, _ = rnn_step_forward(x, prev_h, self.Wx, self.Wh, self.b)
return next_h
```

因為 tensor 會幫我們執行 autograd，所以不需要自己寫。

Image Feature Extraction

使用 MobileNet v2 模型來提取影像特徵。

提取後的資料當成 h0 輸入到 RNN 當中。

Word Embedding

把文字進行編碼(類似 One-hot encoding)方便類神經網路的運算。

```
# in class WordEmbedding

# self.W_embed = Parameter(torch.randn(vocab_size, embed_size, device=device,
dtype=dtype).div(math.sqrt(vocab_size)))

def forward(self, x):
    out = self.W_embed[x]
    return out


def temporal_softmax_loss(x, y, ignore_index=None):
    N, _, _ = x.shape
    loss = F.cross_entropy(x.transpose(2, 1), y, ignore_index=ignore_index,
reduction='sum') / N
    return loss
```

Temporal Affine layer

我們還需要把隱藏層轉換成一個輸出，使用 `nn.Linear` 來實現。

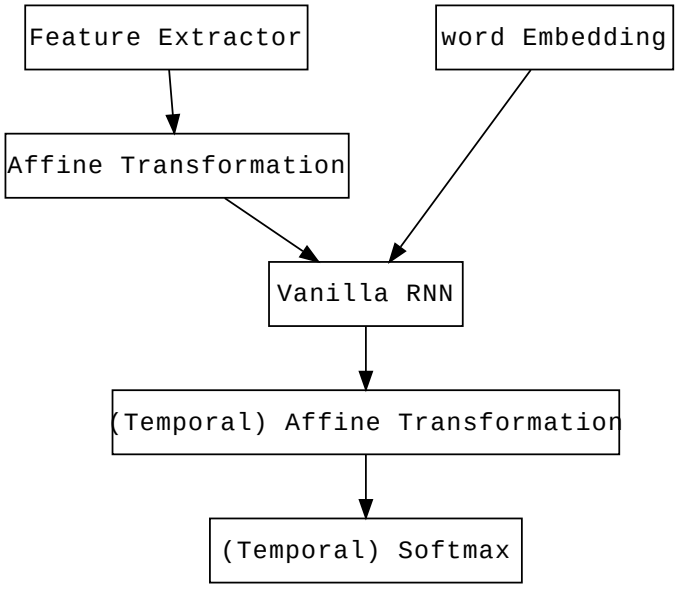
Temporal Softmax loss

最後，設計一個 Loss function 來計算模型的 Loss，跑遍每個 timestep 計算 cross entropy，並且針對 N 平均

```
N, _, _ = x.shape
loss = F.cross_entropy(x.transpose(2, 1), y, ignore_index=ignore_index,
reduction='sum') / N
```

Captioning RNN

把剛剛的所有元件都串起來



```
# Forward

features = self.featureExtractor.extract_mobilenet_feature(images)
h0 = self.featureProjector(features)
embedded_words = self.wordEmbedding(captions_in)
out = self.rnnNetwork(embedded_words, h0)
scores = self.outProjector(out)
loss = temporal_softmax_loss(scores, captions_out, self.ignore_index)
```

開始讓模型 testing。
基本上和 forward 差不多，只是要把每個 timestep 的結果紀錄下來，並且判斷如果 <END> 就停止生成。
要注意的是，因為是數個資料在同一個 minibatch 當中做 testing，每張照片的 <END> 時機都不一樣，所以使用 `end_mask` 紀錄，整個迴圈跑完後再來處理。

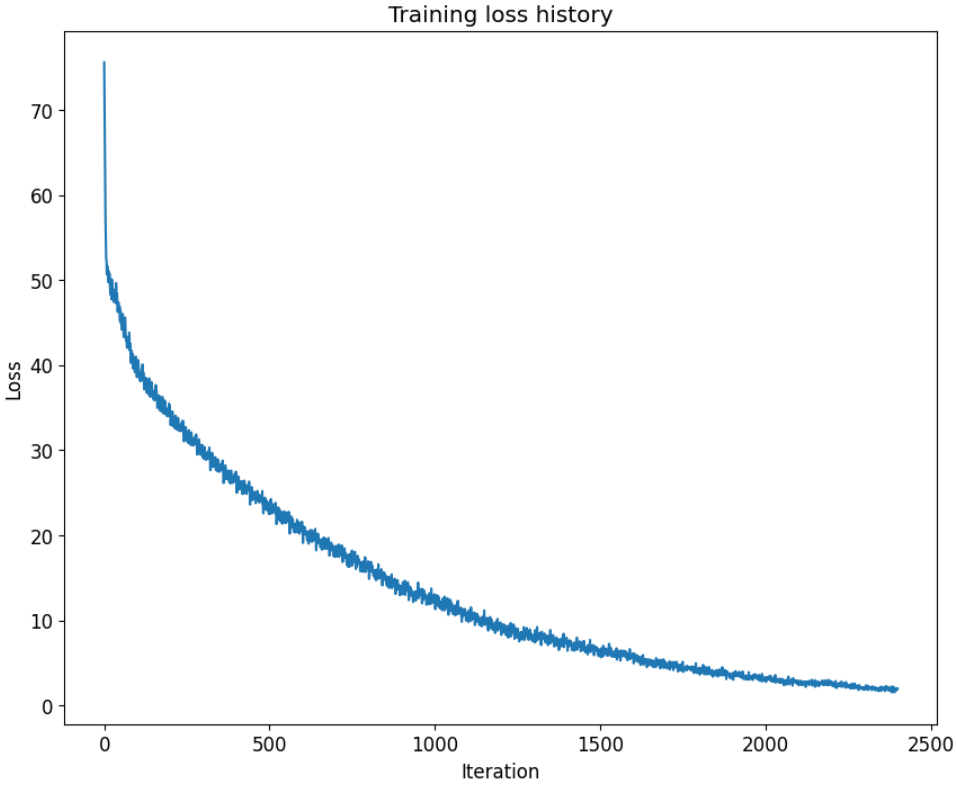
```
# Sample

features = self.featureExtractor.extract_mobilenet_feature(images)
device = features.device
captions = captions.to(device=device)
h = self.featureProjector(features)

word = self._start # <START>
end_mask = torch.full([N], True, device=device)

# Generating captions
for ts in range(max_length):

  x = self.wordEmbedding(word)
  # Apply the RNN forward step. Output is the next hidden state (h).
  h = self.rnnNetwork.step_forward(x, h)
  y = self.outProjector(h)
  word = torch.argmax(y, axis=1)

  # masking
  mask = word == self._end
  # notend = (mask != notend) & (mask == False)
  end_mask[mask] = False
  if not end_mask.any():
    break

  captions[end_mask, ts] = word[end_mask]
```

成果分析

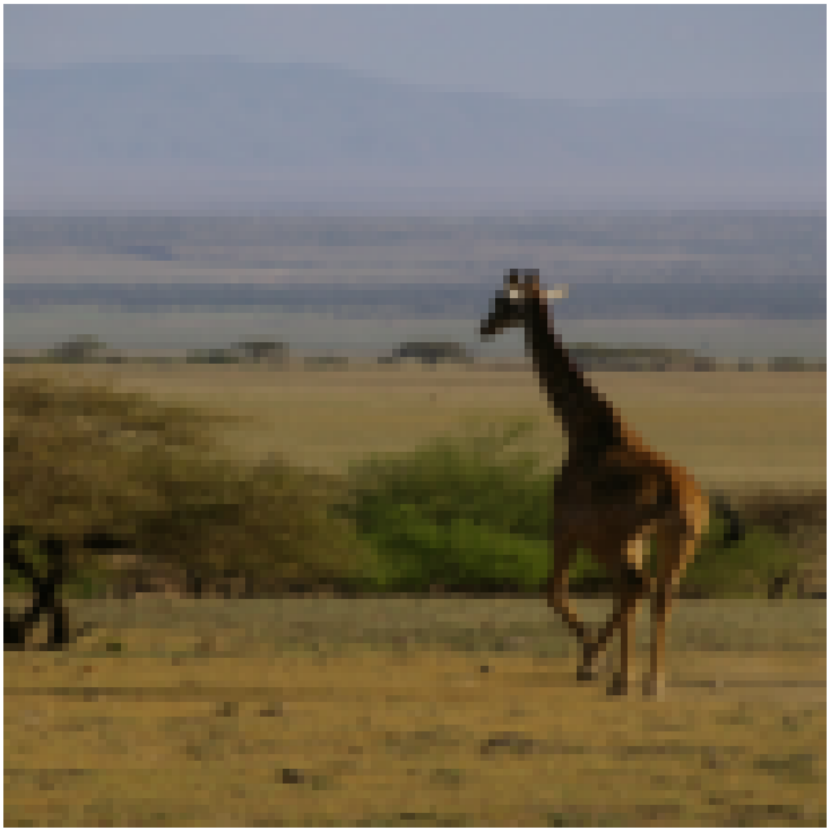
用以下參數進行訓練：

```
num_epochs = 60
batch_size = 250
input_dim = 1280, # hard-coded, do not modify
hidden_dim = 512,
wordvec_dim = 256,
learning_rate = 1e-3
```

得到最後的 training loss = 1.9514。
以 testing 的結果來看，驗證資料生成出來的 captions 似乎有些偏離主題。



train
RNN Generated:a giraffe is off running in a field
GT:<START> a giraffe is off running in a field <END>

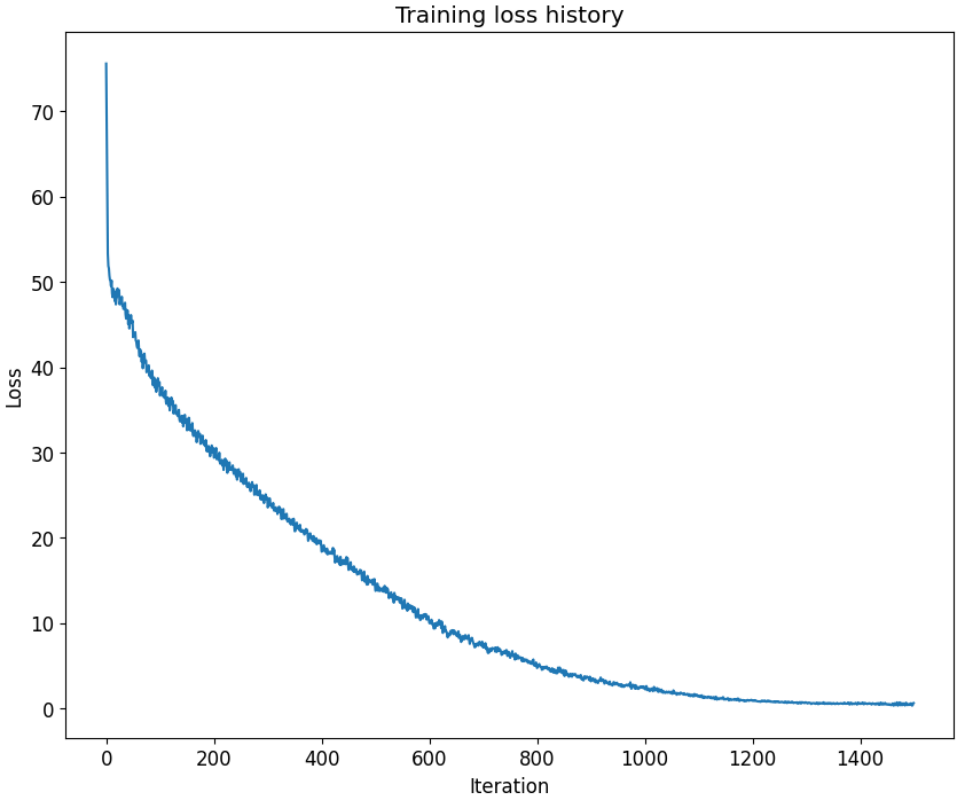


val
RNN Generated:a white train car is parked in the grass
GT:<START> a yellow train moving down the railroad track <END>

嘗試更改模型和訓練方式：
增加隱藏層數量和 batch size

```
num_epochs = 60
batch_size = 400
input_dim=1280, # hard-coded, do not modify
hidden_dim=1024,
wordvec_dim=256,
learning_rate = 1e-3
```

最後得到 training loss = 0.6283，且因為調高了 batch size，所以曲線看起來較平滑。
此外，以這組參數下所訓練出來的結果，有看起來稍微合理，一些了
註：兩個模型皆以 batch size = 10 進行 testing，且模型的好壞是依據個人的主觀判斷來評論。上一個模型在10個 validation 的圖片中，綜合表現相對較差。



val

RNN Generated:a white plate topped with two slices of pizza

GT:<START> a bunch of food is laying in a blue tray <END>

val
RNN Generated:a person riding a snowboard on the beach with a ski board
GT:<START> a <UNK> skier posing at a snowy mountain <END>