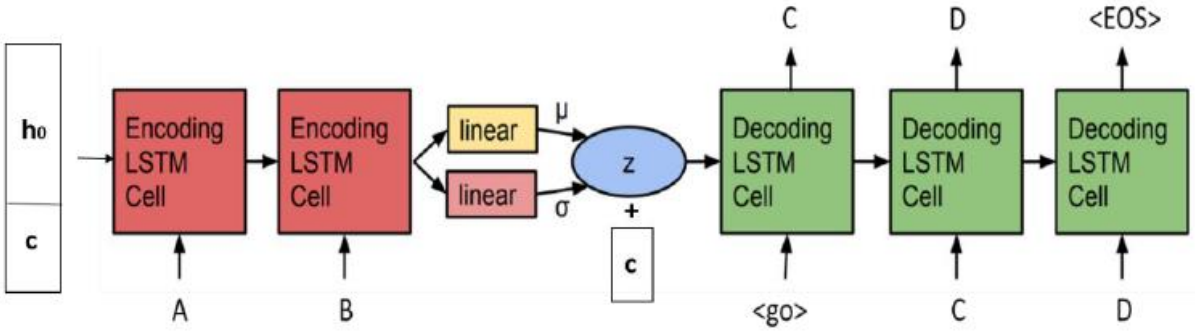
Lab 5: Conditional Sequence-to-sequence VAE

陽明大學 不分系二年級 張凱博

(一) Introduction

這次 lab 主要的內容是著重於使用 seq2seq 的 CVAE model 去做英文單字的時態轉換，只要輸入一個英文單字和時態的限制，就可以讓那個英文單字輸出成為該時態的單字，主要架構如下:



Seq2seq 的架構是由一層 encoder 和另外一層 decoder 組合起來的，將經過 split 後的英文單字經過 embedding 後，將字元一個一個放進 GRU 之中，把 encoder 的最後一層輸出計算 reparameterization trick 以連接 decoder，再來進入 decoder，其中我們用 teacher forcing 控制每次在進入 gru 時的字元是不是輸入正確的字元或預測的字元，最後將每層的 output 收集起來轉換成相對應的字元即可。

CVAE 最大的特點就是在 encoder 的 input 和 decoder 的 input 的後方，concatenate 一個限制條件，這個特點也是 VAE 不存在的，所以我們可以藉由條件限制生成我們像要的目標，使得目標物的生成精確度上升。

(二) Implementation details

A. Describe how to implement model

1. Dataloader

利用 generateData read training pair 和 testing pair，把 data 一行一行讀出來存在 list 裡面，再利用 word2tensor 的這個 function 將 word 轉為以字元為單位的 index tensor，完成 dataloader，若要 implement，則將一個字一個字的送到 CVAE 的 forward 裡面即可。

```
def generateData(lang1):
    print("Reading %s data..."% lang1)
    lines = open('%s.txt' % lang1, encoding='utf-8').read().strip().split('\n')
    data = [l for l in lines]
    return data

# generate training data
pairs=generateData('train')
Training_pairs = [l.split(" ") for l in pairs]
print(Training_pairs)

# generate testing data
pairs=generateData('test')
Testing_pairs = [l.split(" ") for l in pairs]
print(Testing_pairs)
```

```
def wordTotensor(word):
    words = list(word)
    lengthOfword = len(words)
    tensor = torch.zeros(lengthOfword)
    for i in range(lengthOfword):
        tensor[i] = letters[words[i]]
    return tensor
```

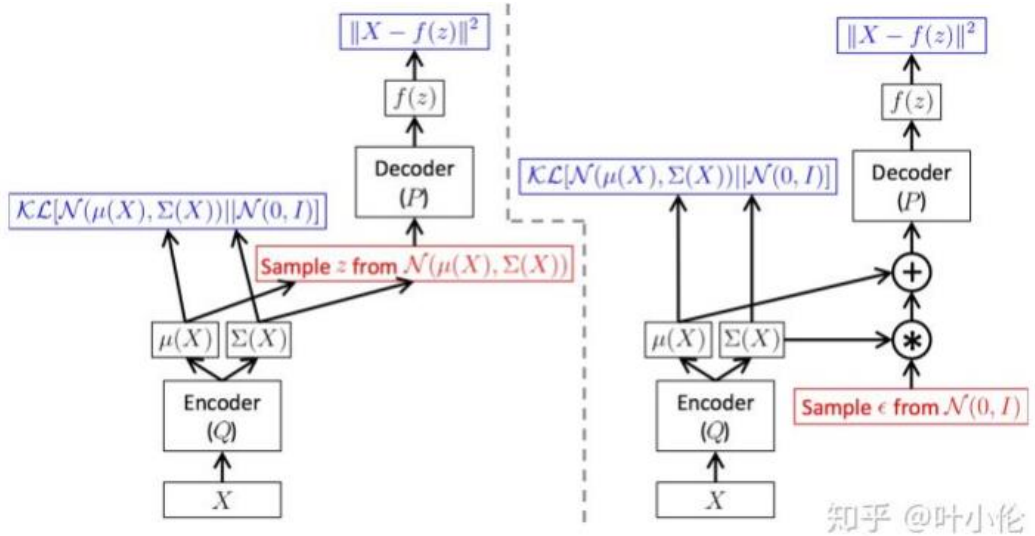
2. Encoder

將 input word 和 conditional tense 做 concatenate 後就可以將其輸入到 encoder 裡面，首先 input word 會被 split 成字元的 index，經過 embedding 投影至 256 的維度後放進 GRU，指定變數 output 為每一層 GRU 的輸出矩陣，hidden 為最後一層的輸出矩陣，encoder 最後的 return 為 output 和 hidden。

```
class Encoder(nn.Module):

    def __init__(self, input_size, hidden_size, latent_dim, n_classes):

        super().__init__()

        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size + n_classes)
        self.hidden_size = hidden_size

        self.mu = nn.Linear(hidden_size + n_classes, latent_dim)
        self.var = nn.Linear(hidden_size + n_classes, latent_dim)

    def forward(self, input_tensor, hidden):
        embedded = self.embedding(input_tensor).view(1, 1, -1)
        output = embedded
        output, hidden = self.gru(output, hidden)
        a=output
        d=hidden
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
```

3. Reparameterization trick

Reparameterization trick 最主要的目的是要確保從 encoder 到 decoer 端的過程中的 hidden (context vector)保有連續的特性，這樣在做 backpropagation 的微分梯度計算時才不會有問題，以下圖來看，左邊的是沒有經過 Reparameterization trick 的 model，mu(X) 和 Sigma(X)在 encoder 和 decoder 端之間屬於離散，但是若經過 Reparameterization，$z = \mu(X) + \sqrt{\Sigma(X)} \times \epsilon$ ，就可以讓 encoder 和 decoder 端之間有連續的關係。



```
z_mu = self.mu(encoder_hidden)
z_var = self.var(encoder_hidden)

# sample from the distribution having latent parameters z_mu, z_var
# reparameterization trick
std = torch.exp(z_var / 2)
eps = torch.randn_like(std)
x_sample = eps.mul(std).add_(z_mu)
```

4. Decoder

   在 decoder 的部分，除了一樣要放入和 conditional tense concatenate 後的 hidden 外，另外還要再放入 SOS 或者是 GO，來告訴 decoder 要開始進行 decode 了，接下來也是經過 GRU，和 encoder 一樣的是指定變數 output 為每一層 GRU 的輸出矩陣，hidden 為最後一層的輸出矩陣，encoder 最後的 return 為 output 和 hidden，最後用 linear 的 output_size 設定為 28。

```
class Decoder(nn.Module):

    def __init__(self, hidden_size, output_size, latent_dim, n_classes)
        super().__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(output_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size + n_classes)
        self.out = nn.Linear(hidden_size + n_classes, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input_tensor , hidden):
        output = self.embedding(input_tensor).view(1, 1, -1)
        output, hidden = self.gru(output, hidden)
        output = self.out(output[0])
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
```

5. Hidden size: 256 維
6. condition embedding size: 256 維
7. RNN type: GRU

B. Specify the hyperparameters

1. KL weight:
   KLD_weight = 0.01 + 0.01 * lw，在每一次的 forward 中，lw 都會加 1。
2. Learning rate = 0.0001
3. Teacher forcing ratio = 0.5
4. Epochs = 80
5. Vocab_size = 28 (26 個英文字母加 SOS 和 EOS)
6. N_classes=4
7. Latent_dim = 256

(三) Results and discussion

A. Plot the loss and KL loss curve during training and discuss the results

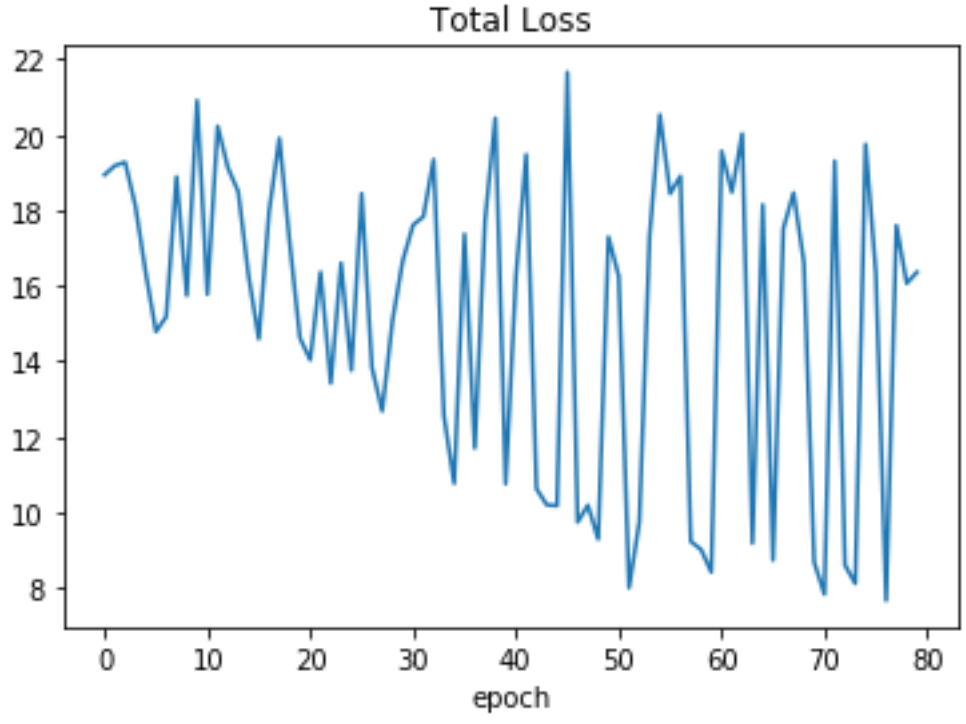
CVAE 和 VAE 的核心在於使用一個 q(Z|X)來對真實的 posterior probability p(X|Z)進行估計，即最小化 q(Z|X)和 p(X|Z)的 KL divergence: KLD(q(Z|X) || p(Z|X))，但是這個無法計算，所以可以將它寫成:

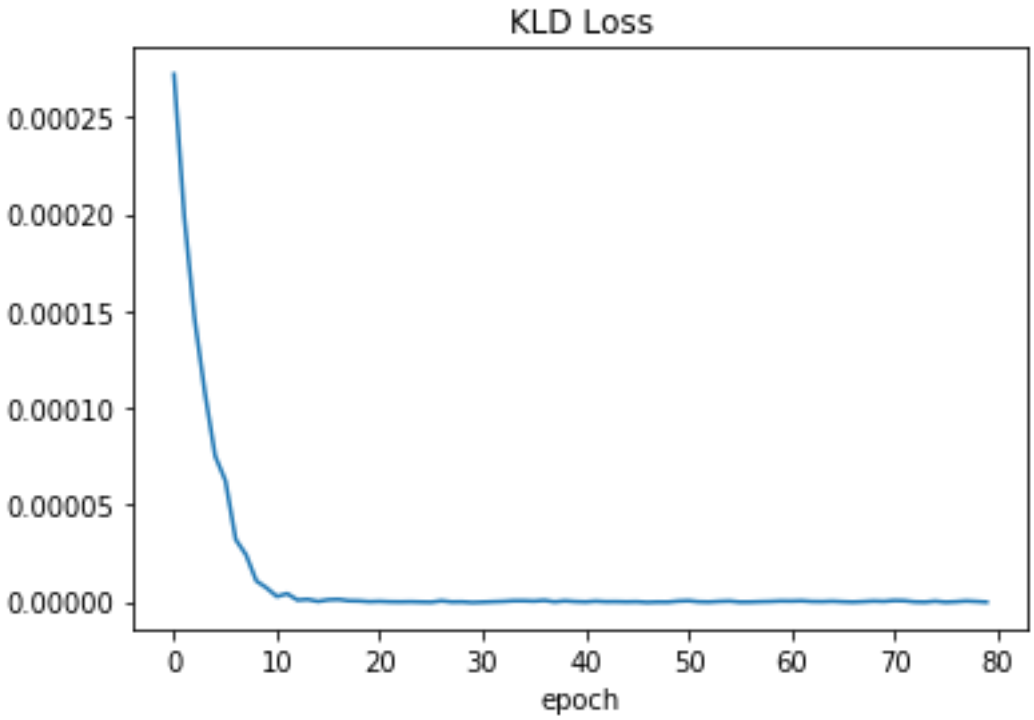
$$
\begin{aligned}
D_{KL}(q(Z|X)||p(Z|X)) &= \mathbb{E}[\log(q(Z|X)) - \log(p(Z|X))] \\
&= \mathbb{E}[\log(q(Z|X)) - \log(\frac{p(X|Z)p(Z)}{p(X)})] \\
&= \mathbb{E}[\log(q(Z|X)) - \log(p(X|Z) - \log(p(Z)))] + \log(p(X)) \\
&= \mathbb{E}[\log(\frac{q(Z|X)}{p(Z)}) - \log(p(X|Z))] + \log(p(X)) \\
&= D_{KL}[q(Z|X)||p(Z)] - \mathbb{E}[\log(p(X|Z))] + \log(p(X))
\end{aligned}
$$

經過簡化:

$$
\log(p(X)) - D_{KL}(q(Z|X)||p(Z|X)) = \mathbb{E}[\log(p(X|Z))] - D_{KL}[q(Z|X)||p(Z)]
$$

所以若要最小化 KLD(q(Z|X) || p(Z|X))的話，就只要將右邊 2 項計算出來，第一項就是程式碼中 Reconstruction 的部分 (Maxium likelihood)，第二項就是 KL divergence，由下面 2 張圖可知，KL divergence loss 已趨近於零，如此一來，使 total loss 上升的就是 Reconstruction loss 的部分雖然從整體來看 loss 雖有在下降的趨勢，但是起伏也很大，估計是因為在 sampling mu 和 sigma 的過程中(決定 prior distribution P(Z) 時)有時會會取到偏離正確值很遠的點，所以才會造成起伏很大的趨勢。

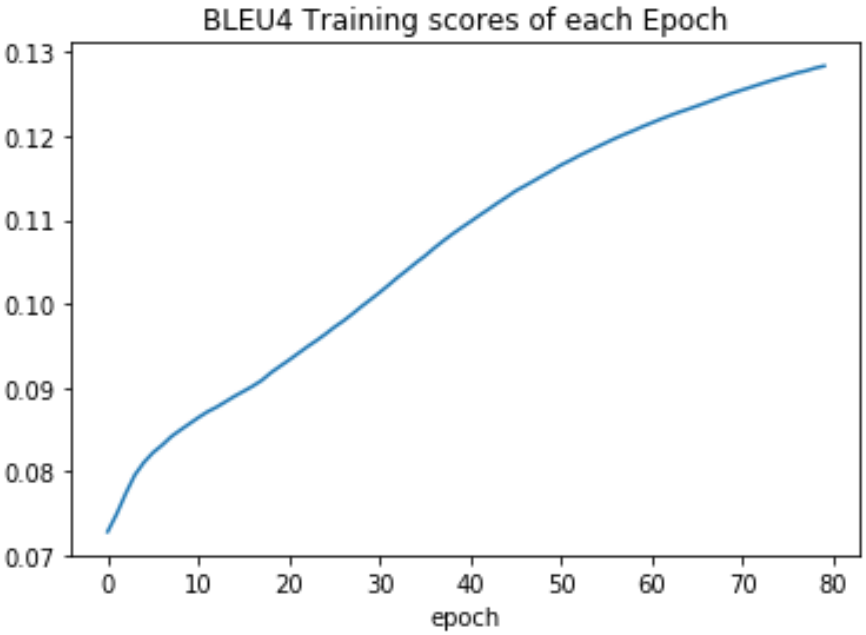


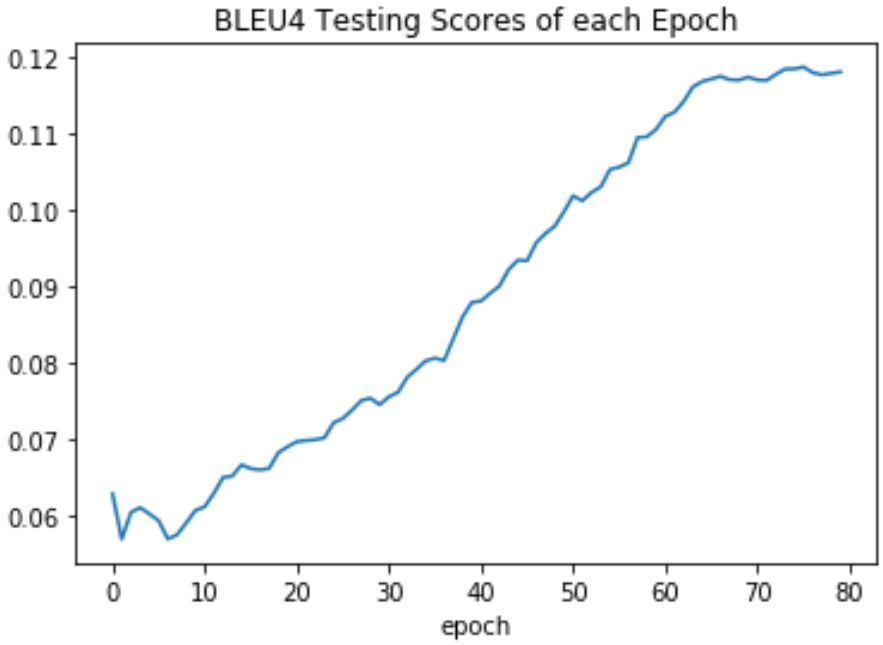


B. Plot the BLEU-4 score of your testing data while training and discuss the result.

BLEU，全稱為 Bilingual Evaluation Understudy（雙語評估替換），是一種對生成語句進行評估的指標。完美匹配的得分為 1.0，而完全不匹配則得分為 0.0。在這次的評分中，明顯的分數不是很高，最高的 training score 為 0.128，最高的 testing score 為 0.119，我猜是因為 vocab_size (=28)太長的原因，所以在 encoder 和 decoder 的時候梯度消失導致 model 無法記住前面的字元，此外，conditional tense 的 array (1*4) 太小也是個問題，加在 word_index 後面無法突顯有 condition 的作用，改進的方法就是可以用試用 attention 使得 model 可以記得較多的東西，或許分數就可以上升了。

```
#compute BLEU-4 score
def compute_bleu(output, reference):
    cc = SmoothingFunction()
    return sentence_bleu([reference], output, smoothing_function=cc.method1)
```




BLEU4 Testing Scores of each Epoch
0.12
0.11
0.10
0.09
0.08
0.07
0.06
0
10
20
30
40
50
60
70
80
epoch