HW11 Image captioning with Attention LSTM

M113040105 劉東霖

壹.使用到的 function:

一. dot_product_attention:

1. 為了做矩陣相乘，我先把 A 的 shape 從(N,H,4,4)變成(N,H,16)，prev_h 的 shape 從(N,H)變成(N,1,H)。

2. 如下圖的公式，利用 batch 矩陣相乘得出 attention scores，並把 attention scores 經過 softmax 變成 attention weight。

$$M_{attn}^{t} = h_{t-1}A/\sqrt{H} \in \mathbb{R}^{4\times4}$$

3. 如下圖的公式，利用 batch 矩陣相乘得出 attention。為了能夠矩陣相乘，attention weight 需 reshape 成(N,16,1)。

$$x_{attn}^{t} = \tilde{A}\tilde{M}^{t}_{attn} \in \mathbb{R}^{H}$$

4. 把 attention 和 attention weight reshape 成題目要的形式。Attention reshape 成(N,H)，attention weight reshape 成(N,4,4)。

程式碼如下:

```
A=A.reshape(N,H,-1)
prev_h=prev_h.reshape(N,1,H)
scores=torch.bmm(prev_h,A)/H**0.5
attn_weights=torch.softmax(scores,dim=-1)
attn=torch.bmm(A,attn_weights.reshape(N,D_a**2,1))
attn=attn.reshape(N,H)
attn_weights=attn_weights.reshape(N,D_a,D_a)
```

二. AttentionLSTM step_forward:

跟 LSTM step forward 的差別只有在計算 a 時多加一項而已。

1. 計算當前時間的輸出 a，a 的定義如下圖:

$$a = W_x x_t + W_h h_{t-1} + W_{attn} x_{attn}^{t} + b.$$

2. 把 a 分成 4 等分，第一份為 ai，第 2 份為 af，第 3 份為 ao，第 4 份為 ag。再來把這 4 等份根據下圖的方式帶入不同的 activation function。

$$i = \sigma(a_i) \qquad f = \sigma(a_f) \qquad o = \sigma(a_o) \qquad g = \tanh(a_g)$$

3. 最後在根據下圖的公式來計算下一個 cell state 和 hidden state。

$$c_t = f \odot c_{t-1} + i \odot g \qquad h_t = o \odot \tanh(c_t)$$

where $\odot$ is the elementwise product of vectors.

完整程式碼如下:

```
a=prev_h @ self.Wh+x @ self.Wx+self.b +attn @ self.Wattn
h=a.shape[1]//4
ai,af,ao,ag=a[:,:h],a[:,h:2*h],a[:,2*h:3*h],a[:,3*h:]
i,f,o,g=torch.sigmoid(ai),torch.sigmoid(af),torch.sigmoid(ao),torch.tanh(ag)
next_c=f*prev_c+i*g
next_h=o*torch.tanh(next_c)
```

三. AttentionLSTM forward:

1. 首先，創建一個形狀為（N,T,H）的張量 hn，並將其初始化為全零，數據類型和設備都與輸入張量 x 相同，並設定第一個狀態 prev_h 為 h0。

2. 將 x 進行轉置，讓 T 變成第一維度，以便我們可以沿著時間計算每個隱藏層的狀態，並將初始 cell state 設為 c0

3. 沿著每一個時間，先代入 dot_product_attention 算出 attention，再代入 step_forward 得出每一個隱藏層狀態 ht 和 cell state c，並把每一個 hidden state 用 hn 儲存起來，並更新 prev_h 為 ht。

完整程式碼如下:

```
T=x.shape[1]
N,H=h0.shape
hn=torch.zeros((N,T,H),dtype=x.dtype,device=x.device)
c=c0
prev_h=h0
x=x.permute(1,0,2)
for i in range(T):
    attn,_=dot_product_attention(prev_h,A)
    ht,c=self.step_forward(x[i],prev_h,c,attn)
    prev_h=ht
    hn[:,i,:]=ht
```

四. CaptioningRNN __init__:

這裡我只會提到與 HW10 不同的地方。

如下圖所示，假如 cell type 為 attn，就定義模型為 AttentionLSTM。輸入為 word_vec，輸出為 hidden_dim。

```
self.rnn=AttentionLSTM(wordvec_dim,hidden_dim)
```

因為 attention lstm 使用 spatial attention 機制，不需要將 CNN 的輸出 feature 進行形狀變換，所以只要將 feature 透過 feature_projection 把每個位置映射到 hidden_dim 即可。程式碼如下:

```
self.feature_projection=nn.Linear(input_dim,  hidden_dim)
```

五. CaptioningRNN forward:

這裡我只會提到與 HW10 不同的地方。
先把 feature 從(N,H,4,4) permute 成(N,4,4,H)才能進入 feature_projection。投影完後，再把結果 permute 成(N, H, 4, 4)得到 attention 矩陣 A。
最後把 A 和輸入 x 代入 AttentionLSTM 裡得到隱藏層的輸出。

```
A=self.feature_projection(features.permute(0,2,3,1)).permute(0,3,1,2)
h=self.rnn(x,A)
```

六. CaptioningRNN sample:

這裡我只會提到與 HW10 不同的地方。
首先得到 attention matrix A，跟 forward 使用的方法一樣。再來初始化 hidden state h 和 cell state c 為 A 在第 2 和第 3 維度(影像的高和寬)上的平均值。

```
A=self.feature_projection(features.permute(0,2,3,1)).permute(0,3,1,2)
h=A.mean(dim=(2,3))
c=A.mean(dim=(2,3))
```

在 for 迴圈進行生成字幕的過程，先用 dot_product_attention 算出 attention 和 attention weight，並把每個 attention weight 儲存起來，再代入寫的 step_forward 得出每一個隱藏層狀態 ht 和 cell state c，直到生成最大字幕為止。

```
for i in range(max_length):
    w=self.embed(words).reshape(N,-1)
    if self.cell_type == 'rnn':
        h=self.rnn.step_forward(w,h)
    elif self.cell_type == 'lstm':
        h,c=self.rnn.step_forward(w,h,c)
    else:
        attn,attn_weights_all[:,i,:,:]=dot_product_attention(h,A)
        h,c=self.rnn.step_forward(w,h,c,attn)
    scores=self.fc(h)
    words=torch.argmax(scores,dim=1)
    captions[:,i]=words
```

貳 程式執行結果:

1. 使用 dot_product_attention 算出 attention 和 attention weight 與預期差異很小。

```
attn error:  1.4410324402829568e-09
attn_weights error:  3.5290517247691244e-08
```

2. 使用 Attention LSTM step forward 算出 next_h 和 next_c 與預期差異很小。

```
next_h error:  2.425617005126452e-09
next_c error:  1.2938551985068886e-09
```

3. 使用 Attention LSTM forward 算出 h 與預期差異很小。

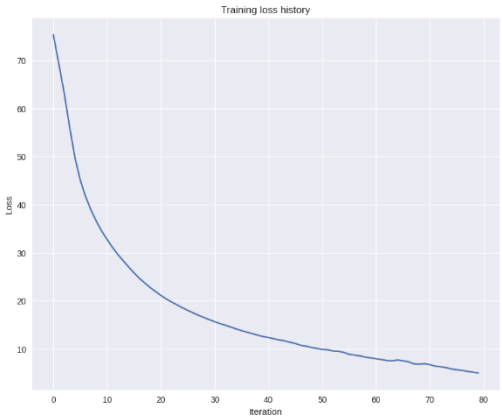
```
h error:  2.487302938381543e-09
```

4. 使用 captioning model 的 forward 算出 loss 與預期差異很小。

```
For input images in NCHW format, shape (2, 3, 224, 224)
Shape of output c5 features: torch.Size([2, 400, 7, 7])
loss:  8.015640258789062
expected loss:  8.015639305114746
difference:  5.9488343140184995e-08
```

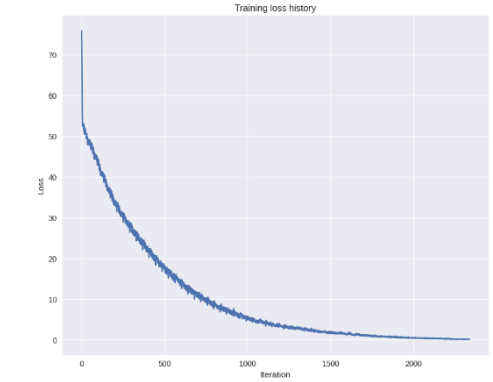
5. 使用 train_captioner 來訓練 CaptioningRNN，cell type 為 attn，且只有使用 50 個 example 來訓練。在 learning rate 為 1e-3 的情況下，發現 loss 小於 9。

```
(Epoch 69 / 80) loss: 6.9239 time per epoch: 0.1s
(Epoch 70 / 80) loss: 6.7375 time per epoch: 0.1s
(Epoch 71 / 80) loss: 6.4052 time per epoch: 0.1s
(Epoch 72 / 80) loss: 6.3095 time per epoch: 0.1s
(Epoch 73 / 80) loss: 6.1161 time per epoch: 0.1s
(Epoch 74 / 80) loss: 5.8436 time per epoch: 0.1s
(Epoch 75 / 80) loss: 5.6655 time per epoch: 0.1s
(Epoch 76 / 80) loss: 5.5344 time per epoch: 0.1s
(Epoch 77 / 80) loss: 5.3345 time per epoch: 0.1s
(Epoch 78 / 80) loss: 5.1536 time per epoch: 0.1s
(Epoch 79 / 80) loss: 5.0076 time per epoch: 0.1s
```

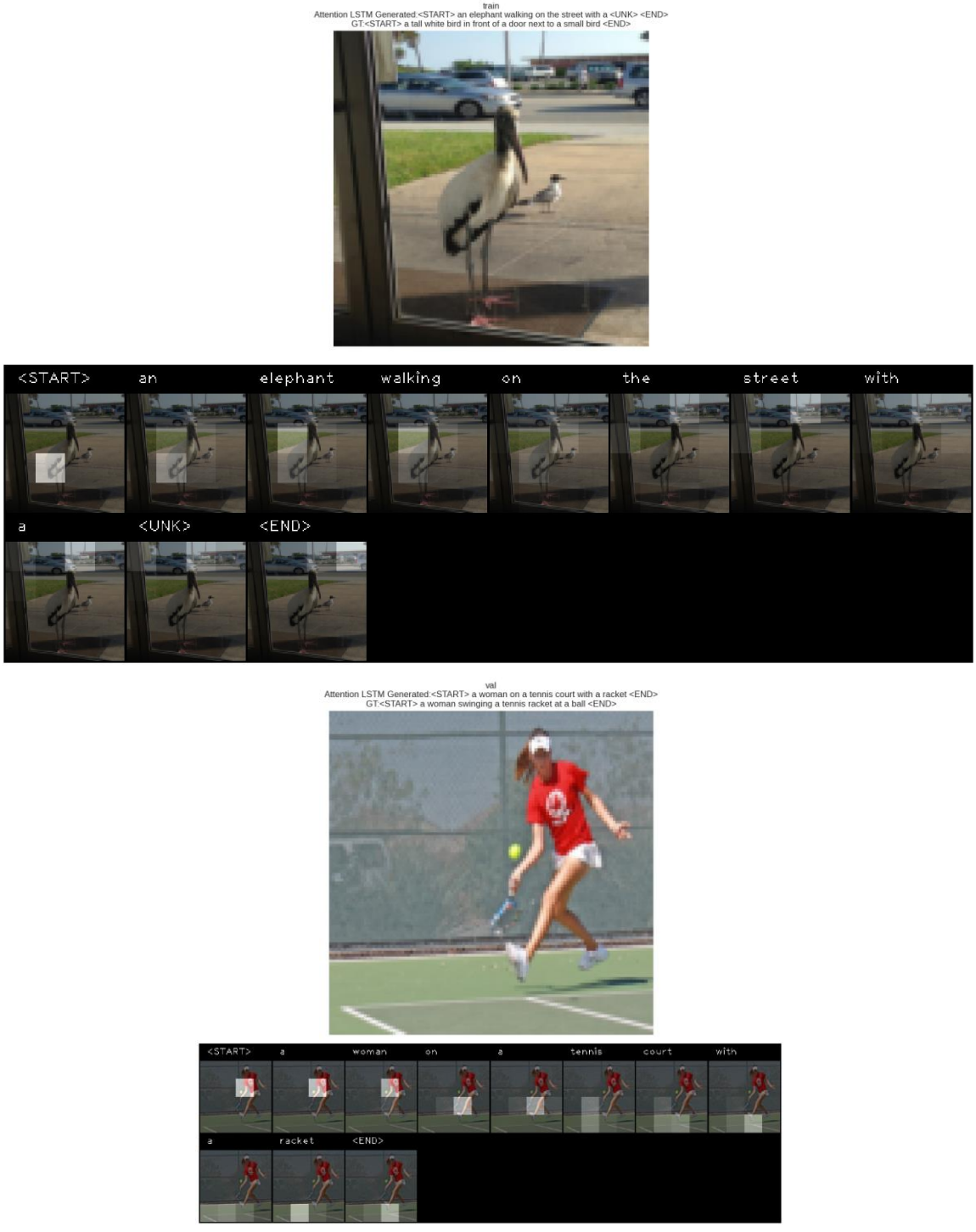


6. 使用 train_captioner 來訓練 CaptioningRNN，cell type 為 attn。在 learning rate 為 1e-3 和 epoch=60 的情況下，發現 loss 小於 0.5。

```
(Epoch 50 / 60) loss: 0.4959 time per epoch: 13.9s
(Epoch 51 / 60) loss: 0.4633 time per epoch: 13.9s
(Epoch 52 / 60) loss: 0.3525 time per epoch: 13.9s
(Epoch 53 / 60) loss: 0.3301 time per epoch: 13.8s
(Epoch 54 / 60) loss: 0.2460 time per epoch: 13.9s
(Epoch 55 / 60) loss: 0.2049 time per epoch: 13.9s
(Epoch 56 / 60) loss: 0.1610 time per epoch: 13.9s
(Epoch 57 / 60) loss: 0.1331 time per epoch: 13.9s
(Epoch 58 / 60) loss: 0.1159 time per epoch: 13.9s
(Epoch 59 / 60) loss: 0.1048 time per epoch: 13.9s
```



7. captions 出來的結果，可發現字串與圖片的特徵越相關，圖片中的某個位置越亮。



train
Attention LSTM Generated:<START> a red <UNK> bus parked next to a sidewalk <END>
GT:<START> a red <UNK> bus parked next to a sidewalk <END>

val
Attention LSTM Generated:<START> a <UNK> computer with a <UNK> computer with a laptop <END>
GT:<START> a man holding a modern cell phone at a desk <END>