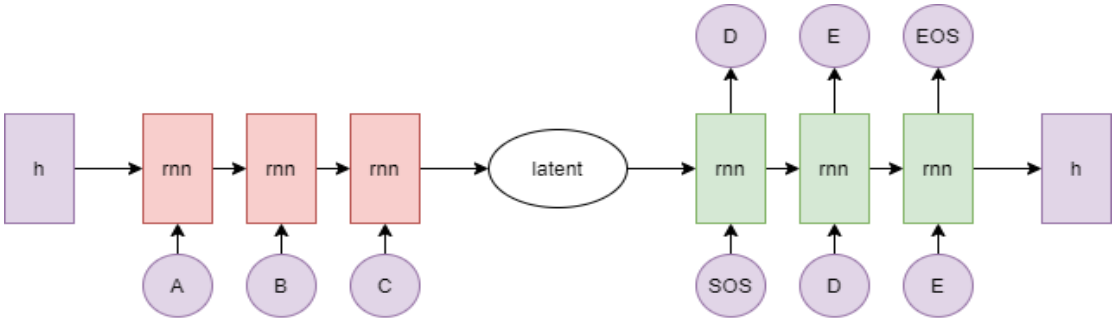
DLP_LAB4_309551124_涂景智

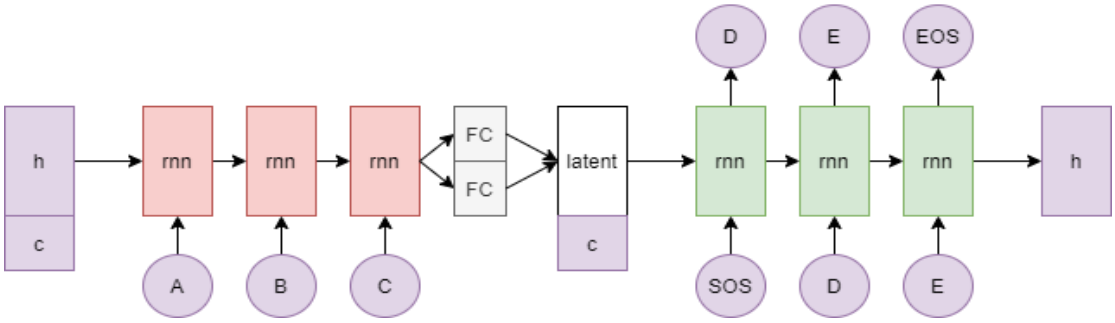
A. Introduction

此次的 LAB 需要實作 CVAE (Conditional Variational Auto-Encoder)。傳統的 VAE 存在一個問題，就是其雖然可以生成樣本，但我們無法指定生成樣本的類別。而 CVAE 主要是透過在訓練時加入一個 one-hot 向量用來表示標籤向量，讓我們的 model 在學習時能加入到標籤因素，使我們之後可以按照指定的標籤生成我們想要生成的類別資料。

VAE



CVAE



比較要注意的是，CVAE 圖中的二個 FC 層，是用來幫助實作 Reparameterization trick 的。另外此次 LAB 有規定，CVAE 中的 RNN 必須要採用 LSTM。也因此，整個 LAB 的架構大致為：

1. 將每個 input word 轉成 tensor，加入對應的 label 後傳入 LSTM（Encoder）中。
2. 將 LSTM 輸出的 hidden （就是上圖中的 latent），加入 target 的 label 後帶入 Decoder 做運算（Decoder 的 input 根據是否要 teach

forcing 而有所不同）

3. 將 Decoder 的輸出再轉換回數個字元，產生新的 word。
4. 算出生成 words 的 Gaussian Score 和 BLEU-4 score，評斷模型結果

B. Derivation of CVAE

原先 VAE 的 objective 為：

$$\log P(X) - D_{KL}[Q(z|X)\|P(z|X)] = E[\log P(X|z)] - D_{KL}[Q(z|X)\|P(z)]$$

其中我們要 optimize 的是 P(X) 的 log likelihood。而原先的 VAE 中包含二個部分，encoder Q(z|X) 和 decoder P(X|z)。

而在 CVAE 中，我們的 variational lower bound objective 改為：

$$\log P(X|c) - D_{KL}[Q(z|X,c)\|P(z|X,c)] = E[\log P(X|z,c)] - D_{KL}[Q(z|X,c)\|P(z|c)]$$

與 VAE 大致相同，差別只在我們在訓練 distributions 時會包含 variable c。

C. Implementation details

1. Data loader

下圖為 data loader 的實作 code：

```
class Dataloader(data.Dataset):
    def __init__(self):
        self.total_word = []
        self.total_num = []
        f = open("lab4_dataset/train.txt")
        line = f.readline()
        while line:
            self.total_word.append(line.rstrip().split(" "))
            line = f.readline()

        for each_line in self.total_word:
            num_each_line = []

            for each_word in each_line:
                none_shuffle = Char_to_Num(each_word)
                num_each_line.append(none_shuffle)

            self.total_num.append(num_each_line)

    def __len__(self):
        return len(self.total_num)

    def return_training_pair(self):
        _return = []

        x = random.randint(0, self.__len__()-1)
        for i in range(2):
            y = random.randint(0,3)

            _return.append(self.total_num[x][y])
            _return.append(y)

        return _return
```

將 train.txt 檔讀進來之後，透過自寫的 「Char_to_num()」 將英文字母轉成數字並儲存在陣列裡。而 function「return_training_pair()」則是在我們訓練 model 時會被呼叫，該函式會回傳二個同一動詞（但時態不一定相同）的字詞和其對應的標籤（時態），做為該次訓練的 input 和 target。

2. Encoder

```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.lstm = nn.LSTM(hidden_size, hidden_size)
        self.fc1 = nn.Linear(hidden_size, latent_size)
        self.fc2 = nn.Linear(hidden_size, latent_size)
        self.fc_c = nn.Linear(hidden_size, latent_size)

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5*logvar)
        eps = torch.randn_like(std)
        return mu + eps*std

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(len(input), 1, -1)
        output = embedded
        output, hidden = self.lstm(output, hidden)
        z_mu = self.fc1(hidden[0])
        z_var = self.fc2(hidden[0])
        z = self.reparameterize(z_mu, z_var)
        c_0 = self.fc_c(hidden[1])
        return output, (z, c_0), z_mu, z_var

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

會先將 input embed 成 hidden size 的維度後，再將其傳入 encoder 的 RNN(使用 LSTM)。比較重要的應該是 function 「reparameterize()」，該函式顧名思義就是實踐 Reparameterization trick。另外，reparameterize() 會回傳 mean + eps*std，其中的 eps 是 Gaussian 分佈中的隨機數值，用來實踐 Gaussian noise。加入 Gaussian noise 的目的在於透過刻意增加雜訊的方式，增強 model 的能力。

3. Decoder

```
class DecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size):
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(latent_size + 4 , hidden_size)
        self.lstm = nn.LSTM(hidden_size, latent_size + 4)
        self.out = nn.Linear(latent_size + 4, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        output = self.embedding(input).view(1, 1, -1)
        output = F.relu(output)
        output, hidden = self.lstm(output, hidden)
        output = self.out(output[0])
        return output, hidden

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

與 Encoder 稍微不同是，因為 Decoder 的輸入要為原本 encoder 輸出的 latent 在加上 label，因此總共 input 的維度是 latent size + 4 (有 4 種時態、4 種 one-hot 標籤)。

4. Train()

trian() 大致上是根據助教提供的 sample code 改編的，程式有點長，在此只特別介紹一些比較重要的部分。

```
    input_tensor = torch.cat((input_tensor, input_label),0)

    encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)

    loss = 0
    KLD_loss = 0

    input_tensor = input_tensor.to(device)
    target_tensor = target_tensor.to(device)
    target_label = target_label.to(device)

    #----------sequence to sequence part for encoder----------#
    encoder_output, encoder_hidden, mu, logvar = encoder(input_tensor, encoder_hidden)


    decoder_input = torch.tensor([[SOS_token]], device=device)

    new_h0 = torch.cat((encoder_hidden[0], target_label), 2).to(device)
    new_c0 = torch.cat((encoder_hidden[1], target_label), 2).to(device)

#     decoder_hidden = encoder_hidden
    decoder_hidden = (new_h0, new_c0)
```

在上圖的程式中，我們先將 input tensor 與其對應的 label concatenate 起來（label 已經有改成 one-hot），並與 encoder_hidden (透過 Encoder.initHidden() 產生) 一起傳進 encoder 中。再將 Encoder 傳出的

hidden (就是我們的 latent) 的 h 跟 c 分別 concatenate target 的 labell。並將其視為 decoder 的 hidden。

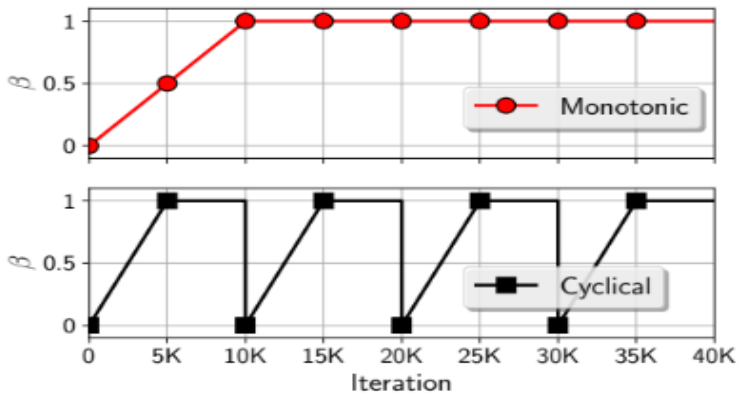
```
use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False

#----------sequence to sequence part for decoder----------#
if use_teacher_forcing:
    # Teacher forcing: Feed the target as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden = decoder(
            decoder_input, decoder_hidden)
        tmp_loss, tmp_KLD_loss = loss_function(decoder_output, target_tensor[di], mu, logvar, epoch)
        loss += tmp_loss
        KLD_loss += tmp_KLD_loss
        decoder_input = target_tensor[di]  # Teacher forcing

else:
    # Without teacher forcing: use its own predictions as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden = decoder(
            decoder_input, decoder_hidden)
        topv, topi = decoder_output.topk(1)
        decoder_input = topi.squeeze().detach()  # detach from history as input

        tmp_loss, tmp_KLD_loss = loss_function(decoder_output, target_tensor[di], mu, logvar, epoch)
        loss += tmp_loss
        KLD_loss += tmp_KLD_loss

        if decoder_input.item() == EOS_token:
            break
```

再來是 Decoder input 的部分。Decoder 的 input 會依據是否要使用 teacher forcing 而定。（teacher_forcing_rate 我是設定 0.7，意即我們會有 70% 的機率使用 teacher forcing）。若使用 teacher forcing，則 decoder 的 input 為 target tensor 的數值（意即「正確應該產生的資料」），若不使用 teacher forcing，則 decoder 的 input 為我們在上一輪 RNN 中產生的 output。

5. Other hyperparameters

KL weight：

KL weight 起始都是 0，根據助教提供的 pdf，KL weight 的數值有二種曲線，分別是 Monotonic 和 Cyclical。後面會介紹這二種曲線效能的比較結果。



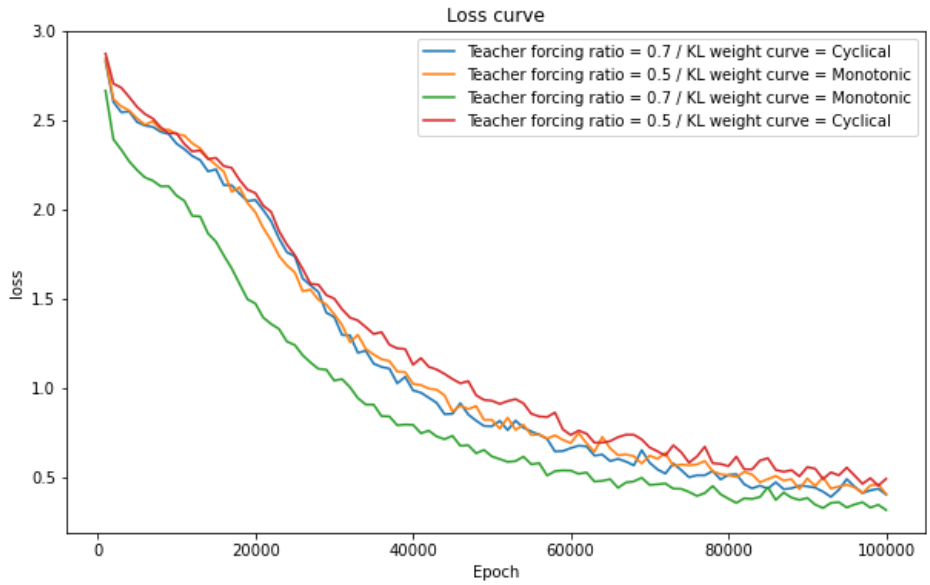
另外，Hidden size, latent size 分別為 32 和 256 (與原本 sample code 預設的相同), learning rate 根據實測結果，設為 0.005 最佳。

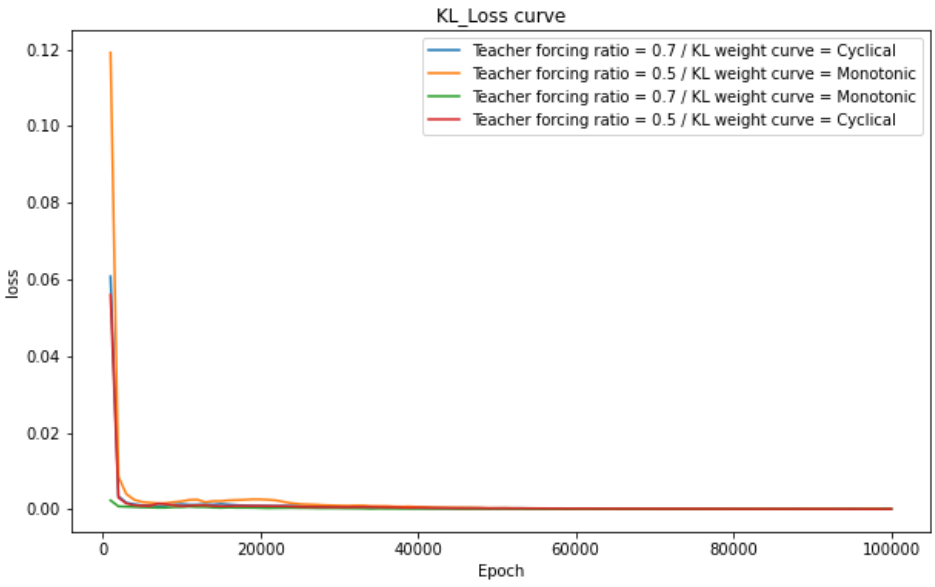
D. Results and discussion

在此總共比較下列四種 model 的 loss, KL loss, Gaussian score, BLEU-4 score：

- Teacher forcing ratio = 0.7 / KL weight curve = Cyclical
- Teacher forcing ratio = 0.5/ KL weight curve = Monotonic
- Teacher forcing ratio = 0.7/ KL weight curve = Monotonic
- Teacher forcing ratio = 0.5/ KL weight curve = Cyclical

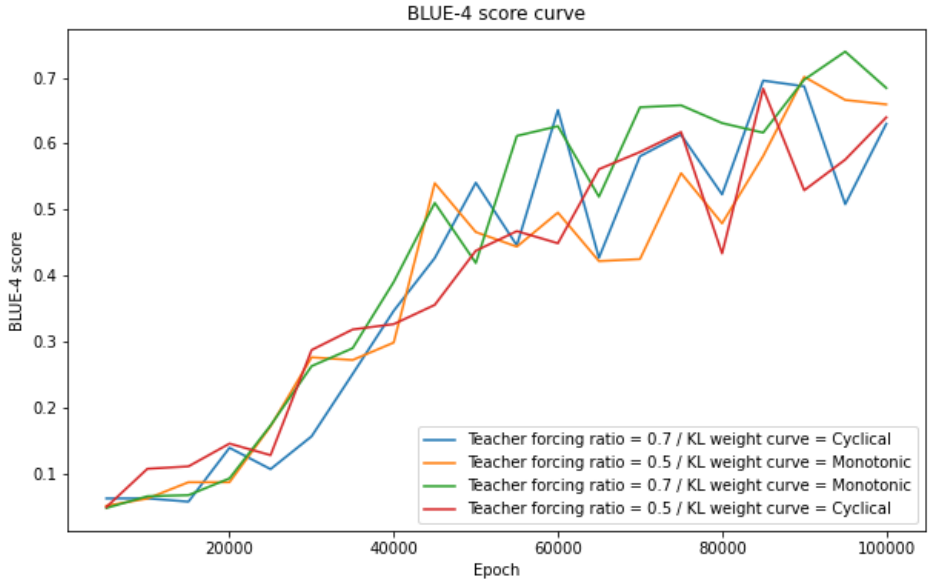
1. Loss curve & KL Loss curve

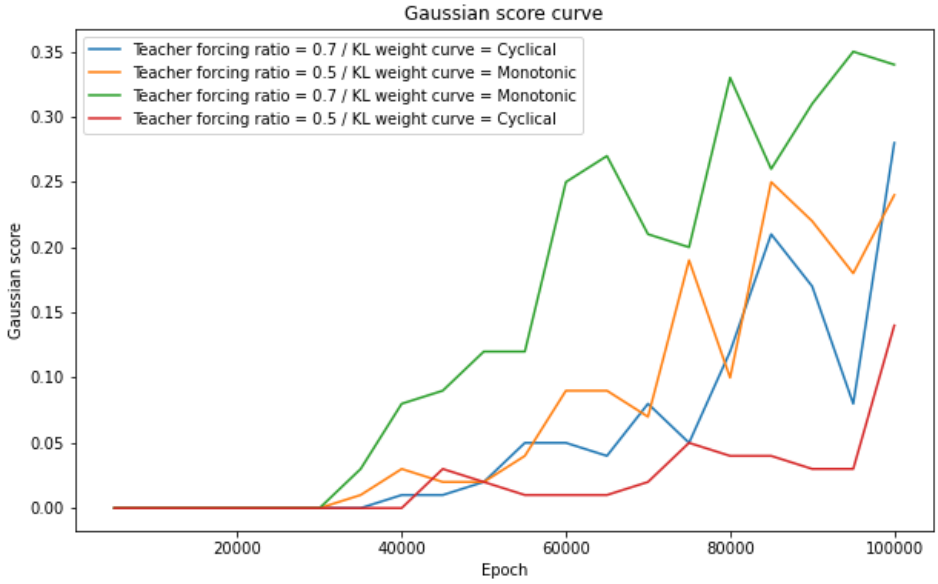




根據上述二張圖的結果，就 Loss 而言，Monotonic 的表現比 Cyclical 來得佳，且 Teacher forcing ratio 要設定的越高，模型表現效果也會越好。至於 KL Loss 的部分，因為 KL Loss 隨著訓練次數的增多會趨近於 0，因此四種模型並無顯著差異，只有訓練剛開始時，隨機起始位置的不同。

2. Gaussian score & BLEU-4 score





與前述的 Loss 的比較結果相同，一樣是 Monotonic 的表現比 Cyclical 來得佳，且 Teacher forcing ratio 要設定的越高。推測 Teacher forcing ratio 要設定高，模型會表現比較好的原因，應該是因為以往若過度使用 Teacher forcing，可能會導致我們的 model 在遇到沒看過的測資時，會無法準確的預測結果。但因為在此次 LAB 中，我們用來測量 BLEU-4 score 的 testing data，都是包含在我們 training data 中的單字，因此比較不會發生 Teacher forcing 過度使用時所產生的問題

最後，我們印出我們表現最好的 model （Teacher forcing ratio = 0.7/ KL weight curve = Monotonic）的結果：

上圖上方的字串陣列，就是我們 model 預測出來的結果 (random 100 words with 4 tenses)，結果顯示 Gaussian score 為 0.34。

右圖則是測試 BLEU-4 score 的結果，上面有詳細印出 input、target（我們希望印出的結果）、prediction（model 預測的結果）的 words。可以發現 target 和 prediction 大致相近，且整體的平均 BLEU-4 score 為 0.74 左右。