

MLDS HW2-1 Video Caption Generation

(1) Model description(3%)

1. 詞向量：
 - a. 資料前處理：
使用training及testing label製作corpus。
 - b. 製作wordvec model：
使用gensim套件的wordvec，對corpus做self training，產生128維的wordvec model。詞出現頻率小於3次就不做列入訓練。
 - c. 製作詞字典：
先將前四個index保留為<PAD>、<SOS>、<EOS>以及<UNK>後，再從wordvec model取出詞排序。
2. 資料前處理：
 - a. Feature：由於提供的資料已經抽出，不再做處理。我們改寫Datasets class以便後續的loader。
 - b. Label：逐句讀入後，將詞轉為詞向量。這裡我們設一個句子長度的限制，最多不能超過10個詞，是因為發現過長的句子不好提高performance。過長的句子直接在尾端加<EOS>，過短的句子加完<EOS>後補<PAD>。
3. 模型概況：

```
Encoder(  
    (fc): Linear(in_features=4096, out_features=128, bias=True)  
    (gru): GRU(128, 256, num_layers=3, batch_first=True, dropout=0.5, bidirectional=True)  
)  
Decoder(  
    (embedding): Embedding(3744, 128)  
    (fc1): Linear(in_features=256, out_features=1280, bias=True)  
    (fc2): Linear(in_features=1280, out_features=1280, bias=True)  
    (fc3): Linear(in_features=1280, out_features=3744, bias=True)  
    (gru): GRU(128, 256, num_layers=3, batch_first=True, dropout=0.5)  
)
```

- a. Encoder：
 - i. fc層：將4096維的feature降至128維，並做dropout。
 - ii. rnn層：三層GRU，維度由128轉至256維。
 - iii. 輸出層操作：由於是bidirectional，因此rnn出來的output與hidden都需要將雙向的值疊加。
- b. Decoder：
 - i. Embedding層：將vocabulary的維度降至前處理時所產生的詞向量維度。並且load進pretrained wordvector，未被訓練的四個保留字使用random產生詞向量後，包進embedding的weight內，且在訓練過程將weight固定住。

```
self.embedding = nn.Embedding(vocab_size, embed_size)  
pretrained = word2vec.Word2Vec.load('model/word2vec.128d')  
pretrained_vectors = np.array([pretrained[word] for word in (pretrained.wv.vocab)])  
pretrained_vectors = np.concatenate((np.random.rand(4,128)-0.5, pretrained_vectors))  
self.embedding.weight.data.copy_(torch.from_numpy(pretrained_vectors))  
self.embedding.weight.requires_grad = False
```

- ii. Attention層（視情況加入）：輸入為將hidden layer與embedding output的維度連在一起（256+128），輸出則是sequence長度（80）。

```
# decoder_input: previous word(embedded), (batch_size, 1, embed_size)
# decoder_hidden: previous hidden, (n_layer, batch_size, hidden_size)
# encoder_outputs, (batch_size, seq_len, hidden_size)

# (1, batch_size, embed_size)+ (1, batch_size, hidden_size)
# -> (1, batch_size, embed_size+ hidden_size)
mix = torch.cat([decoder_input.transpose(0,1),
                 decoder_hidden[0,:,:].unsqueeze(0)], dim=2)

# (1, batch_size, seq_len)
mix = self.linear(mix)

# Attention weight |
# (1, batch_size, seq_len)
weight = F.softmax(mix, dim=2)
# print(weight.shape)
# print(encoder_outputs.shape)
# (batch_size, 1, seq_len) * (batch_size, seq_len, hidden_size)
# -> (batch_size, 1, hidden_size)
attention_output = torch.bmm(weight.transpose(1,0), encoder_outputs)
```

- iii. attention output與embedded output連在一起，經過relu後通過rnn。

```
if self.attention_model is None:
    embed = self.embedding(x)
    out, hidden = self.gru(embed, hidden)
else:
    embed = self.embedding(x)
    attention_output = self.attention_model(embed, hidden, encoder_outputs)
    out = torch.cat([attention_output, embed], dim=2)
    out = F.relu(out)
    out, hidden = self.gru(out, hidden)
out = self.fc1(out)
out = F.relu(out)
out = self.fc2(out)
out = F.relu(out)
out = self.fc3(out)
out = F.log_softmax(out, dim=2)
```

- iv. rnn的output經過三層linear與relu後，以log_softmax輸出。

4. 其他

- a. 優化器：SGD或Adam
- b. Loss function：NLLLoss

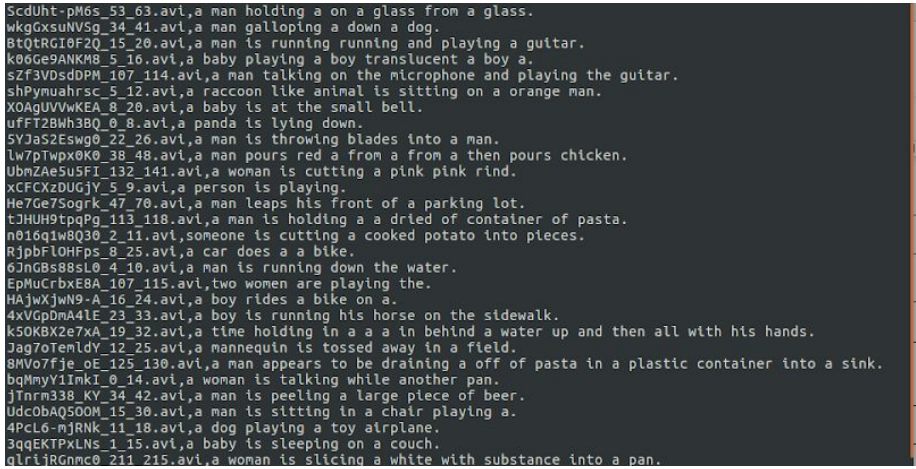
(2) Write down the method that makes you outstanding(1%), why do you use it? (1%), analysis and compare your model without the method(1%)

1. 使用gensim訓練詞向量

描述	普遍來說，使用pretrain的詞向量是常見的作法，雖然這個task全部的caption只有25906個句子，不過我們猜想先使用word2vec建立詞向量或許可以有更好的結果。
分析	<p>結果：（相同參數下）</p> <ol style="list-style-type: none"> 1. 使用GloVe的100d與200d，BLEU@1大約為0.52、0.48 2. 使用自訓練的word2vec詞向量，BLEU@1大約為0.58 <p>推論：</p> <ol style="list-style-type: none"> 1. 由於這次的task還算小，詞出現的不多，因此不能算是個generalized的case。這時提供經過數百萬個字訓練的GloVe容易有bias：在general case中很相近的字或許在這次的task並不相近。

2. 使用所有的caption

描述	一開始每一個影片只有使用固定一個caption，雖然可以訓練的很好，loss非常小，在training data上完全符合ground truth，但generalization的能力很爛，因此最後決定使用所有的caption，增加model理解畫面和詞之間關係的能力。
----	---

分析	<p>僅僅使用單一caption的結果，訓練loss可以非常小，在training data中產生的句子可以完全符合groubd truth，但是testing的結果是high bias，語意跟影片關聯不大，但句子的都頗為完整，testing的bleu score為0.45~0.5。</p> <p><使用單一caption的驗證結果></p> 
----	---

3. 不在feature加入雜訊

描述	此應用中，相同的影片可以有許多不同但皆為合理的輸出，因此一個影片會對應到多個可能的ground truth，因此在dataloader每次要製造出下一個batch之前加入隨機的小雜訊，猜想這樣做可以讓model更為robust。
分析	<p>結果：</p> <p>加入雜訊使得訓練前的預處理非常耗時。BLEU@1的結果並沒有很突出0.45~0.5。</p> <p>推論：</p> <p>一般來說，在較為簡單且訓練資料量大的情況下加入雜訊，有助於generalize。但這次的feature僅1450項，光是feature與多種caption對應的task已經足夠複雜，這時再加入雜訊將會導致performance變差。</p>

4. 不減少使用的幀數

描述	data的總數為1450個影片，而且影片大多短而且畫面變化少，理論上僅需focus在某些特徵上，就很容易符合true label，進而增高BLEU。所以我們猜想可以用少於80個畫面來完成訓練。
分析	<p>結果：</p> <p>減少幀數有助於提升訓練速度。BLEU@1的結果與控制組相近0.5~0.55。</p> <p>推論：</p> <p>降低幀數等於減少機器可使用的feature，理論上就算能在training set表現佳，在沒有看過的case (test set) 上應該會因為不夠generalize導致表現變差。但這次的結果還算是可以接受。</p>

5. 使用Attention layer

描述	使用attention應該有助於幫助encoder在讀入sequence data時，學習focus在某些特定的序列上。
分析	結果：詳見下題。

6. 使用schedule sampling

方法	訓練的初期model通常無法給出太正確的答案，因此直接拿ground truth輸入decoder的input可以加快訓練，而若是總是拿ground truth作為input，那在evaluation，缺少ground truth的情況時，model若是輸出了有偏差的output並且以此作為下一個時間點的input，因為是訓練時從沒看過的情況，model很可能會給出更不合理的輸出。因此在訓練中後
----	---

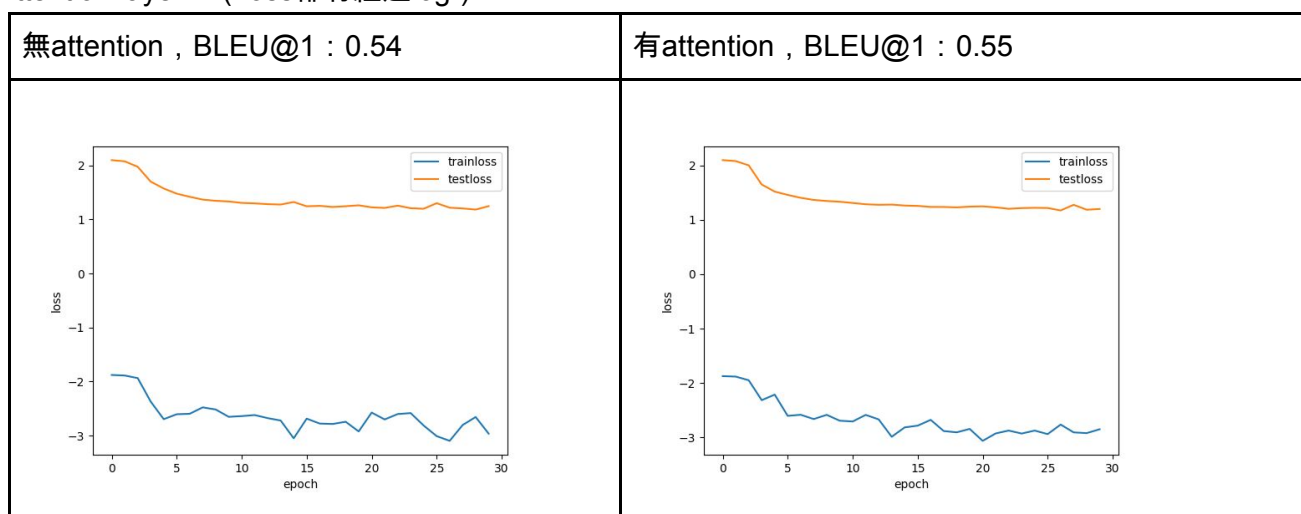
	期可以使用model自己的輸出作為下一個時間點的輸入，讓model較generalize，比較能面對不同distribution的資料。
描述	結果：詳見下題。

(3)Experimental results and settings(1%)

1. BLEU score的觀察：

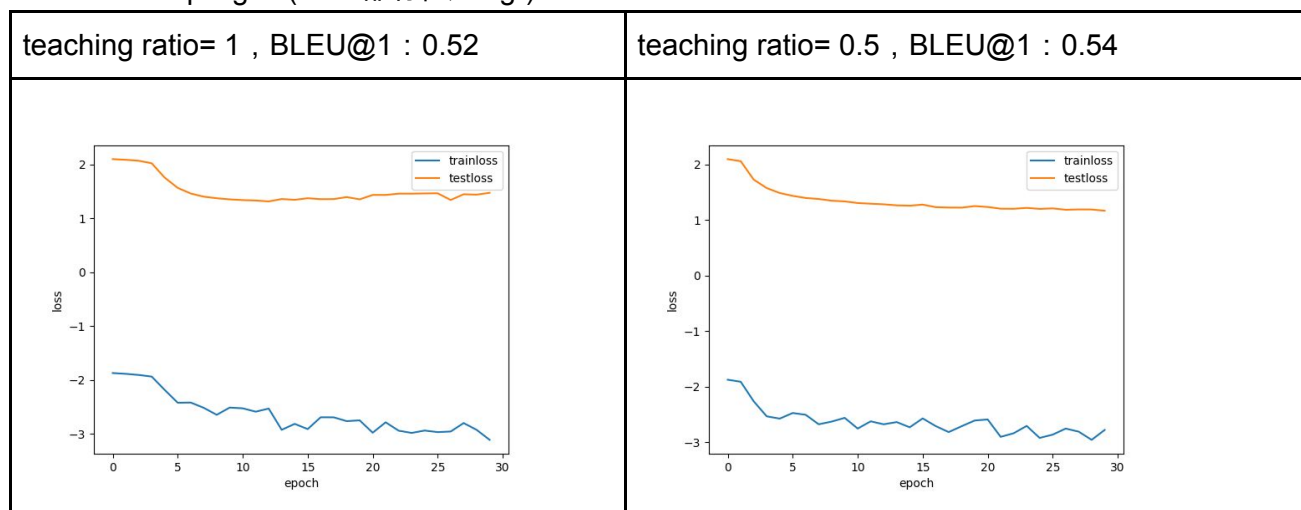
training的loss和testing的bleu score沒有必然的相關性，有時反而稍微train幾個epoch的bleu score是最高的（輸出像是a man is a a a），而train直到開始有文法架構的句子時bleu score卻掉到0.4~0.5，我認為在這個task中，generalization十分困難。

2. Attention layer：（loss都有經過log）



為了彰顯結果的顯著性，這時training的資料量只有100。其實結果差不多，可能是因為這次的task序列不夠長，且資料量不足以顯著提升attention的效果。

3. Schedule sampling：（loss都有經過log）



當teaching ratio= 1，也就是全部都照著前一個詞做訓練的結果，會發現到後期在training loss上都持續下降，然而test loss會緩緩上升。這與預期相符，因此我們認為加入適當的un-taught case，可以達到較好的效果