

Lab 5 Conditional sequence to sequence VAE

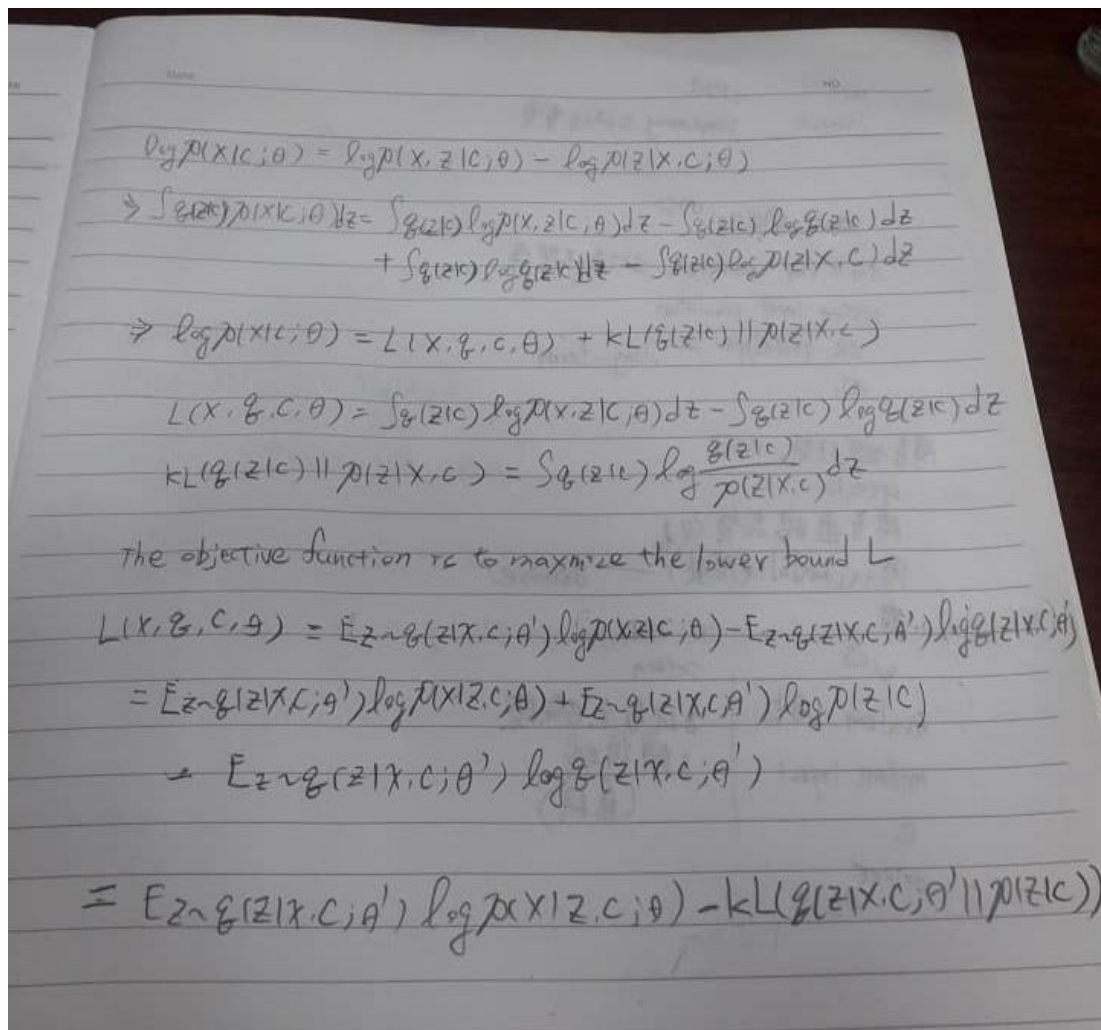
309553005 賴佑家

Introduction

本次的 lab 為使用 CVAE 以及 s2s 的架構來處理英文的時態轉換，將英文字詞 embedding 成向量後加上時態的 condition (simple present, third present, present progressive, simple past)，輸入到 model 中，經過 encoder 會產生 latent code，latent code 再經過 decoder 就會解碼出英文的字詞

而當我們訓練完 CVAE 後，我們就可以給予 CVAE 一個 Gaussian noise 來去 generate input 英文單字的四個時態

Derivation of CVAE


$$\begin{aligned}\log p(x|c; \theta) &= \log p(x, z|c; \theta) - \log p(z|x, c; \theta) \\ \Rightarrow \int q(z|c) \log p(x|c; \theta) dz &= \int q(z|c) \log p(x, z|c; \theta) dz - \int q(z|c) \log q(z|c) dz \\ &\quad + \int q(z|c) \log q(z|c) dz - \int q(z|c) \log p(z|x, c) dz \\ \Rightarrow \log p(x|c; \theta) &= L(x, q, c, \theta) + KL(q(z|c) || p(z|x, c)) \\ L(x, q, c, \theta) &= \int q(z|c) \log p(x, z|c; \theta) dz - \int q(z|c) \log q(z|c) dz \\ KL(q(z|c) || p(z|x, c)) &= \int q(z|c) \log \frac{q(z|c)}{p(z|x, c)} dz \\ \text{The objective function is to maximize the lower bound } L \\ L(x, q, c, \theta) &= E_{z \sim q(z|x, c; \theta')} \log p(x, z|c; \theta) - E_{z \sim q(z|x, c; \theta')} \log q(z|x, c; \theta') \\ &= E_{z \sim q(z|x, c; \theta')} \log p(x|z, c; \theta) + E_{z \sim q(z|x, c; \theta')} \log p(z|c) \\ &\quad - E_{z \sim q(z|x, c; \theta')} \log q(z|x, c; \theta') \\ &= E_{z \sim q(z|x, c; \theta')} \log p(x|z, c; \theta) - KL(q(z|x, c; \theta') || p(z|c))\end{aligned}$$

Implementation details

- Encoder
首先將 condition (tense) embedding 後與 hidden state concat 在一起成 LSTM 的 hidden state，之後再一個字母一個字母與 LSTM 的 hidden state 與 cell state 傳入 encoder，而 encoder 產生出的 hidden state 與 cell state 再當成下一階段的 input，最後再把 encoder 最後的 hidden state 經過兩個 fully connect layer 產生出 mean 與 logvar
- Sample
有了 encoder output 出的 mean 與 logvar 後，便可以用 mean 與 logvar 使用 reparameterization trick 去 sample 出 latent code
- Decoder
經過 sample 有了 latent code 後，我們像 encoder 一開始一樣，將 latent 與 condition concat 在一起，當成 LSTM 的 hidden state，之後再將一個 SOS 的 token 與 LSTM 的 hidden state 與 cell state 傳入 decoder，而最後 decoder 產生出的 output 為 prediction distribution，那機率最高的即為 predict 的字母
- Reparameterization trick
從 $N(\text{mean}, \exp(\log_var))$ 中 sample 一點

```
def reparameterize(mean, log_var):  
    """  
    reparameterization trick  
    """  
    eps = torch.randn_like(log_var, device=device)  
    return eps * torch.exp(0.5 * log_var) + mean
```

- Text generation by Gaussian noise
產生一個 Gaussian noise 當成 latent code，再丟到 model 中，與四個型態的 condition 經過 decoder 產生四個時態的單字

```

# Gaussian score
words = []
for i in range(100):
    latent = torch.randn(1, 1, latent_size).to(device)
    words.append(model.generate_words(latent, tense_list))
for word_list in words:
    for i in range(len(word_list)):
        word_list[i] = transformer.tensor2words(word_list[i])
gaussian_score = Gaussian_score(words)

```

```

def generate_words(self, latent, tense_list):
    word_list = []
    for tense in tense_list: # 4 tense
        tense_tensor = torch.tensor(tense, device=device).view(-1,1)
        c = self.conditional2embed(tense_tensor)
        decoder_input = torch.tensor([[SOS_token]], device=device)
        decoder_hidden = torch.cat((latent, c), dim=-1)

        # decoder_hidden = decoder_hidden.long()
        decoder_hidden = self.latent2embed(decoder_hidden)
        decoder_cell = self.decoder.initCell()
        predict_word=[]

        for i in range(self.max_length):
            decoder_output, decoder_hidden, decoder_cell = self.decoder(decoder_input, decoder_hidden, decoder_cell)
            predict_class = torch.argmax(decoder_output)
            predict_word.append(predict_class)
            if predict_class.item() == EOS_token:
                break
            decoder_input = predict_class

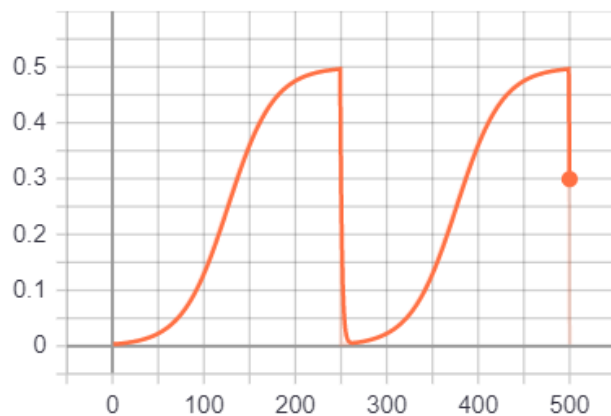
        word_list.append(predict_word)

    return word_list

```

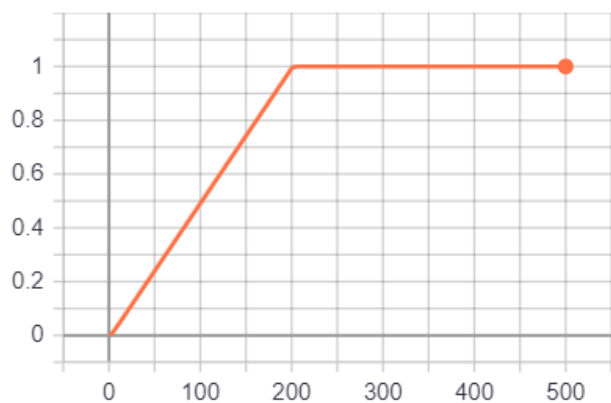
- Dataloader
 - a. 有一個 transformer 的 class，會設立英文字母與 index 的對應以及 index 與英文字母的對應並存成 dictionary，之後便可以來進行單字與 tensor 之間的相互轉換
 - b. Dataloader 方面我是使用 pytorch 的 dataloader，從 dataset 中拿到單字與時態後，利用前面的 transformer 轉換成 tensor 後回傳
- Hyperparameters
 - Epochs: 500
 - Hidden size: 256
 - Learning rate: 0.05
 - KL weight
 - Cycle
 - 使用 sigmoid annealing

KL weight
tag: Setting parameter/KL weight



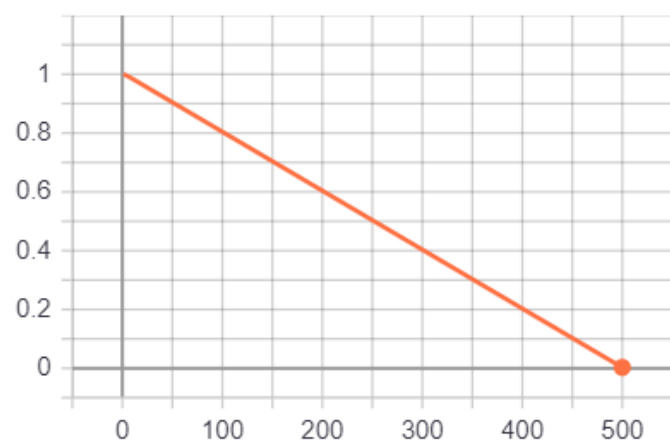
■ Monotonic

KL weight
tag: Setting parameter/KL weight



Teacher forcing ratio: 初始為 1 隨著 epoch 增加，遞減為 0

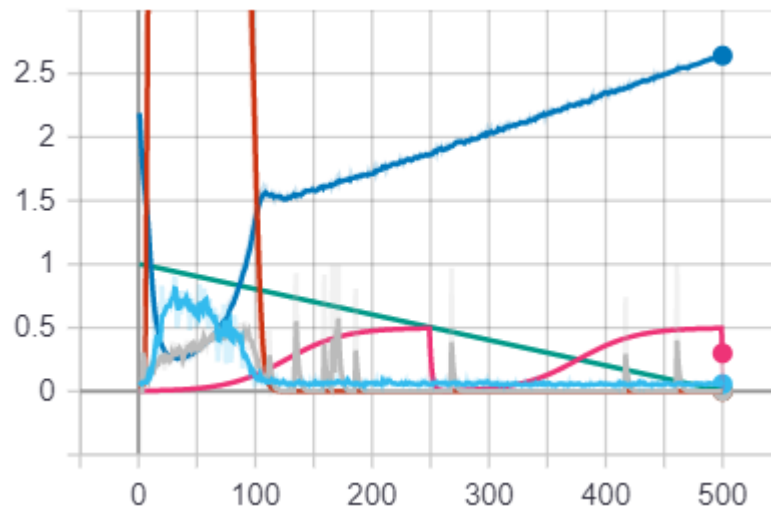
teacher forcing ratio
tag: Setting parameter/teacher forcing ratio



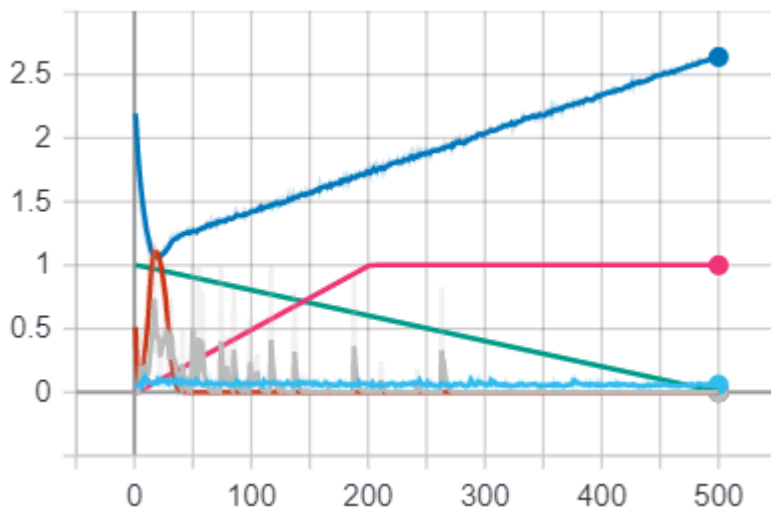
Results



1. cycle



2. monotonic



X 軸: epochs, Y 軸: value

BLEU-4

```
-----  
Input:  functioning  
Target: functioned  
Prediction: functioned  
-----  
  
-----  
Input:  healing  
Target: heals  
Prediction: heals  
-----  
  
Average BLEU-4 score: 0.7555811953127856
```

Gaussian score

```
foredo, foredoes, foredoing, foredoed  
cause, causes, causing, caused  
declaim, declaims, declaiming, declaimed  
recall, recarges, recarging, recalled  
yield, yields, finging, yielded  
finalize, finalizes, finalizing, finalized  
feer, feers, feering, fermented  
roubl, roubles, rnunbing, rnunbed  
schedule, schedules, strubling, scheduled  
Gaussian score: 0.31
```

Discussion

一開始 KL weight 還很低，比較偏重在 cross entropy 的計算，讓 model 專注在重建單字的部分，因此在 KL weight 低的時候，cross entropy loss 會開始下降，隨著 cross entropy loss 的下降，BLEU 也會隨著提升，但同時這樣也會讓 latent distribution 與 $N(0,1)$ 越來越不像，導致 KL loss 往上升

到後面當 KL weight 開始上升到一定階段時，KL 的計算比重會開始增加，導致 KL loss 會開始下降，隨著 KL loss 的下降，Gaussian score 也會隨著提高，但 reconstruction loss 會開始上升，兩者其實有點是對抗的感覺

在 KL annealing 方面，我的 cycle annealing 的 performance 比 monotonic 的還要好，經過觀察我覺得是我 monotonic 在前面 weight 上升時的斜率沒有調整好，沒有像我的 cycle annealing，讓 model 前面有足夠的時間可以讓 model 學好 reconstruction，但由於作業時間的以及還有其他作業的關係，就沒有在對 monotonic 的 annealing 方面多進行 tuning