NCTU DLP Lab5-Report
Conditional Seq2seq VAE

廖家鴻 0786009
2020/5/12

Outline

- **Introduction**
- **Derivation of CVAE**
- **Implementation Details**
  - A. Dataloader
  - B. Encoder
  - C. Decoder
  - D. Reparameterization Trick
  - E. Gaussian Word Generation
  - H. KL-Annealing Schedule
- **Results and Discussion**
  - A. Crossentropy- and KL- loss curve
  - B. BLUE-4 score testing curve
  - C. Tense Conversion
  - D. Word Generation

Introduction

- **Derivation of CVAE**
- **Implementation Details**
  A. Dataloader
  B. Encoder
  C. Decoder
  D. Reparameterization Trick
  E. Gaussian Word Generation
  H. KL-Annealing Schedule
- **Results and Discussion**
  A. Crossentropy- and KL- loss curve
  B. BLUE-4 score testing curve
  C. Tense Conversion
  D. Word Generation

首先會將CVAE的objective function
推導的過程與結果show出來。

接著在在Implementation中，
會截code圖呈現&說明實作細節。

在Result and Discussion中，會呈現：

1. Crossentropy- and KL-loss curve並討論Annealing schedule的影響，及相關Hyperparameters。
2. 會比較monotonic與cyclical schedule對BLEU-4可能的影響。
3. Show出某次training中最佳的時態轉換結果。
4. Show出某次training中最佳的gaussian noise for word generation結果。

Derivation of CVAE

和原本的VAE一樣

$P(x|c;\theta)=\int p(x|z,c;\theta)p(z)dz$ 是intractable

所以也要採用 EM algorithm，並且從 chain rule 可得：

$$\log P(X|c;\theta) = \log P(X,Z|C;\theta) - \log p(Z|X,c;\theta)$$

在上式等號兩邊同乘以 $q(z)$，並對z積分：

$$\int q(z)\log p(X|c;\theta)\,dz$$
$$= \int q(z)\log p(x,Z|c;\theta)\,dz - \underline{\int q(z)\log q(z)\,dz}$$
$$+ \underline{\int q(z)\log q(z)\,dz} - \int q(z)\log p(z|X,c;\theta)\,dz$$

$$\Rightarrow \log p(x|c;\theta) = \mathcal{L}(X,c,q,\theta) + KL(q(z)\,\|\,p(z|X,c;\theta))$$

其中 $\begin{cases} \mathcal{L}(X,c,q,\theta) = \int q(z)\log p(X,Z|c;\theta)\,dz - \int q(z)\log q(z)\,dz \\ KL(q(z)\,\|\,p(z|X,c;\theta)) = \int q(z)\log \dfrac{q(z)}{p(z|X,c;\theta)}\,dz \end{cases}$

又 $\because$ KL-divergence ≥ 0 $\therefore \log p(X|c;\theta) \geq \mathcal{L}(X,c,q;\theta)$

並且 if and only if $q(z)=p(z|X,c;\theta)$，則上式等號成立

因此 $\mathcal{L}(X,c,q;\theta)$ 是 $p(z|X,c;\theta)$ 的 lower bound.

sample出來的distribution

接著將 $q(z|X,c;\theta')$ 代入 $\mathcal{L}(X,c,q;\theta)$：

註: $p(x,z|c)=p(z|c)p(x|z,c)$

$$\mathcal{L}(X,C,q,\theta) = E_{z\sim q(z|X,c;\theta')}\log p(x|z,c;\theta)$$
$$+ E_{z\sim q(z|X,c;\theta')}\log p(z|c) - E_{z\sim q(z|X,c;\theta')}\log q(z|X,c;\theta')$$
$$= E_{z\sim q(z|X,c;\theta')}\log p(X|z,c;\theta) - KL(q(z|X,c;\theta')\,\|\,p(z|c))$$ #

得證

Implementation Details

Implementation Details-Detail of Dataloader-1

```python
def getData(mode):
    if mode == 'train':
        train_wds = pd.read_table('dataset/train.txt',sep=' ',header=None)
        train_wds = train_wds.values
        inputs = []
        tense = []
        inputs_len = []
        for i in range(train_wds.shape[0]):
            for j in range(train_wds.shape[1]):
                w = train_wds[i,j]
                inputs.append(w)
                inputs_len.append(len(w))
                tense.append(j)
        return np.array(inputs), np.array(tense), np.array(inputs_len)
    else:
        test_wds = pd.read_table('dataset/test.txt',sep=' ',header=None)
        test_wds = test_wds.values

        inputs = test_wds[:,0]
        inputs_tense = test_wds[:,2]
        targets = test_wds[:,1]
        targets_tense = test_wds[:,3]
        return inputs, inputs_tense, targets, targets_tense

x_train, x_tense, x_train_len = getData('train')
x_test, x_test_tense, y_test, y_test_tense = getData('test')
max_seq_len = np.max(x_train_len) +5
```

test - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
abandon abandoned 0 3
abet abetting 0 2
begin begins 0 1
expend expends 0 1
sent sends 3 1
split splitting 0 2
flared flare 3 0
functioning function 2 0
functioning functioned 2 3
healing heals 2 1
```

左圖即為將txt檔用pandas讀進來先成為DataFrame再.values()成為numpy array type的code。

由於training時，input和target的字是一樣的，所以(x_train, x_tense)同時是model input & target。

至於test data， input和target的word 時態是不一樣的，所以分別取出x_test和y_test(註: test.txt檔我有手動加上時態index如上圖)

我同時也把各word的length算出來，從max_seq_len的搜尋，發現word最長的有15個字母，然後我刻意加0到總sequence長度為20。並且可用於training的word共有1227x4=4908個

Implementation Details-Detail of Dataloader-2

```
vocab_size = 28
SOS_token = 0
EOS_token = 27
```

```python
characters = ' '+string.ascii_lowercase

def letter2index(letter):
    return characters.find(letter)

def word2seqToken(line, eos=True):
    ary = np.zeros(len(line))
    tensor = torch.LongTensor(ary)
    eos_tensor = torch.LongTensor([EOS_token])
    for li, letter in enumerate(line):
        tensor[li] = letter2index(letter)
    if eos:
        return torch.cat((tensor,eos_tensor))
    else:
        return tensor

def formPair(x_train, x_tense, y_train, y_tense):
    w_inp_list = []
    t_inp_list = []
    w_tar_list = []
    t_tar_list = []
    for i in range(len(x_train)):
        w_inp = word2seqToken(x_train[i])
        t_inp = x_tense[i]
        w_inp_list.append(w_inp)
        t_inp_list.append(t_inp)

        w_tar = word2seqToken(x_train[i])
        t_tar = y_tense[i]
        w_tar_list.append(w_tar)
        t_tar_list.append(t_tar)
    return list(zip(zip(w_inp_list,t_inp_list),zip(w_tar_list,t_tar_list)))
```

Vocabulary size則設為28，因為有26個字母加上sos、eos兩個token。

左圖即為將words逐步轉成token sequence pairs的functions，同時data type轉成torch tensor。並且zip成[(輸入字,時態),(輸出字,時態)]的pair。

而training_pairs與test_pairs均由formPair()這個function產生，然後這pairs objects的indexing dimension詳情如下圖：

```python
training_pairs = formPair(x_train, x_tense, x_train, x_tense)
test_pairs = formPair(x_test, x_test_tense, y_test, y_test_tense)
'''
training_pairs[i][j][k]
i-dim=0,1: index for [(input,input_tense),(target,target_tense)] pair
j-dim=0,1: index for (input,target) pair
k-dim=0,1: index for (word_tensor, tense) pair
'''
random.shuffle(training_pairs)
```

在training_pairs用random.shuffle是為了在後面training時，每個epoch的input order都隨機打亂而不一樣。

另外，我依個人喜好將SOS設成0，EOS設成27，中間的1~26前為那26個小寫字母。

Implementation Details-Detail of Encoder

```python
###### Encoder
class EncoderRNN(nn.Module):
    def __init__(self, vocab_size, hidden_size, latent_dim, n_tense_embed):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.input_size = hidden_size #為word embedding後的dim
        # self.n_layers = n_layers
        self.latent_dim = latent_dim
        self.n_tense_embed = n_tense_embed

        self.embedding = nn.Embedding(vocab_size, hidden_size)
        self.lstm = nn.LSTM((self.input_size+n_tense_embed), hidden_size)

    def forward(self, input1, input2, hn, cn):
        c_embed = input1.view(1, 1, -1)
        embedded = self.embedding(input2)
        wd_embed = embedded.view(1, 1, -1)  # [batch, seq, embedding_size]
        c_wd_inp = torch.cat((c_embed, wd_embed), dim=2)

        output,(hn,cn) = self.lstm(c_wd_inp, (hn,cn))
        return output, (hn,cn)

    def initHidden(self):
        # 各个维度的含义是 (Seguence, minibatch_size, hidden_dim)
        return torch.zeros(1, 1, (self.hidden_size-self.n_tense_embed), device=device)
```

左圖即LSTM based的encoder class。

與GRU不同的是，LSTM有cell stat (cn)，跟hidden state (hn)。並且這兩類LSTM的hidden state的dimension在CVAE的應用場景時，要包含conditional的dimension，如左圖的：(hidden_size+n_tense_embed)。

而n_tense_embed即為word時態embedding加進來concatenate的dimension數。

Embedding layer在pytorch則是吃token值，不是onehot vector，然後會輸出長度為hidden_size的word vector，再餵進lstm裡進行forward。

Implementation Details-Detail of Decoder

```python
##### Decoder
class DecoderRNN(nn.Module):
    def __init__(self, hidden_size, vocab_size, latent_dim, n_tense_embed):
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.input_size = hidden_size #為word embedding後的dim
        self.latent_dim = latent_dim
        self.n_tense_embed = n_tense_embed

        self.latent_to_hidden = nn.Linear((latent_dim +n_tense_embed), (hidden_size +n_tense_embed))

        self.embedding = nn.Embedding(vocab_size, hidden_size)
        self.lstm = nn.LSTM((self.input_size+latent_dim +n_tense_embed), (hidden_size +n_tense_embed))
        self.out = nn.Linear((hidden_size +n_tense_embed), vocab_size)

    def forward(self, input1, input2, hn, cn):
        latent_z = input1.view(1, 1, -1)
        wd_embed = self.embedding(input2).view(1, 1, -1)
        wd_embed = F.relu(wd_embed)
        lz_wd_inp = torch.cat((latent_z, wd_embed), dim=2)

        output,(hn,cn) = self.lstm(lz_wd_inp, (hn,cn))
        output = self.out(output[0])
        return output, (hn,cn)

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

```python
decoder_hn = decoder.latent_to_hidden(latent_z)
decoder_cn = torch.zeros(1, 1, (256 +8), requires_grad=True).to(device)
```

左圖即LSTM based的decoder class。

在VCAE較不一樣的是因為有latent vector要輸入到decoder的LSTM的hn，所以建了latent_to_hidden這個net，並且在train CVAE時調用它來將latent_z輸入到decoder_hn

同時我也將latent_z也輸入到decoder的input，如左圖的綠框部分。

Implementation Details-Reparameterization Trick

```python
class Cond_Embed(nn.Module):
    def __init__(self, N_TENSE, COND_EMB_DIM, pad_id=0, dropout=0.1):
        super(Cond_Embed, self).__init__()
        self.embedding = nn.Embedding(N_TENSE, COND_EMB_DIM, padding_idx=pad_id)
        self.dropout = nn.Dropout(p=dropout)
    def forward(self, input):  # [batch, seq]
        tense_embed = self.embedding(input).view(1, 1, -1)
        tense_embed = self.dropout(tense_embed)
        return tense_embed  # [batch, seq, embedding_size]
```

```python
class Reparame(nn.Module):
    def __init__(self, COND_EMB_DIM, hidden_size, latent_length):
        super(Reparame, self).__init__()
        self.COND_EMB_DIM = COND_EMB_DIM
        self.hidden_size = hidden_size
        self.latent_length = latent_length

        self.hidden_to_mean = nn.Linear(self.COND_EMB_DIM +self.hidden_size, self.latent_length)
        self.hidden_to_logvar = nn.Linear(self.COND_EMB_DIM +self.hidden_size, self.latent_length)

    def forward(self, cell_output):
        self.latent_mean = self.hidden_to_mean(cell_output)
        self.latent_logvar = self.hidden_to_logvar(cell_output)

        std = torch.exp(self.latent_logvar / 2)
        eps = torch.randn_like(std)
        x_sample = eps.mul(std).add_(self.latent_mean)
        return self.latent_mean, self.latent_logvar, x_sample
```

```python
z_mu, z_var, x_sample = Reparame(encoder_hn)
x_sample.unsqueeze(0).unsqueeze(0)

c_tar = Cond_Embed(ts_tar)
latent_z = torch.cat((c_tar, x_sample), dim=2)

decoder_hn = decoder.latent_to_hidden(latent_z)
decoder_cn = torch.zeros(1, 1, (256 +8), requires_grad=True).to(device)
```

左邊最上圖的Cond_Embed即為word tense的conditional embedding 的net。

左邊中間圖Reparame則為Details-Reparameterization Trick的net。主要是要將encoder的final hidden state轉成mean、log_variance以及x_sample vector。

mean和logvar主要是for計算KL散度，而x_sample則是要跟tense的embedding合併成latent_z，然後再輸入成為decoder_hn的initial值。

Implementation Details-Gaussian Generation

```
###### Gaussian Generator
def Gaussian_Gen(decoder, Cond_Embed, max_seq_len, latent_dim):
    decoder.eval()
    Cond_Embed.eval()
    with torch.no_grad():
        decoder_words = []
        for i in range(100):
            x_sample = torch.randn(1, 1, latent_dim).to(device)
            decoder_cn = torch.zeros(1, 1, (256 +8)).to(device)
            wd4ts = []
            for j in range(4):
                ts_tar = torch.LongTensor([j]).to(device)
                c_tar = Cond_Embed(ts_tar)
                latent_z = torch.cat((c_tar, x_sample), dim=2)

                decoder_hn = decoder.latent_to_hidden(latent_z)
                decoder_input = torch.tensor([[SOS_token]], device=device)
                decoded_letters = []
                for di in range(16):
                    decoder_output, (decoder_hn,decoder_cn)= decoder(
                                latent_z, decoder_input, decoder_hn,decoder_cn)
                    topv, topi = decoder_output.data.topk(1)
                    if topi.item() == EOS_token:
                        break
                    else:
                        decoded_letters.append(characters[topi.item()])
                    decoder_input = topi.squeeze().detach()
                ##### token to word #####
                decoder_word = ''.join(decoded_letters)
                wd4ts.append(decoder_word)
            decoder_words.append(wd4ts)
        return decoder_words
```

左圖為decoder端的word generation function。

1. 首先在第一層i-loop，用torch.randn產生100個Gaussian noise生成的x_sample。
2. 接著在將x_sample在第二層j-loop中與四個tense的conditional embedding向量concatenate成latent_z
3. 然後將latent_z輸出成為decoder_hn的intitial
4. 最後進行各時態word生成。

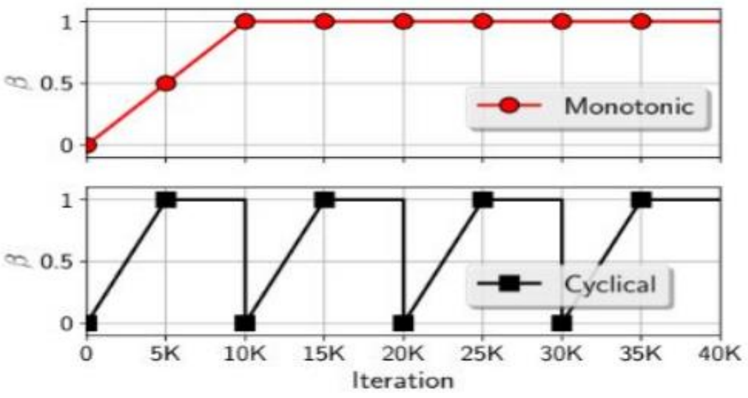
Implementation Details-KL Annealing Schedule

```python
######### KLD_Weight #########
KLD_max_weight = 0.33
def KL_anneal(mode, max_wet = KLD_max_weight):
    if mode == 'mono':
        linear = np.linspace(0, max_wet, 10000)
        horizon = np.linspace(max_wet, max_wet, 390000)
        KL_anneal_mono = np.hstack([linear,horizon])
        return KL_anneal_mono

    else:
        linear = np.linspace(0, max_wet, 10000)
        horizon = np.linspace(max_wet, max_wet, 10000)
        KL_anneal = np.hstack([linear, horizon])
        for i in range(20):
            if i==0:
                KL_anneal_cycl = KL_anneal
            else:
                KL_anneal_cycl = np.hstack([KL_anneal_cycl,KL_anneal])
        return KL_anneal_cycl
```

紅色框裡即為monotonic schedule，KLD_max_weight為0.33

黃色框裡即為cyclical schedule，KLD_max_weight一樣為0.33。而週期則改成20000次iterations

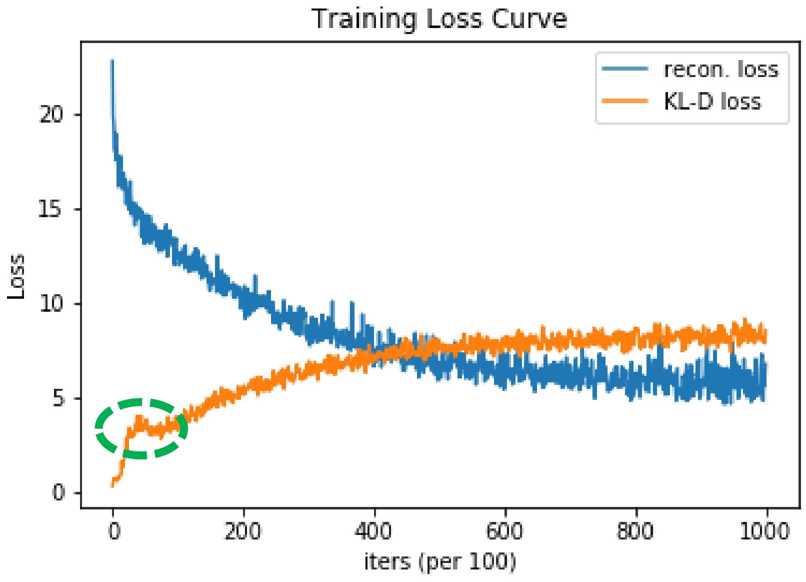


上圖即為KL annealing schedule的function，形狀的設計與助教的Spec裡的圖類似，唯高度(KLD_max_weight與週期有調整)

註：其它Hyperparameters會在後面的Results一起呈現出來討論。

Results and Discussion

Results and Discussion– Training Loss: Monotonic



左上圖是這次train s2s-CVAE的 reconstruction loss 及 KL-Divergence loss。

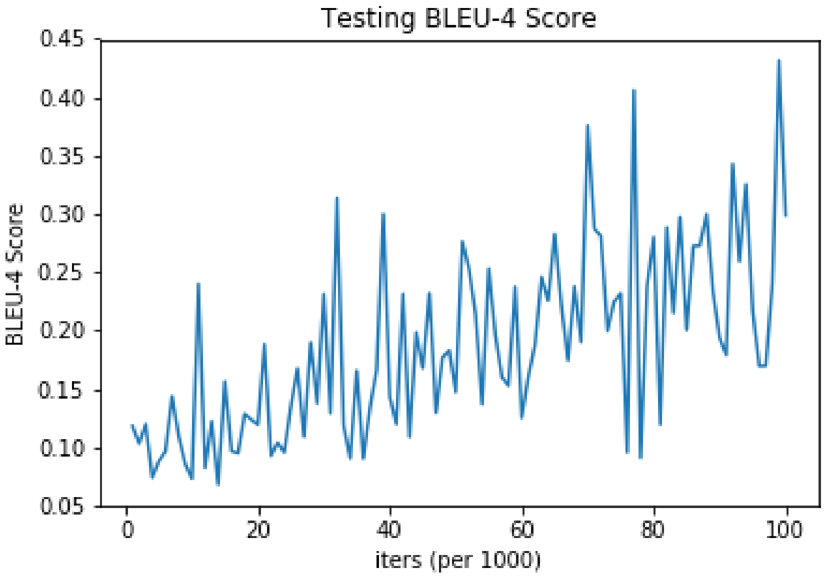
用monotonic KL-annealing schedule，會發現一開始KL loss很小，但後來就會超過recon. loss。

並且會發現在training初期，KL-loss會有個小peak。所以推測是在training初期讓KL-weight小，有助於recon.-loss收斂。

```
model_name = 'S2S_CVAE'
hidden_size = 256
N_TENSE = 4
COND_EMB_DIM = 8
LATENT_DIM = 32
epochs = 21
num_iters = len(x_train)  #4908
Tforce = np.linspace(0.5, 0.5, epochs+1)
KLD_max_weight = 0.33
ANL_mode = 'mono'
lr = 0.05
```

- Optimizer: SGD
- Reconstruction Loss: CrossEntropyLoss()

上圖是這次training的相關參數。
(每train完4908個word為一個epoch)

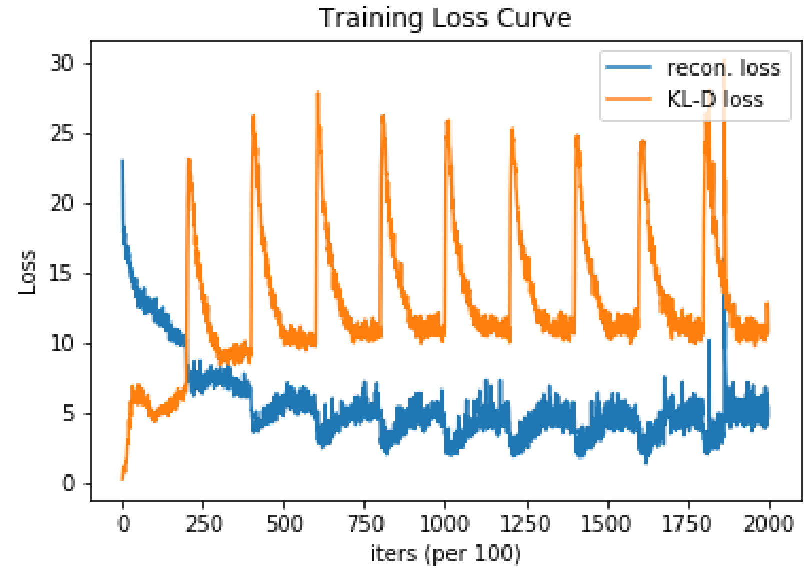


左下圖是test data在這次s2s-CVAE的BLEU-4 curve。

Train 10萬多次iteration後，BLEU-4 score最高到0.435

(雖然沒有比後面slide的cyclical schedule高，但monotonic的Gaussian_score平均較cyclical高。)

Results and Discussion– Training Loss: Cyclical

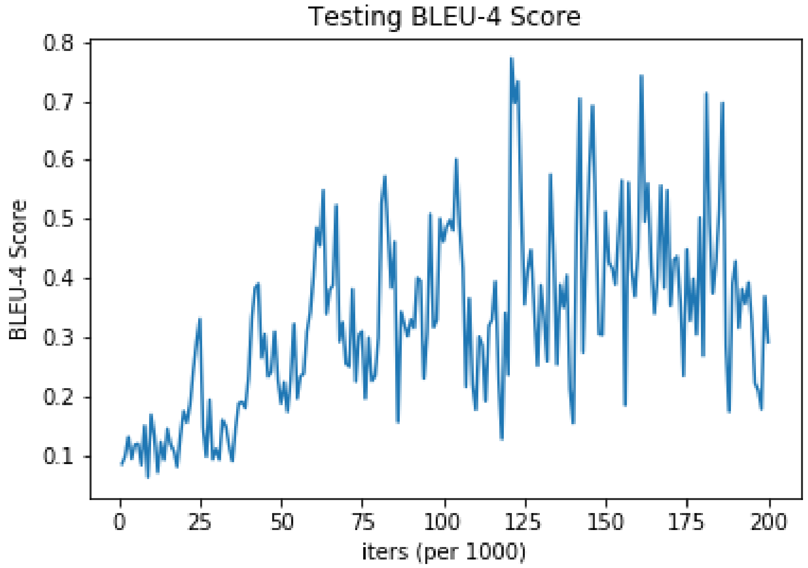


左上圖是這次train s2s-CVAE的 reconstruction loss 及 KL-Divergence loss。

用cyclical KL-annealing schedule，會發現當KL-weight變小時，KL-loss會變大，然同時recon.-loss會變小。

推測可能是因為KL-weight小時，model在train decoder的比重較高，試圖讓recons.-loss下降較多來得到更精確的output。然而此時卻會讓KL-loss上升，這意味著encoder的final hidden state出來的結果，noise更不穩定、會更不似Normal distribution N(0,1)。

所以試圖週期性循環，來看能不能達到同時兼顧encoder和decoder的性能。



左下圖是test data在這次s2s-CVAE的BLEU-4 curve。

Train 20萬多次iteration後，BLEU-4 score最高到0.786

(跟前一頁slide同只看前10萬次iters的話，cyclical schedule的BLEU_score整體較高，但實際上Gaussian_score平均另外比較起來較monotonic低。)

```
model_name = 'S2S_CVAE'
hidden_size = 256
N_TENSE = 4
COND_EMB_DIM = 8
LATENT_DIM = 32
epochs = 41
num_iters = len(x_train)  #4908
Tforce = np.linspace(0.5, 0.5, epochs+1)
KLD_max_weight = 0.33
ANL_mode = 'cycl'
lr = 0.05
```

- Optimizer: SGD
- Reconstruction Loss: CrossEntropyLoss()

Results and Discussion– Results of Tense Conversion

```
==========================
input:   abandon
target: abandoned
pred:    abandoned
==========================
input:   abet
target: abetting
pred:    abet
==========================
input:   begin
target: begins
pred:    begins
==========================
input:   expend
target: expends
pred:    expends
==========================
input:   sent
target: sends
pred:    sends
==========================
```

```
==========================
input:   split
target: splitting
pred:    split
==========================
input:   flared
target: flare
pred:    flare
==========================
input:   functioning
target: function
pred:    function
==========================
input:   functioning
target: functioned
pred:    functioned
==========================
input:   healing
target: heals
pred:    heals
==========================
BLEU-4 score:0.8817
```

左圖是Tense conversion，在某次training時最好的結果，但相應的Gaussian_score，卻只有0.07。

看起來似乎較高的teacher forcing ratio對BLEU score有正面影響，但對gaussian_score反而沒那麼好。

Results and Discussion– Gaussian Word Generation

```
['substitute', 'substitutes', 'substituting', 'substituted']
['arouse', 'arouses', 'arousing', 'aroused']
['arch', 'arraigns', 'arche', 'arched']
['acclaim', 'acclaims', 'acclaiming', 'acclaimed']
['flash', 'shakes', 'shaving', 'flashed']
['exhort', 'exhorts', 'exhorting', 'exhorted']
['recove', 'recoverts', 'recoving', 'recovered']
['athase', 'stammeres', 'stamming', 'stammed']
['sugge', 'sugges', 'sugging', 'sugged']
['contain', 'encuts', 'containing', 'encutted']
['retain', 'retains', 'retaining', 'retained']
['detouge', 'denotes', 'denotifying', 'denoted']
['suppose', 'supposes', 'supposing', 'supposed']
['desist', 'desists', 'desisting', 'desisted']
['sight', 'sifts', 'sifting', 'sifted']
['accuse', 'accuses', 'coughing', 'accused']
['forsag', 'fashes', 'faspening', 'fashed']
['requist', 'requists', 'requising', 'requisted']
['gash', 'gashes', 'surving', 'gazed']
['exhaule', 'exhaules', 'reluting', 'exhauled']
['dismount', 'dismounts', 'dismounting', 'dismounted']
['regulate', 'regulates', 'regulating', 'regulated']
['address', 'stares', 'staresting', 'stared']
['suspent', 'supplementeds', 'suspenting', 'suspented']
['regan', 'regains', 'reching', 'reched']
['desist', 'profects', 'preceding', 'refitted']
['read', 'dives', 'reading', 'readid']
['exhaust', 'exhausts', 'exhausting', 'exhauled']
['suspot', 'suspets', 'suspeting', 'suspeted']
['stammonize', 'stammers', 'stammonizing', 'stammed']
['request', 'requests', 'requesting', 'requested']
['reflect', 'reflects', 'reflecting', 'reflected']
['announce', 'cranks', 'enabling', 'announced']
```

```
['charge', 'crowdes', 'charging', 'crowded']
['rotate', 'rotates', 'rotating', 'rotated']
['repoin', 'estiments', 'exproting', 'projected']
['suffuse', 'suffuses', 'suffusing', 'suffused']
['exhash', 'exhashes', 'exhashing', 'exhashed']
['finist', 'distracts', 'insting', 'distracted']
['counsel', 'counsells', 'concluding', 'counselled']
['gooze', 'dominates', 'goozing', 'gouged']
['truggel', 'truggels', 'truggelling', 'truggelded']
['arrange', 'arranges', 'arranging', 'arranged']
['affert', 'stammers', 'stamming', 'stammered']
['expire', 'expires', 'expiring', 'expired']
['endude', 'endudes', 'endudging', 'enduded']
['restound', 'resuses', 'requinting', 'restounded']
['contran', 'contreaches', 'conventing', 'contended']
['suffure', 'suffures', 'surging', 'sufformed']
['replit', 'replects', 'precting', 'replected']
['appoit', 'appoits', 'apoiting', 'appoited']
['addire', 'addidds', 'addiding', 'addidd']
['prote', 'prots', 'protesting', 'protested']
['wrinkle', 'wrinkles', 'wrinkling', 'wrinkled']
['shout', 'shouts', 'showing', 'shouted']
['paul', 'paules', 'paulaying', 'pauled']
['steach', 'steems', 'steating', 'steemed']
['attache', 'excuses', 'attaching', 'excused']
['exhibe', 'exhibers', 'exhibling', 'exhibered']
['flank', 'flants', 'flanking', 'flanked']
['exhaust', 'encogns', 'exhausting', 'engated']
['proclip', 'proclips', 'procliping', 'proclipted']
['gue', 'rotates', 'rotating', 'rotated']
['rot', 'rots', 'roting', 'rothered']
['sumer', 'sumers', 'suaring', 'sumered']
['bestify', 'bests', 'bestighing', 'bestifed']
```

```
['thretche', 'threts', 'thretting', 'thretted']
['accompant', 'expands', 'accose', 'expanded']
['relient', 'relects', 'relienting', 'relected']
['rendew', 'rendes', 'rendering', 'rended']
['desish', 'designs', 'designing', 'desished']
['resule', 'results', 'reflecting', 'resulted']
['renot', 'renots', 'recoverting', 'renoted']
['couple', 'couples', 'coupling', 'coupled']
['puzzle', 'puzzles', 'pilling', 'puzzled']
['proffed', 'profies', 'proficiating', 'profied']
['exhibit', 'exhibits', 'recognizing', 'exhibited']
['summolate', 'summolashes', 'smacking', 'smacked']
['slash', 'smells', 'slashing', 'smelled']
['support', 'supports', 'soothing', 'shoothed']
['picture', 'pictures', 'complying', 'pictured']
['probe', 'probes', 'probing', 'probe']
['retain', 'retains', 'recognizing', 'recoverted']
['exhilarate', 'exhilarates', 'exhilaiming', 'exhilarated']
['retain', 'retains', 'retaining', 'recorded']
['excuse', 'excuses', 'excusing', 'excused']
['bethod', 'bets', 'betting', 'betted']
['attain', 'attains', 'attaigning', 'attained']
['strain', 'strains', 'strainting', 'strainted']
['stamment', 'stamments', 'staking', 'stammed']
['laben', 'launches', 'labending', 'launted']
['adomint', 'adomints', 'adoming', 'adomined']
['dismember', 'dismembers', 'dismembering', 'dismembered']
['confuse', 'sunges', 'counsenting', 'sung']
['fly', 'flickers', 'flying', 'flickered']
['loathe', 'loathes', 'loathing', 'loathed']
['conside', 'considens', 'considening', 'consided']
['conse', 'conserves', 'conserving', 'consided']
Gaussian score: 0.19
```

實驗結果：左圖是train monotonic時，Gaussian score出現最好的一次。

並且大概框出幾個明顯看起來正確的字組時態。

然後奇怪的是，較高的Gaussian_score，似乎BLEU不一定也比較高，像左圖這次相應的BLEU_score就還不到0.3

The End