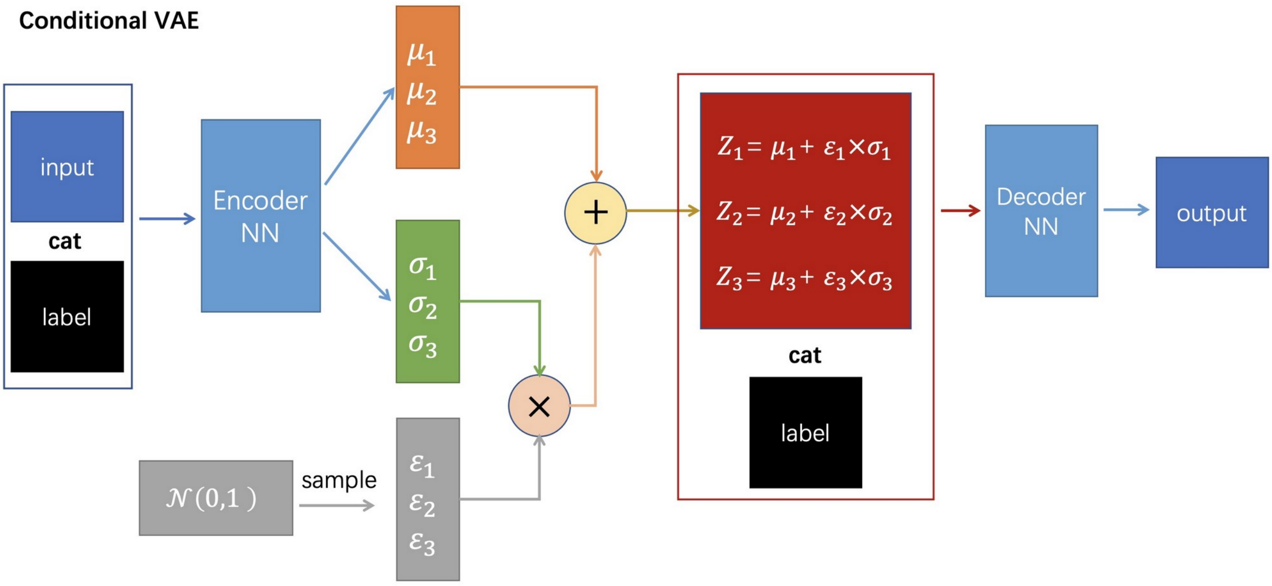
DLP Lab5 Conditional Sequence-to-Sequence VAE

## 1. Introduction

每個英文動詞皆有不同的時態，本 Lab 利用 Conditional Sequence-to-Sequence VAE 對英文動詞做時態的轉換處理，每組 training pair 有 4 種不同的時態，simple present(sp), third person(tp), present progressive(pg)和 simple past(p)，在進入 encoder 和 decoder 時會與 data 合併在一起。下圖為 CVAE 的架構:




## 2. Derivation of CVAE

encoder decoder

$$X \rightarrow q_\phi(z|X) \rightarrow z \rightarrow P_\theta(X|z)$$

$$q_\phi(z|X) = N(P_\phi(X), \Sigma_\phi(X))$$

$$D_{KL}[q_\phi(z|X) \,||\, P_\phi(z)] = D_{KL}[N(P_\phi(X), \Sigma_\phi(X)), N(0,1)]$$

$$= \frac{1}{2}\left[\underbrace{Tr(\Sigma_\phi(X))}_{\text{sum of } \Sigma_\phi(X)} + P_\phi(X)^T P_\phi(X) - \underbrace{k}_{\text{dim of Gaussian}} - \log|\underbrace{\Sigma_\phi(X)}_{\text{diagonal matrix}}|\right]$$

$$= \frac{1}{2}\sum_k \left[\Sigma_\phi(X) + P_\phi^2(X) - 1 - \log \Sigma_\phi(X)\right]$$

$$L(\theta,\phi) = -\underset{z \sim q_\phi(z|X)}{E}(\log P_\phi(X|z)) + \frac{1}{2}\sum_k(\Sigma_\phi(X) + P_\phi^2(X) - 1 - \log \Sigma_\phi(X))$$

$$N(\mu, ae^\sigma) = \mu + \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \cdot e^\sigma$$

$$\downarrow$$

$$N(0,1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

3. Implementation details

在 CVAE 中，主要由三個部份組成，Encoder，中間的 sample part 以及 Decoder，餵 data x 進 encoder 會產生出 latent vector z，然後將 z 餵進 decoder 會產生 output y。

- Encoder

  在 Encoder 中，會先把英文單字 embedding 成一個多維向量，透過 nn.LSTM，最後輸出 output, hidden_state 以及 cell_state。這裡的 input_size 為 28（SOS, EOS, a-z），hidden_size 為 256。

```python
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        """
        :param input_size: 28 (containing:SOS,EOS,a-z)
        :param hidden_size: 256 or 512
        """
        super(VAE.EncoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.rnn = nn.LSTM(hidden_size, hidden_size)

    def forward(self, input, hidden_state, cell_state):
        """forwarding an alphabet (batch_size here is 1)
        :param input: tensor
        :param hidden_state: (num_layers*num_directions=1,batch_size=1,vec_dim=256)
        :param cell_state: (num_layers*num_directions=1,batch_size=1,vec_dim=256)
        """
        embedded = self.embedding(input).view(1,1,-1)  # view(1,1,-1) due to input of rnn must be (seq_len,batch,vec_dim)
        output, (hidden_state, cell_state) = self.rnn(embedded, (hidden_state, cell_state))
        return output, hidden_state, cell_state
```

- 中間的 sample part

  在 VAE 中，latent vector 的分佈是 multivariate Gaussian distribution，所以這邊透過 fully connected layer 變成 32 dimension 的 mean 和 log variance，這裡取 log variance 是因為在定義上，variance 皆為正值，但 fully connected layer 可能會輸出負值。

  有了 mean 和 log variance 後，就可以透過以下公式 reparameterization trick sample 一個 32 dimension 的 latent vector，這個 latent vector 再與 condition concatenate 後，再透過一個 fully connected layer 轉為 hidden_state 的維度。

$$z = z^* * \exp(logvar/2) + mean$$

```python
"""
middle part forwarding
"""
mean = self.hidden2mean(encoder_hidden_state)
logvar = self.hidden2logvar(encoder_hidden_state)
# sampling a point
latent = self.reparameterize(mean,logvar)
decoder_hidden_state = self.latentcondition2hidden(torch.cat((latent, c), dim=-1))
decoder_cell_state = self.decoder.init_c0()
decoder_input = torch.tensor([[SOS_token]], device=device)
```

- Decoder

  seq2seq 中，decoder 需要先前中間 sample part 的輸出來決定現在的 output。在 CVAE 中，decoder decode latent vector z 和 condition c 來得到 output。所以這邊輸入的 hidden_state 為先前中間 sample part 的輸出，cell_state 則初始化為 0 tensor。

```python
class DecoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(VAE.DecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.rnn = nn.LSTM(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, input_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden_state, cell_state):
        """forwarding an alphabet
        """
        output = self.embedding(input).view(1, 1, -1)
        output = F.relu(output)
        output, (hidden_state, cell_state) = self.rnn(output, (hidden_state, cell_state))
        output = self.softmax(self.out(output[0]))
        return output, hidden_state, cell_state
```

- reparameterization trick

  從 Gaussian(mean, exp(log variance))中 sample 出一個點。

```python
def reparameterize(self, mean, logvar):
    """reparameterization trick
    """
    std = torch.exp(0.5 * logvar)
    eps = torch.randn_like(std)
    latent = mean + eps * std
    return latent
```

- Dataloder

  為了方便模型做訓練，所以使用 nn.embedding()，這邊將 SOS, EOS, a - z，分別對應到 0 – 27，共 28 種類別。

```python
class DataTransformer:
    def __init__(self):
        self.char2idx=self.build_char2idx()  # {'SOS':0,'EOS':1,'a':2,'b':3 ... 'z':27}
        self.idx2char=self.build_idx2char()  # {0:'SOS',1:'EOS',2:'a',3:'b' ... 27:'z'}
        self.tense2idx={'sp':0,'tp':1,'pg':2,'p':3}
        self.idx2tense={0:'sp',1:'tp',2:'pg',3:'p'}
        self.max_length=0  # max length of the training data word(contain 'EOS')

    def build_char2idx(self):
        dictionary={'SOS':0,'EOS':1}
        dictionary.update([(chr(i+97),i+2) for i in range(0,26)])
        return dictionary

    def build_idx2char(self):
        dictionary={0:'SOS',1:'EOS'}
        dictionary.update([(i+2,chr(i+97)) for i in range(0,26)])
        return dictionary
```

```python
def get_dataset(self,path,is_train):
    """
    :returns:
    if(train):  words=[w1,w2,w3,.....,wn], tenses:[0,1,2,3,0,1,2,3...]
    if(test):  words=[[w1,w2],[w3,w4]....[wn-1,wn]], tense:[[sp,p],[sp,pg]...,[pg,tp]]
    """
    words=[]
    tenses=[]
    with open(path,'r') as file:
        if is_train:
            for line in file:
                words.extend(line.split('\n')[0].split(' '))
                tenses.extend(range(0,4))
        else:
            for line in file:
                words.append(line.split('\n')[0].split(' '))
            test_tenses=[['sp','p'],['sp','pg'],['sp','tp'],['sp','tp'],['p','tp'],['sp','pg'],['p','sp'],['pg','sp'],['pg','p'],['pg','tp']]
            for test_tense in test_tenses:
                tenses.append([self.tense2idx[tense] for tense in test_tense])
    return words,tenses
```

```python
def __len__(self):
    return len(self.words)

def __getitem__(self, idx):
    """
    :returns:
    if(train): word: (time1,1) tensor, tense: (1) tensor
    if(test): word1: (time1,1) tensor, tense1: (1) tensor, word2: (time2,1) tensor, tense2: (1) tensor
    """
    if self.is_train:
        return self.string2tensor(self.words[idx],add_eos=True),self.tense2tensor(self.tenses[idx])
    else:
        return self.string2tensor(self.words[idx][0],add_eos=True),self.tense2tensor(self.tenses[idx][0]),\
                self.string2tensor(self.words[idx][1],add_eos=True),self.tense2tensor(self.tenses[idx][1])

def get_max_length(self,words):
    max_length=0
    for word in words:
        max_length=max(max_length,len(word))
    return max_length
```

- Text generation by Gaussian noise

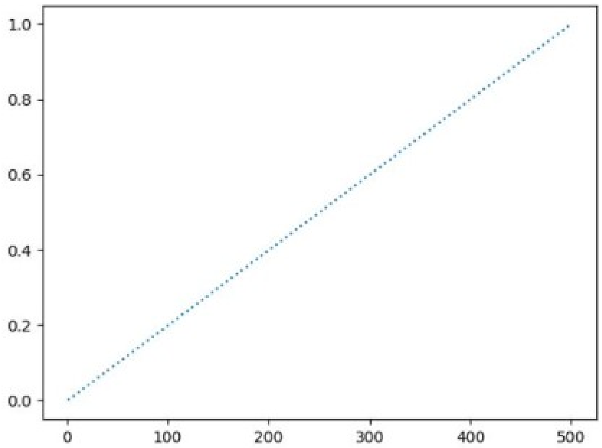
  用 torch.randn()隨機產生一個 32 dimension 的 latent vector z, 再把這個 latent vector 與 tense concatenate, 並作為 decoder 的 hidden_state 餵入。

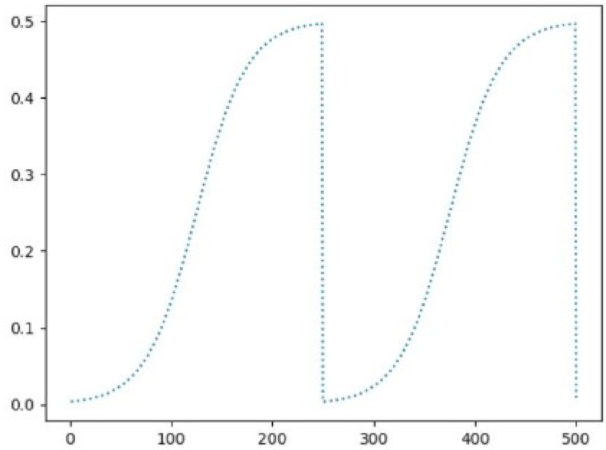
```python
def generateWord(vae, latent_size, tensor2string):
    vae.eval()
    re = []
    with torch.no_grad():
        for i in range(100):
            latent = torch.randn(1, 1, latent_size).to(device)
            tmp = []
            for tense in range(4):
                word = tensor2string(vae.generate(latent, tense))
                tmp.append(word)

            re.append(tmp)
    return re
```

```python
def generate(self, latent, tense):
    """
    :param latent: (1,1,latent_size) tensor
    :param tense: 0~3 int
    :return predict_output: (predict_output_length,1) tensor   (very likely contain EOS)
    """
    tense_tensor = torch.tensor([tense]).to(device)
    c = self.tense_embedding(tense_tensor).view(1, 1, -1)
    decoder_hidden_state = self.latentcondition2hidden(torch.cat((latent, c), dim=-1))
    decoder_cell_state = self.decoder.init_c0()
    decoder_input = torch.tensor([[SOS_token]], device=device)

    """
    decoder forwarding
    """
    predict_output = None
    for di in range(self.max_length):
        output, decoder_hidden_state, decoder_cell_state = self.decoder(decoder_input, decoder_hidden_state,
                                                                        decoder_cell_state)
        predict_class = output.topk(1)[1]
        predict_output = torch.cat((predict_output, predict_class)) if predict_output is not None else predict_class

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

    return predict_output
```

- KL weight

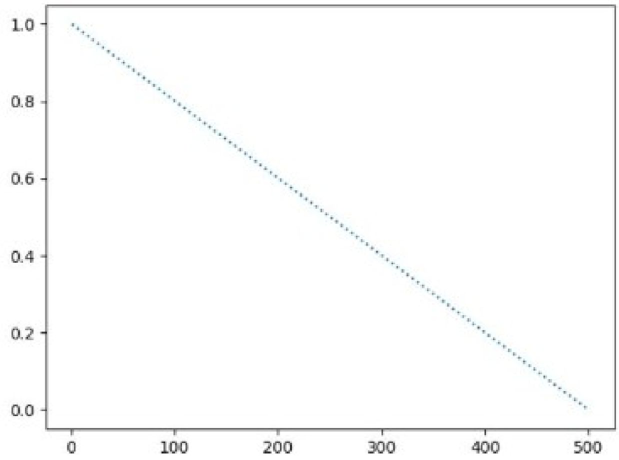
  有兩種，一個是 monotonic 的，一個是 cycle 的。




```python
def get_kl_weight(epoch, epochs, kl_annealing_type, time):
    """
    :param epoch: i-th epoch
    :param kl_annealing_type: 'monotonic' or 'cycle'
    :param time:
        if('monotonic'): # of epoch for kl_weight from 0.0 to reach 1.0
        if('cycle'):     # of cycle
    """
    if kl_annealing_type == 'monotonic':
        return (1./(time-1))*(epoch-1) if epoch<time else 1.

    else: #cycle
        period = epochs//time
        epoch %= period
        KL_weight = sigmoid((epoch - period // 2) / (period // 10)) / 2
        return KL_weight
```

- teacher forcing ratio
  是一個隨著 epoch 變大，由 1 線性遞減至 0 的函數。

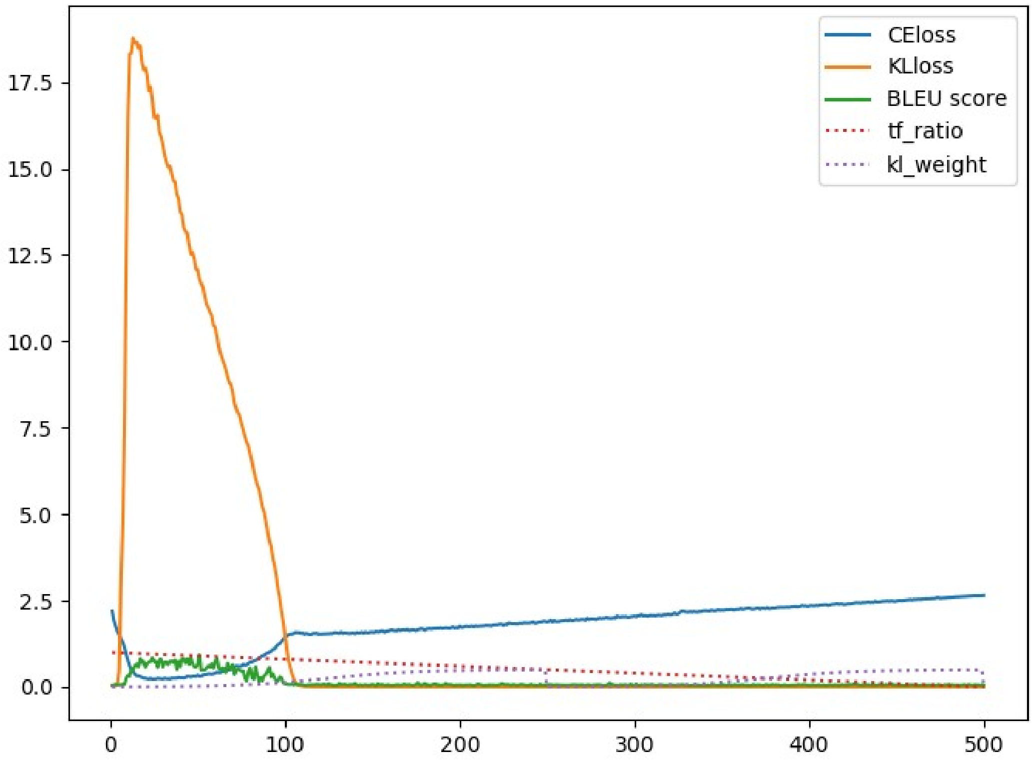



```
def get_teacher_forcing_ratio(epoch, epochs):
    # from 1.0 to 0.0
    teacher_forcing_ratio = 1.-(1./(epochs-1))*(epoch-1)
    return teacher_forcing_ratio
```

- learning rate : 0.05
- epochs : 500

4. Results and discussion

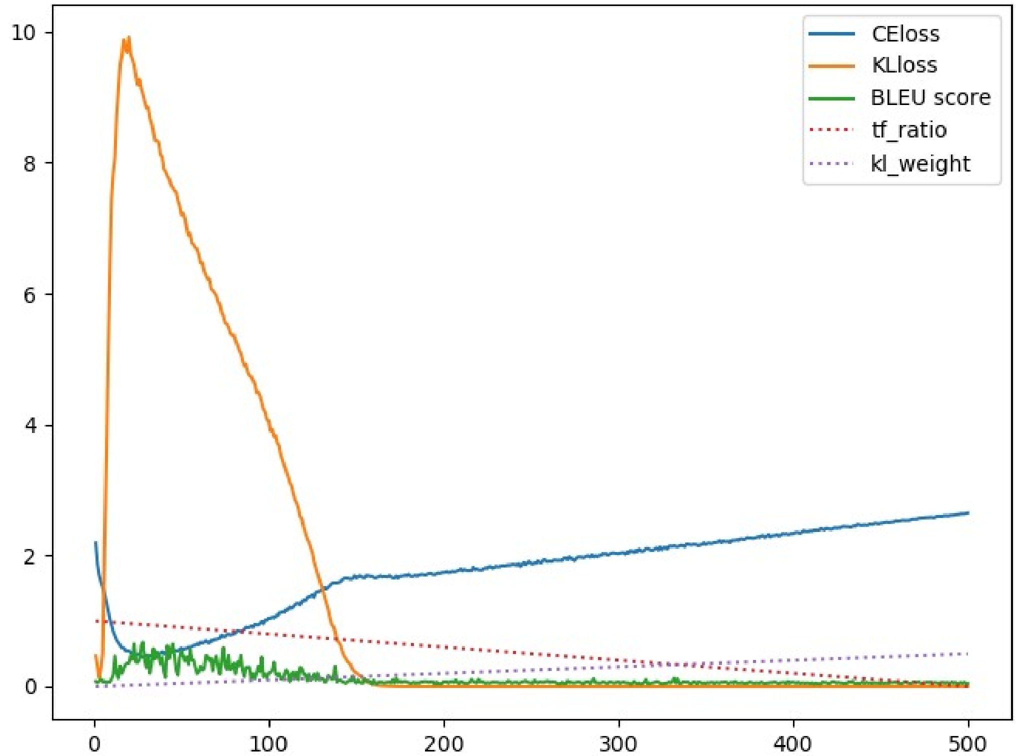
- cycle KL weight
  train 500 個 epochs，teacher forcing ratio 隨著 epoch 由 1 線性遞減至 0
  第 1 到 250 個 epoch 與第 251 到 500 的 epoch 間 KL weight 由 sigmoid 的方式上升至 0.5



- monotonic KL weight

  train 500 個 epochs， teacher forcing ratio 隨著 epoch 由 1 線性遞減至 0

  KL weight 從 0 線性上升至 1




- test.txt prediction

```
[['abandon', 'abandoned', 'abandoned'], ['abet', 'abetting', 'abetting'], ['begin', 'begins', 'begs'],
```

- Gaussian noise distribution 生成 word

```
['prefer', 'prefers', 'preferting', 'progressed'], ['abet', 'belts', 'belting', 'belted'], ['glint', 'glints', 'glinting', 'glinted']
```

- test 10 次的 average BLEU score 和 Gaussian score

```
avg BLEUscore 0.73
avg Gaussianscore 0.33
```

- Discuss

  起初， KL weight 很小， 造成 KL loss 變得很高， CE loss 變得很低， 當 KL weight 慢慢變大時， KL loss 也開使下降， CE loss 開始上升， 原因是 KL weight 變大， KL weight * KL loss 開始 dominate 整個 loss function， 所以 KL loss 會被迫往下降， CE loss 因而上升連帶影響 BLEU score 下降。由圖中可以發現， 發生分數高的地方 KL loss 很高且 CE loss 很低。所以在後面的 epoch， 由於 KL weight 一直上升， 使得 KL loss 一直都很低， BLEU score 也因此無法提高。