深度視覺 HW8

M113040064 李冠宏

Part I. Preparation

這邊是做一些事前的資料處理，因此不再贅述

Part II. Recurrent Neural Network

首先是 rnn_step_forward 的實作，這個步驟首先要將循環神經網路的 forward 拆成一個 timestamp 的小部分，首先先將 input 和他的權重做相乘，且把上一個 timestamp 的 hidden state 和他的權重做相乘，然後加上 bias。最後再經過 tanh activation function 輸出。且把這些參數都存到 cache 中。

```
# 計算單次的 hidden state
a = torch.matmul(x, Wx) + torch.matmul(prev_h, Wh) + b
next_h = torch.tanh(a)

# 將結果存到cache
cache = (x, prev_h, Wx, Wh, a)
```

測試結果：

```
next_h error:  2.3200594408551194e-09
```

接下來是 rnn_step_backward 的實作，這部分也是將 backward 拆成一個 timestamp 的小部分，首先先把 forward 的 cache 資料抓下來，然後計算 activation function tanh 的梯度，接下來根據 chain rule 把每個參數的梯度計算出來，最後回傳這些值。

```
x, prev_h, Wx, Wh, a = cache

# 計算 activation function tanh 的梯度
dtanh = dnext_h * (1 - torch.tanh(a) * torch.tanh(a))

# 計算各個參數的梯度
dx = torch.matmul(dtanh, torch.t(Wx))
dprev_h = torch.matmul(dtanh, torch.t(Wh))
dWx = torch.matmul(torch.t(x), dtanh)
dWh = torch.matmul(torch.t(prev_h), dtanh)
db = torch.sum(dtanh, dim=0)
```

測試結果：

```
dx error:  3.530076340711781e-10
dprev_h error:  1.0233331691183l2e-09
dWx error:  1.3202846102464692e-10
dWh error:  3.0029385298691 91e-10
db error:  1.33164624598737 77e-10
```

再來是 rnn_forward 的實作，這邊主要是呼叫剛剛已經定義好的 rnn_step_forward 來做完整的循環神經網路，首先會先初始化 h(最後要回傳的值)，將現在的 h 做初始化，並且初始化 cache 以供待會 backward 使用。接下來用一個迴圈去跑所有的 timestamp，在每一個回合中做 rnn_step_forward，並且儲存結果，並將參數存入 cache 內。

```
N, T, D = x.shape
_, H = h0.shape

# 初始化h
h = torch.zeros(N, T, H, dtype=x.dtype, device=x.device)

# 初始化 current hidden state (變成 initial state)
curr_h = h0

# 初始化 cache
cache = []

for t in range(T):
    # 提取現在time step 的 input
    xt = x[:, t, :]

    # forward pass (參數共享)
    curr_h, cache_t = rnn_step_forward(xt, curr_h, Wx, Wh, b)

    # 儲存結果
    h[:, t, :] = curr_h

    # 存 cache
    cache.append(cache_t)
```

測試結果：

```
h error:  4.242275290213816e-09
```

最後是 rnn_backward 的實作，首先將 shape 定義好，並且把梯度做初始化，再來是將迴圈反轉以做 backward，我們會先計算加入前一個 timestamp 的梯度，接下來呼叫剛剛定義好的 rnn_step_backward 來算出各個參數的梯度，最後累加梯度並將這些值回傳出去。

```python
N, T, H = dh.shape
x, h0, Wx, Wh, b = cache[0]
D = x.shape[1]

# 初始化梯度
dx = torch.zeros(N, T, D, dtype=x.dtype, device=x.device)
dh0 = torch.zeros(N, H, dtype=x.dtype, device=x.device)
dWx = torch.zeros(D, H, dtype=x.dtype, device=x.device)
dWh = torch.zeros(H, H, dtype=x.dtype, device=x.device)
db = torch.zeros(H, dtype=x.dtype, device=x.device)

# 初始化梯度
dprev_h = torch.zeros(N, H, dtype=x.dtype, device=x.device)

for t in reversed(range(T)):
    # 計算總梯度
    dh_total = dh[:, t, :] + dprev_h

    # backward pass
    dx_t, dprev_h, dWx_t, dWh_t, db_t = rnn_step_backward(dh_total, cache[t])

    # 累加梯度
    dx[:, t, :] += dx_t
    dWx += dWx_t
    dWh += dWh_t
    db += db_t

dh0 = dprev_h
```

測試結果：

```
dx error:  1.2719319196887 28e-09
dh0 error:  8.638061857982 29e-10
dWx error:  5.161722983399557e-10
dWh error:  7.042436908041916e-10
db error:  9.360267729270615e-10
```

再來是 pytorch 裡面的 autograd 功能，使用這個功能我們可以自動的做梯度下降，在這邊我們會將做 autograd 的結果和我們的比較，結果如下：

```
dx error:  9.703371940851615e-17
dh0 error:  9.134424184301861e-17
dWx error:  1.2423045378039655e-16
dWh error:  4.1301393536898844e-17
db error:  1.9902473375401266e-16
```

Part III. RNN for image captioning

在這個部分我們將會來實作用 RNN 來產生圖片的說明文字
首先我們會先建立一個特徵提取器，這部分是已經定義好的，以下為其參數

```
Total params: 2,223,872
Trainable params: 2,223,872
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.14
Forward/backward pass size (MB): 38.94
Params size (MB): 8.48
Estimated Total Size (MB): 47.57
```

再來是 word embedding，這邊我們會將不同的字詞做編碼，而方式相對簡單，只要透過 pytorch 的陣列索引存取建構式定義好的 self.W_embed 即可產生正確的輸出

```
def forward(self, x):
    out = None
    ##############################################################################
    # TODO: Implement the forward pass for word embeddings.                      #
    #                                                                            #
    # HINT: This can be done in one line using PyTorch's array indexing.         #
    ##############################################################################
    # Replace "pass" statement with your code

    out = self.W_embed[x]

    ##############################################################################
    #                               END OF YOUR CODE                             #
    ##############################################################################
    return out
```

執行結果：

```
out error:  2.7272727753724473e-09
```

接下來是 temporal affine layer，在這邊會將每個 timestamp 每個字的 RNN hidden vector 轉換成一個分數，這邊會使用 nn.Linear 來實作

執行結果：

```
affine layer - input shape: torch.Size([2, 4]), output shape: torch.Size([2, 3])
dx error:  6.021897849884195e-09

temporal affine layer - input shape: torch.Size([2, 3, 4]), output shape: torch.Size([2, 3, 3])
dx error:  6.039603964764421e-09
```

接下來是 temporal softmax loss 的實作，在每一個 timestamp 我們會產生分數，而我們會用 softmax lose function 來計算其 loss，實作如下：

```
# 計算 cross-entropy loss
loss = nn.functional.cross_entropy(
    torch.transpose(x, 1, 2), y,
    ignore_index=ignore_index, reduction='sum') / x.shape[0]
```

執行結果：

```
def check_loss(N, T, V, p):
    x = 0.001 * torch.randn(N, T, V, **to_double_cuda)
    y = torch.randint(V, size=(N, T), **to_long_cuda)
    mask = torch.rand(N, T, **to_double_cuda)
    y[mask > p] = 0
    # YOUR_TURN: Impelement temporal_softmax_loss
    print(temporal_softmax_loss(x, y, NULL_index).item())

check_loss(1000, 1, 10, 1.0)   # Should be about 2.00-2.11
check_loss(1000, 10, 10, 1.0)  # Should be about 20.6-21.0
check_loss(5000, 10, 10, 0.1) # Should be about 2.00-2.11
```

```
2.0815791002459574
20.79024042178467
2.103639258712438
```

最後是要實作 CaptioningRNN 類別，我們將會整合剛剛所有做的東西到這個類別，以實作用 RNN 來生成圖片的說明文字，首先我們要先定義建構式，將一些 attribute 定義好：

```
# 特徵提取
self.featureExtractor = FeatureExtractor(pooling=True, device=device, dtype=dtype)

# feature projector
self.featureProjector = nn.Linear(1280, hidden_dim).to(device, dtype)

# word embedding
self.wordEmbedding = WordEmbedding(vocab_size, wordvec_dim, device=device, dtype=dtype)

# RNN
self.coreNetwork = RNN(wordvec_dim, hidden_dim, device=device, dtype=dtype)

# out projector
self.outProjector = nn.Linear(hidden_dim, vocab_size).to(device, dtype)
```

接下來定義 forward method，主要是用建構式定義好的 attributes 來進行操作，最後算出 loss 並回傳

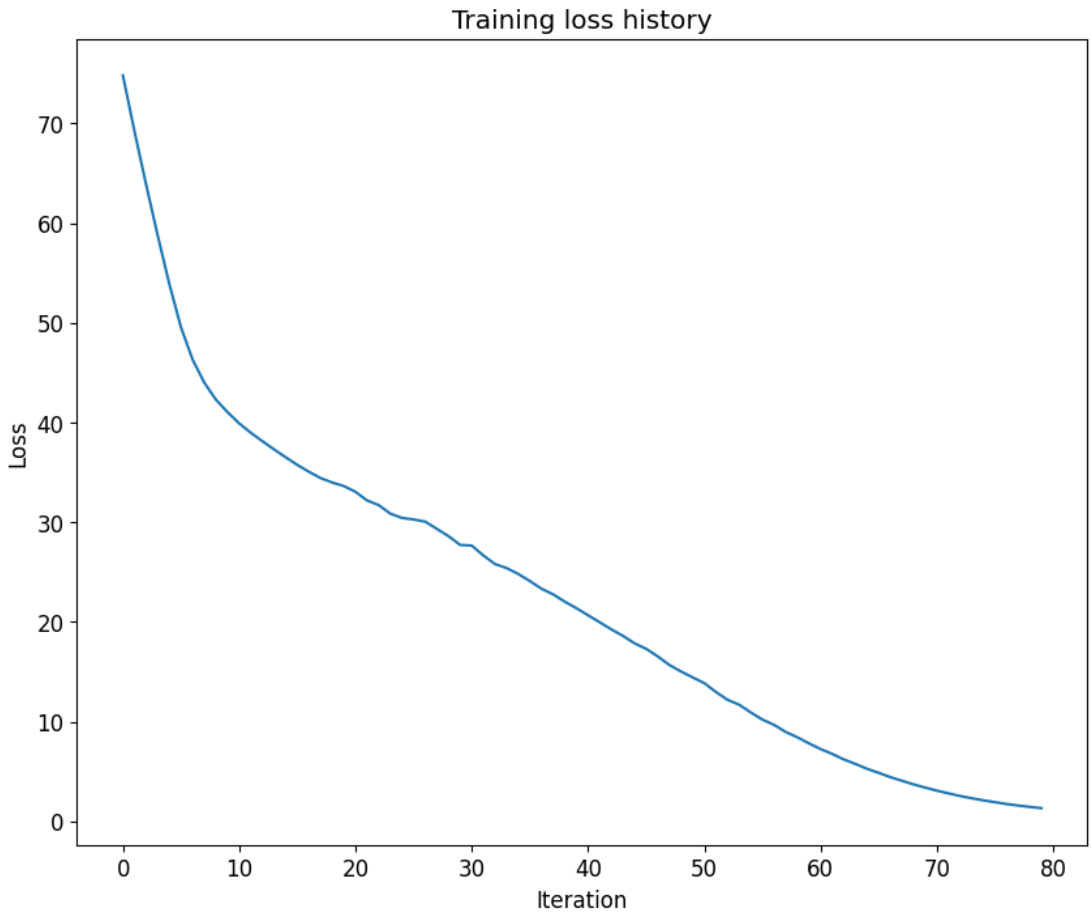
```
# 從圖片中抓特徵
features = self.featureExtractor.extract_mobilenet_feature(images)

# Step (1): 轉換
h0_A = self.featureProjector(features)

# Step (2): 產生word embedded
embed_words = self.wordEmbedding(captions_in)

# Step (3): 利用 RNN 產生 embed words 的序列並且產生 hidden state vectors
hstates = self.coreNetwork(embed_words, h0_A)

# Step (4): 用 temporal 轉換來計算單字的分數
scores = self.outProjector(hstates)

# Step (5): 用 temporal_softmax_loss 來計算 loss
loss = temporal_softmax_loss(scores, captions_out, self.ignore_index)
```

測試結果：

```
expected loss:  150.6090393066
loss:  150.60903930664062
difference:  0.0
```

定義好這些之後，我們會先用少量的資料來測試我們的 model，在這裡稱為 overfit small data，以下為訓練結果：

```
(Epoch 75 / 80) loss: 1.9251 time per epoch: 0.0s
(Epoch 76 / 80) loss: 1.7434 time per epoch: 0.0s
(Epoch 77 / 80) loss: 1.5925 time per epoch: 0.0s
(Epoch 78 / 80) loss: 1.4542 time per epoch: 0.0s
(Epoch 79 / 80) loss: 1.3274 time per epoch: 0.0s
```



最後我們要實作 RNN class 的 sample 部分，以生成圖片的說明文字，以下為其實作：

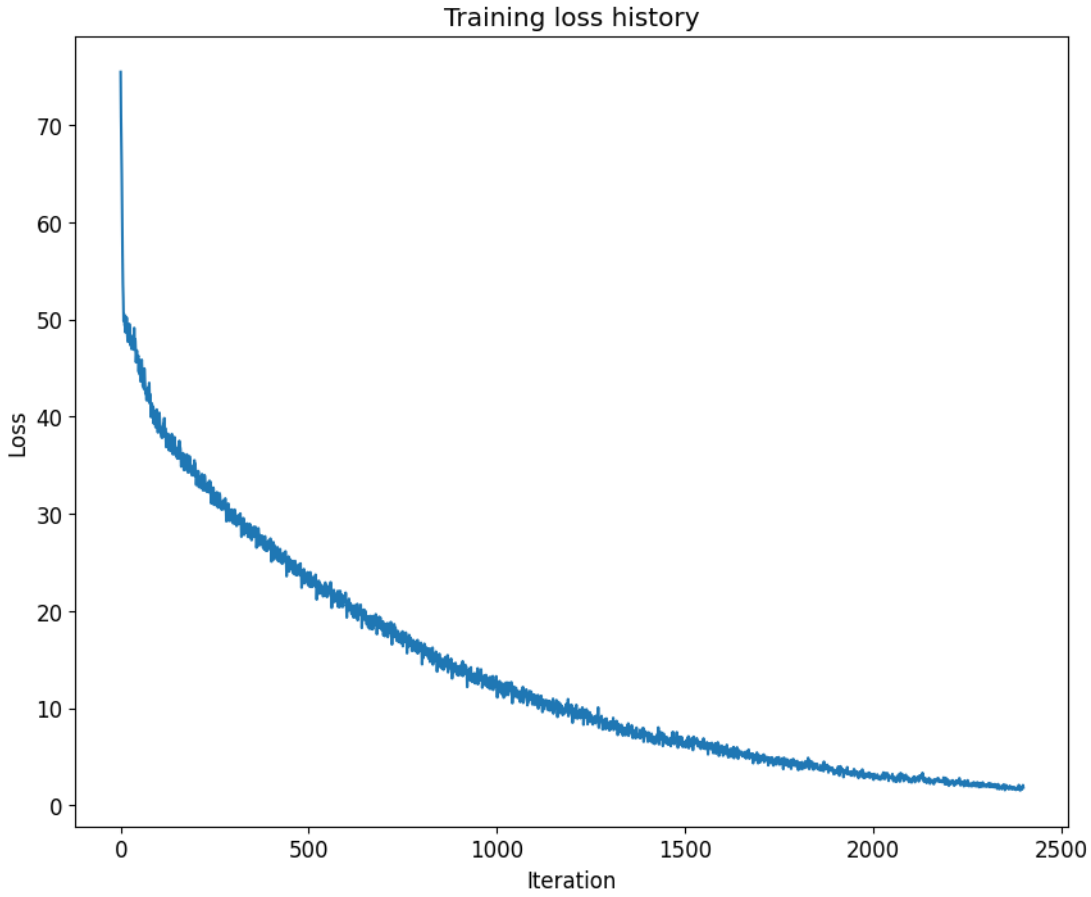
```
# 從圖片中抓特徵
features = self.featureExtractor.extract_mobilenet_feature(images)

# 設定計算裝置
device = features.device
captions = captions.to(device=device)

# affine轉換
h = self.featureProjector(features)

# <START> token
fwords = self._start

notend = torch.full([N], True, device=device)
```

```
# 開始為 minibatch sample 產生 token
for ts in range(max_length):
    x = self.wordEmbedding(fwords) # word embedded
    h = self.coreNetwork.step_forward(x, h) # 透過 RNN 做 forward
    hts = h.unsqueeze(1) # reshape
    temp = self.outProjector(hts) # temporal affine forward
    temp = temp.squeeze() # reshape

    # 選分數最高的字
    fwords = torch.argmax(temp, axis=1)

    # 確認 <END> 在每個 sample 中 都有遇到，並且標記
    mask = fwords == self._end
    notend[mask] = False

    # 確認已經遇到 <END>，如果已經遇到則停止產生 token
    if not notend.any():
        break

    # 如果 <END> 還沒被遇到，則將當前 timestamp 生成的單字加到 caption 中
    captions[notend, ts] = fwords[notend]
```

實作好 class 內的 sample method 後，我們就可以來訓練 model 並產生圖片說明文字了

訓練：

```
(Epoch 55 / 60) loss: 2.2918 time per epoch: 2.8s
(Epoch 56 / 60) loss: 2.2148 time per epoch: 2.8s
(Epoch 57 / 60) loss: 2.0268 time per epoch: 2.8s
(Epoch 58 / 60) loss: 1.7302 time per epoch: 2.8s
(Epoch 59 / 60) loss: 1.8411 time per epoch: 2.8s
```

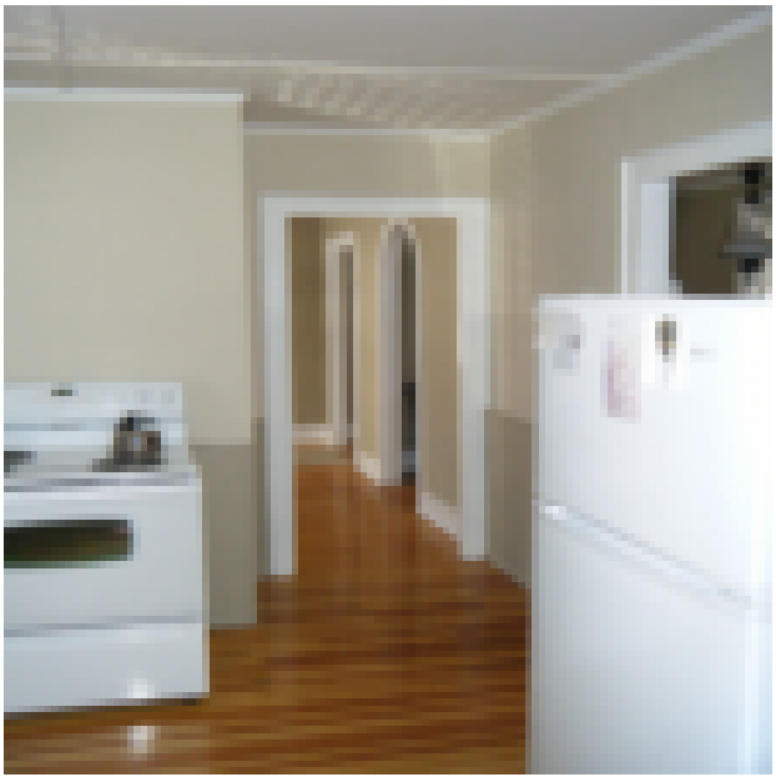


以下是一些 sample 後的 picture caption 結果：

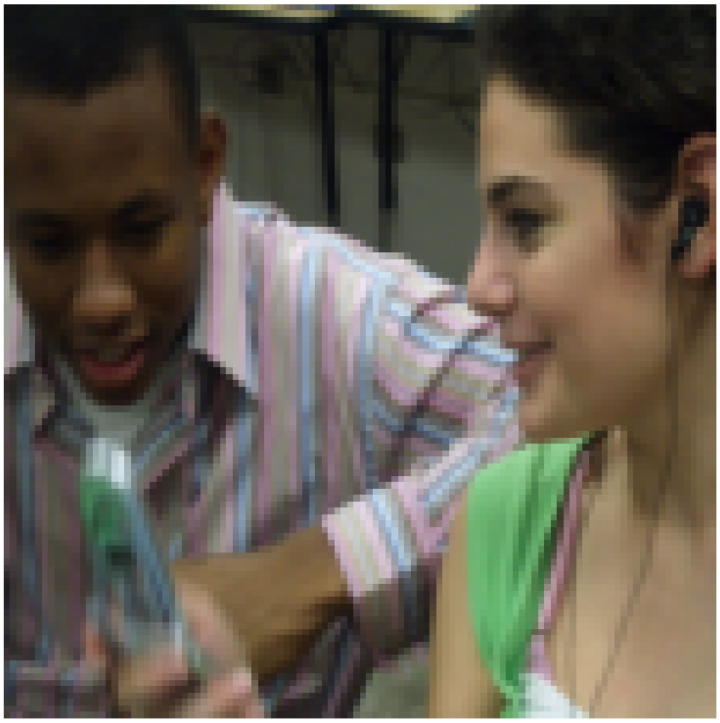
train
RNN Generated:a man skiing on a mountain in the snow
GT:<START> a group of skiers are standing in the snow <END>

train
RNN Generated:a group of people walking along a dirty snow covered road
GT:<START> a group of people walking along a dirty snow covered road <END>

train
RNN Generated:a kitchen counter with a <UNK> and pots
GT:<START> a view of a kitchen and a <UNK> in a home <END>



val
RNN Generated:two guys one person holding a teddy bear at a <UNK> cake
GT:<START> a boy and a girl sitting next to each other looking at an <UNK> <UNK> <END>



val
RNN Generated:a white toilet sitting inside of a bathroom next to two sinks
GT:<START> a close up of a white toilet and <UNK> can <END>

val
RNN Generated:a <UNK> sitting at a table with a very large pizza in it
GT:<START> a plate of <UNK> and bread sit next to a beer bottle on a table <END>

其中包含了 train data 和 validation data 的結果，我們可以看到 caption 和圖片實際的情況是符合的。