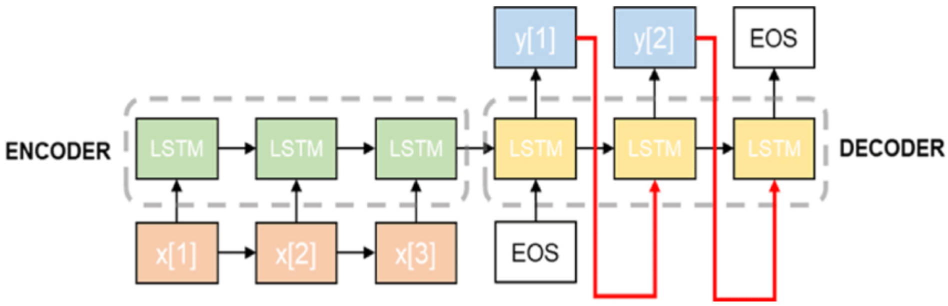
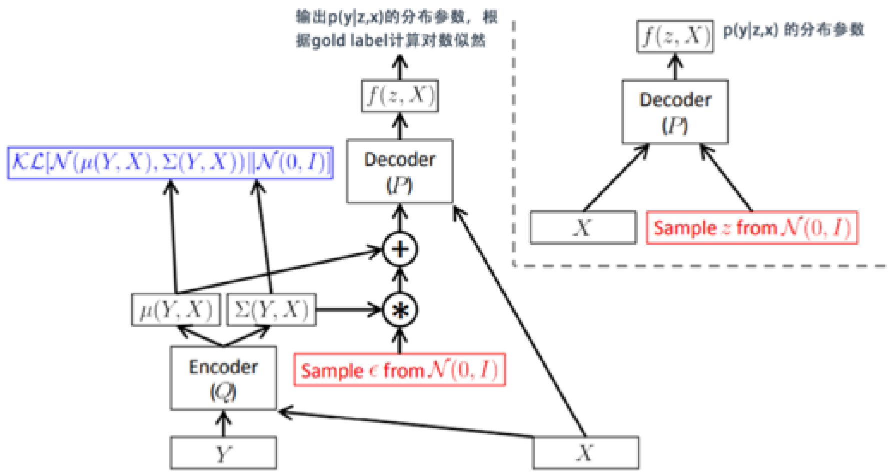
Deep Learning and Practice

#Lab05 Conditional Sequence-to-Sequence VAE　309505002 鄭紹文

1. Introduction：

此次lab要實作和NLP相關的實現，訓練一個對英文動詞時態的conditional seq2seq VAE，了解VAE以及CVAE的差別。每個英文動詞有四個限制的時態，分別是：simple present (sp), third person (tp), present progressive (pg), simple past (p)，而在訓練時要應用到reparameterization trick, teacher-forcing的概念以及kl loss annealing的方法來訓練模型，最後畫出cross-entropy loss和 KL loss在訓練過程中，最後由BLEU 4 score (from tense A to tense B)，和Gaussian score (Output the results generated by a Gaussian noise with 4 tenses)來對model評分。





2. Derivation of CVAE：

- <u>Derivation of CVAE</u>

在CVAE的testing(inference)中，所做的即是算後驗機率 $p(z|x)$ ← 不易求得.
so，在面對最佳化問題時，要找到一個 $q(z)$ 逼近 $p(z|x)$，故要找 2 個機率分布的距離，KL divergence 就很好用!!

$$KL(q(z)\,\|\,p(z|x)) = -\sum_z q(z)\log\frac{p(z|x)}{q(z)} = -\sum_z q(z)\left[\log\frac{P(x,z)}{q(z)} - \log P(x)\right]$$
$$= -\sum_z q(z)\log\frac{P(x,z)}{q(z)} + \log P(x).$$

$$\rightarrow \log P(x) = KL(q(z)\,\|\,p(z|x)) + \sum_z q(z)\log\frac{P(x,z)}{q(z)}.$$
$$= KL(q(z)\,\|\,p(z|x)) + \mathcal{L}(q)$$

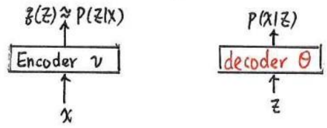
$p(x,z)$ 相對 $p(z|x)$ 好求! $\log P(x)$ 和 $q(z)$ 無關，又 $KL \geq 0$，$\underbrace{KL\downarrow = \mathcal{L}(q)\uparrow}$

max $\mathcal{L}(q)$ 使 $q(z) \approx p(z|x)$
但微分有期望值因素，
使 gradient 有誤差，
可用 reparameterization trick 解決.

VAE 中，
$$\mathcal{L}(v) = \mathbb{E}_{z\sim q}[\log p(x,z) - \log q(z;v)]$$
$$= \mathbb{E}_{z\sim q}[\log p(x|z) + \log p(x) - \log q(z;v)]$$
$$= \mathbb{E}_{z\sim q}[\log P(x|z)] + \mathbb{E}_{z\sim q}\left[\log\frac{P(z)}{q(z;v)}\right]$$
$$= \mathbb{E}_{z\sim q}[\log P(x|z)] - KL(q(z;v)\,\|\,p(z))$$

使 $P(x|z)$ 受 θ 調整.

$$\mathcal{L}(v,\theta) = \mathbb{E}_{z\sim q}[\log P(x|z,\theta)] - KL(q(z;v)\,\|\,p(z)).$$



EM中，希望找到一組 θ 使 marginal likelihood $\log P(x|\theta)$ max.

$$\tilde{\theta} = \arg\max_\theta \log P(x|\theta).$$

$$\log P(x|\theta) = KL(q(z)\,\|\,p(z|x,\theta)) + \sum_z q(z)\log\frac{P(x,z|\theta)}{q(z)}$$
$$= KL(q(z)\,\|\,p(z|x,\theta)) + \mathcal{L}(q,\theta)$$

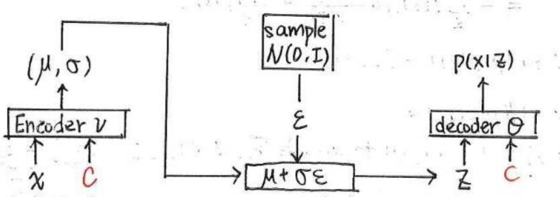
$\mathcal{L}(q,\theta)$ 是 lower bound，將 θ^{old} 代入：

$$\log P(x|\theta^{old}) = KL(q(z)\,\|\,p(z|x,\theta^{old})) + \mathcal{L}(p(z|x,\theta^{old}),\theta^{old})$$
$$= 0 + \mathcal{L}(p(z|x,\theta^{old}),\theta^{old}) \leq \max \mathcal{L}(p(z|x,\theta^{old}),\theta).$$

再求 $\theta^{new} = \arg\max_\theta \mathcal{L}(p(z|x,\theta^{old}),\theta)$ 即可再提高 lower bound.

CVAE 可控制產生 output by 給定條件，
加上 condition (c)！(推導和VAE一樣).

$$\log p(x|c) = KL(q(z|c)\,\|\,p(z|x,c)) + \sum q(x|c)\log\frac{p(x,z|c)}{q(z|c)}$$
$$\mathcal{L}(v,\theta|c) = \mathbb{E}_{z\sim q}[\log p(x|z,\theta,c)] - KL(q(z;v|c)\,\|\,p(z))$$



3. Implementation details：

A. Describe how you implement your model：

- Dataloader

因為英文字都由”字母”字元所組成，所以這部分先創造一個字母的dictionary，方便做字母←→數字的相對應轉換。同時define string 和longtensor之間的轉換。

```python
# 將英文字母投影到 0~25，SOS、EOD分別為 26, 27
class CharDict:
    def __init__(self):
        self.word2index = {} # 單詞 --> 索引  {"0":a, "1":b, ......}
        self.index2word = {} # 索引 --> 單詞  {"a":0, "b":1, ......}
        self.n_words = 0     # 累計

        for i in range(26):
            self.addWord(chr(ord('a') + i))

        tokens = ["SOS", "EOS"]
        for t in tokens:
            self.addWord(t)

    def addWord(self, word):
        # 判斷單詞是否已存在，如果不存在，加個，同時統計字符出現頻率
        if word not in self.word2index:
            self.word2index[word] = self.n_words  # 單詞對應的索引
            self.index2word[self.n_words] = word  # 索引對應的單詞
            self.n_words += 1                     # 索引加一

    def longtensorFromString(self, strs):
        strs = ["SOS"] + list(strs) + ["EOS"]     # strs 極為加上SOS EOS的總字串
        return torch.LongTensor([self.word2index[chars] for chars in strs])

    def stringFromLongtensor(self, line, show_token=False, check_end=True):
        strs = ""
        for i in line:
            chars = self.index2word[i.item()]
            if len(chars) > 1:       # len(SOS)=len(EOS)=3, len(正常字元)=1
                if show_token:
                    __chars = "<{}>".format(ch)    # SOS, EOS
                else:
                    __chars = ""
            else:
                __chars = chars    # __ch = a,b,c,d,......
            strs += __chars        # 組在一起
            if check_end and chars == "EOS":
                break
        return strs
```

這裡實際上做dataloader的任務，將train.txt以及test.txt讀入string，再轉換成longTensor。透過__getitem__()，training時可以拿到當前index、一個字(longTensor)以及時態條件(condition)，testing時可以拿到兩個字以及相對應的時態。

```python
# dataloader
class wordsDataset(Dataset):
    def __init__(self, train=True):
        if train:
            f = './train.txt'
        else:
            f = './test.txt'

        with open(f) as file:
            contents = file.read()

        self.words = contents.split()   # 讀字詞
        self.n_words = len(self.words)  # 總共幾個字詞

        del(contents)

        if not train:
            self.targets = np.array([
                [0, 3], #sp -> p
                [0, 2], #sp -> pg
                [0, 1], #sp -> tp
                [0, 1], #sp -> tp
                [3, 1], #p  -> tp
                [0, 2], #sp -> pg
                [3, 0], #p  -> sp
                [2, 0], #pg -> sp
                [2, 3], #pg -> p
                [2, 1], #pg -> tp
            ])

        self.tenses = [
            'simple-present', 'third-person', 'present-progressive', 'simple-past'
        ]

        self.chardict = CharDict()
        self.train = train

        for i in range(len(self.words)):
            self.words[i] = self.chardict.longtensorFromString(self.words[i])

    def __len__(self):
        if self.train:
            return self.n_words
        else:
            return len(self.targets)

    def __getitem__(self, index):
        if self.train:
            condition = index % len(self.tenses)    # condition：時態 (共四種 0~3), index抓值
            return index, self.words[index], condition
        else:   # testing
            i  = self.words[2*index]                # input字詞
            o  = self.words[2*index+1]
            condition_i = self.targets[index, 0]    # input的時態
            condition_o = self.targets[index, 1]

            return i, condition_i, o, condition_o
```

- **Encoder and Decoder**

CVAE中，encoder會透過input data x產生latent vector z，再透過decode產生目標的輸出y。先對時態條件(condition)做embed再將其和input data x連接。latent vector distribution是一個多維高斯分布，而透過encoder配合上reparameterization trick，會得到mean以及variance。Decoder會根據前一個輸出(這次輸入)來決定這次輸出。

```python
# Encoder
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(VAE.EncoderRNN,self).__init__()
        self.hidden_size = hidden_size
        self.embedding   = nn.Embedding(input_size, hidden_size) # embedding成詞向量
        self.lstm         = nn.LSTM(hidden_size, hidden_size)     # useing LSTM

    def forward(self, input, hidden_state, cell_state):
        # LSTM INPUT format  : (seq_length, batch_size, embedding_dim)
        # 先把形狀為 (batch_size, seq_length) 的 input 轉置，再把每個 value (char index) 轉成 embedding vector
        embedded = self.embedding(input).view(1, 1, -1)   # (seq_length(字母), batch_size)
        output, (hidden_state, cell_state) = self.lstm(embedded, (hidden_state, cell_state))
        return output, hidden_state, cell_state

    # Inputs: input, (h_0, c_0), shape :(seq_len, batch, input_size)
    # Initialize ho, co  (num_layers * num_directions, batch, hidden_size)
    def init_h0(self, size):
        return torch.zeros(1, 1, size, device=device)

    def init_c0(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
# Decoder
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.lstm = nn.LSTM(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, input_size)

          self.softmax = nn.LogSoftmax(dim=1)

    # output即 predict結果
    def forward(self, input_, hidden_state, cell_state):

        output = self.embedding(input_).view(1, 1, -1)
        output = F.relu(output)
        output, (hidden_state, cell_state) = self.lstm(output, (hidden_state, cell_state))
        output = self.out(output[0])
          output = self.softmax(self.out(output[0])) # 分類輸出
        return output, hidden_state, cell_state

    def init_h0(self):
        pass

    def init_c0(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
```

- **Reparameterization trick**

做最佳化時要找目標函數的 gradient 經過推倒後發現找出來 gradient 誤差很大，基本上做不了訓練(沒辦法透過 decoder 的 loss train encoder)，故使用 Reparameterization trick 解決這方面問題。首先先在 multivariate normal distribution ～N(0, 1)隨機採樣一個點 z*，再透過 z = z* ∗ exp(logvar/2) + mean 得到 z，這樣得到的 geadient 會小的 variance，就可以進行 BP 運算做訓練。

```python
# reparameterization trick
def reparameterize(self, mean, logvar):
    std    = torch.exp(0.5*logvar)
    z_star = torch.randn_like(std)  # 隨機採樣一個點 z*
    latent = mean + z_star*std      # z = z* *exp(logvar/2) + mean
    return latent
```

B. Specify the hyperparameters

- **KL weight**

Monotonic 的版本會根據 epoch 來決定數值，最後維持維一，而 cycle 版本我有時做了兩種，一種是如 pdf 上所說的週期波版本，另一種是經過調整之後用 sigmoid 的版本，原因是希望 KL weight 可以維持小的數值長一點時間。

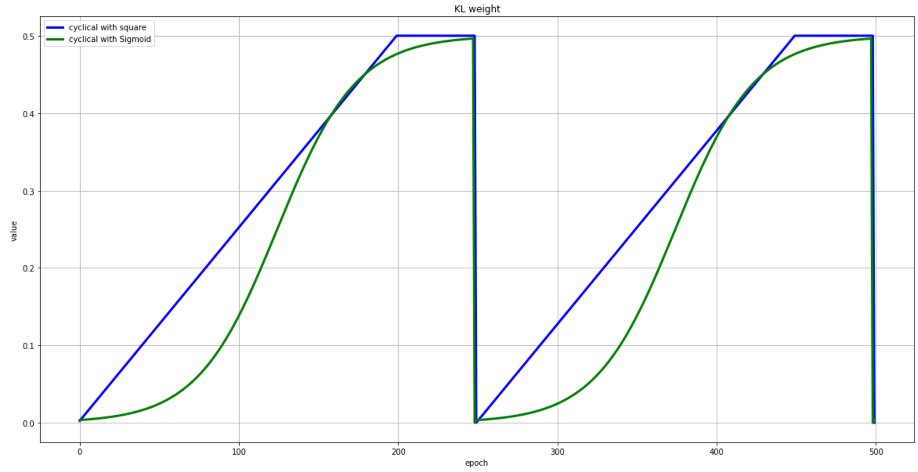
```python
def get_kl_weight(epoch, epochs, kl_annealing_type):

    assert kl_annealing_type=='monotonic' or kl_annealing_type=='cycle','kl_annealing_type not exist!'

    if kl_annealing_type == 'monotonic':
        if epoch < 50:
            return 0.02*epoch
        else:
            return 1
    else: #cycle
        period = epochs//2
        epoch %= period
        if epoch<200:
            return 0.0025*epoch
        else:
            return 0.5
```

```python
def get_kl_weight(epoch, epochs, kl_annealing_type):

    assert kl_annealing_type=='monotonic' or kl_annealing_type=='cycle','kl_annealing_type not exist!'
    if kl_annealing_type == 'monotonic':
        if epoch < 50:
            return 0.02*epoch
        else:
            return 1
    else: #cycle
        period = epochs//2
        epoch %= period
        if epoch < 249:
            return sigmoid((epoch-125)/25)/2
        else:
            return 0
```



- **Teacher forcing ratio**

讓teacher forcing ratio越來越小，希望一開始多利用teacher forcing，盡可能避免學習到錯誤的值，前面錯了，後面也完了，到後期就盡量降低依賴性。

```
def get_teacher_forcing_ratio(epoch):
    teacher_forcing_ratio = 1.-(0.0018*epoch)
```

- **Learning rate**：固定 0.05
- **Epoch**：300 or 500
- **KL annealing type : motononic or cyclical**

- **KL loss**：

$$
\begin{aligned}
&KL(N(u,\sigma^2)||N(0,1)) \\
&= \int \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-u)^2}{2\sigma^2}} \left(\log \frac{e^{-\frac{(x-u)^2}{2\sigma^2}} / \sqrt{2\pi\sigma^2}}{e^{\frac{-x^2}{2}} / \sqrt{2/pi}} \right) dx \\
&= \int \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-u)^2}{2\sigma^2}} \log \left(\frac{1}{\sqrt{\sigma^2}} \exp \frac{1}{2} (x^2 - \frac{(x-u)^2}{\sigma^2}) \right) dx \\
&= \int \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-u)^2}{2\sigma^2}} \frac{1}{2} \left(-\log \sigma^2 + x^2 - \frac{(x-u)^2}{\sigma^2} \right) dx \\
&= \frac{1}{2}(-\log \sigma^2 + u^2 + \sigma^2 - 1)
\end{aligned}
$$

```
def loss_function(predict_distribution, predict_output_length, target, mu, logvar):
    criterion = nn.CrossEntropyLoss()
    CEloss = criterion(predict_distribution[:predict_output_length], target[1:predict_output_length+1])
    #如果字母原本7個, 前面ditribution 0~7, length =8,

    # KL(N(mu,variance)||N(0,1))
    KLloss = -0.5 * torch.sum(1 + logvar - mu**2 - logvar.exp())
    return CEloss, KLloss
```

4. Results and Discussion：

▫ Show the results of tense conversion and generation

- **BLEU-4 score：**

```
--------------------------
input word  :abandon
target word :abandoned
predict word:abandoned
--------------------------
input word  :abet
target word :abetting
predict word:abetting
--------------------------
input word  :begin
target word :begins
predict word:begins
--------------------------
input word  :expend
target word :expends
predict word:enacts
--------------------------
input word  :sent
target word :sends
predict word:sends
--------------------------
input word  :split
target word :splitting
predict word:splitting
--------------------------
input word  :flared
target word :flare
predict word:flare
--------------------------
input word  :functioning
target word :function
predict word:function
--------------------------
input word  :functioning
target word :functioned
predict word:functioned
--------------------------
input word  :healing
target word :heals
predict word:heals

BLEU score :  0.9080876486277946
```

- **Gaussian score：**

```
[['parcusi', 'parcusing', 'pitching', 'parcusing'], ['unscrew', 'unscrews', 'unscreing', 'consurd'], ['finge', 'functions', 'fu
nctioning', 'functioned'], ['crew', 'crews', 'crewing', 'crew'], ['need', 'needs', 'needing', 'needed'], ['participate', 'part
s', 'participating', 'participated'], ['snicker', 'snickers', 'snickering', 'snideed'], ['bend', 'bends', 'bending', 'bendine
d'], ['tuck', 'tucks', 'tucking', 'tucked'], ['replie', 'replies', 'replieding', 'replied'], ['launch', 'launches', 'launchin
g', 'launched'], ['occeive', 'enscors', 'enscoring', 'occeived'], ['equate', 'auttempts', 'auttempting', 'auttended'], ['reconn
ing', 'refling', 'recoining', 'refling'], ['coiln', 'coilns', 'coiling', 'coiled'], ['poison', 'poisons', 'poisoning', 'pushin
g'], ['interpret', 'intermarries', 'interminning', 'intermarried'], ['remark', 'remarks', 'remarking', 'remarked'], ['finquir
e', 'finquishes', 'finting', 'fintioned'], ['resign', 'retires', 'resigning', 'retrised'], ['went', 'wents', 'wenting', 'wen
t'], ['render', 'renders', 'rending', 'rendered'], ['input', 'inputs', 'introducing', 'introduced'], ['feign', 'feigns', 'feign
ing', 'velyded'], ['alluge', 'alluges', 'alluging', 'alluged'], ['charge', 'charges', 'charging', 'charged'], ['lurk', 'llotter
s', 'lurking', 'lloted'], ['amount', 'amounts', 'amounting', 'amounted'], ['brist', 'brists', 'bristling', 'bristled'], ['lar
e', 'lares', 'tarns', 'tared'], ['arch', 'affords', 'affirming', 'affirmed'], ['grab', 'grabs', 'grabbing', 'grabbed'], ['inclu
de', 'includes', 'including', 'included'], ['abstract', 'abstracts', 'abstracting', 'abstracted'], ['paint', 'paints', 'paintin
g', 'painted'], ['sprawl', 'sprawls', 'sprawling', 'sprawled'], ['sidle', 'sids', 'seizing', 'sid'], ['got', 'gotalles', 'gotal
ling', 'glowed'], ['figge', 'fidgets', 'fidging', 'fidgeted'], ['lurk', 'lurks', 'lurking', 'lurked'], ['resold', 'resolds', 'l
osening', 'tread'], ['consist', 'coasts', 'consisting', 'coasted'], ['enable', 'earns', 'enabling', 'earned'], ['protrude', 'pr
otrudes', 'probeing', 'probend'], ['poison', 'poisons', 'outshining', 'poisoning'], ['vine', 'violates', 'violating', 'violate
d'], ['intend', 'intends', 'indicing', 'kinked'], ['hing', 'hings', 'hinging', 'hinded'], ['suck', 'sents', 'senting', 'sent'],
['suggest', 'suggests', 'sugging', 'suggested'], ['undo', 'undoes', 'undoeing', 'undoed'], ['oblaunc', 'oblaunces', 'explainin
g', 'oblaunced'], ['react', 'detends', 'detending', 'detended'], ['indergain', 'renies', 'inderwing', 'inderwended'], ['glunt',
'glances', 'plaguing', 'glanced'], ['diin', 'disciplines', 'diining', 'disinherited'], ['leab', 'bled', 'leabing', 'bled'], ['f
rolt', 'floreshors', 'lortifying', 'lore'], ['snather', 'snats', 'saluting', 'shatted'], ['bestride', 'bestrides', 'bestirrin
g', 'bestrideed'], ['invoice', 'intends', 'intending', 'invoiced'], ['became', 'becames', 'becaming', 'became'], ['ment', 'sent
s', 'mentioning', 'seemed'], ['giggle', 'giggles', 'giggling', 'gispened'], ['lurk', 'lurks', 'lurking', 'lurked'], ['accuse',
'accuses', 'accusing', 'accused'], ['voin', 'voins', 'voing', 'voinned'], ['mistrap', 'mistracts', 'mistracting', 'mistrappe
d'], ['fesign', 'fesigns', 'fesigning', 'fesigned'], ['hear', 'hears', 'hearing', 'hearned'], ['bethink', 'bethinks', 'bethinki
ng', 'bethinked'], ['enlist', 'enlists', 'entitling', 'enlisted'], ['dispost', 'dispossesses', 'dispatching', 'disposted'], ['i
nspir', 'inspires', 'inspiring', 'inspired'], ['occup', 'sopponsts', 'occupiing', 'occupied'], ['pray', 'fades', 'fading', 'pra
yed'], ['equate', 'equates', 'explaring', 'explared'], ['request', 'steams', 'sentting', 'sent'], ['single', 'singles', 'singli
ng', 'singled'], ['argue', 'argues', 'arguing', 'barged'], ['indicate', 'inderies', 'indicating', 'indicated'], ['blackmare',
'blackmains', 'backfiring', 'blackmained'], ['inserm', 'intermingles', 'intermingling', 'imigated'], ['bribe', 'bribes', 'bribi
ng', 'bribed'], ['spun', 'splans', 'changing', 'spun'], ['remain', 'remains', 'remaining', 'remained'], ['skid', 'skidds', 'ski
dding', 'skidded'], ['inquire', 'inquires', 'inquiring', 'inquired'], ['ode', 'overreacts', 'odvening', 'overreacted'], ['ait',
'attracts', 'aiting', 'attained'], ['feel', 'levots', 'levoting', 'levoted'], ['amount', 'amounts', 'amounting', 'amounted'],
['ondeer', 'precedes', 'preceding', 'opened'], ['look', 'looks', 'looking', 'looked'], ['fear', 'fears', 'fearing', 'feared'],
['squeeze', 'squeezes', 'squeezing', 'squeezed'], ['cret', 'crets', 'creting', 'cret'], ['tare', 'tares', 'tarking', 'tared'],
['charge', 'charges', 'tumbling', 'charged'], ['blinge', 'blids', 'blinging', 'blinged']]
Gaussian score :  0.22
```

- Plot the loss and score curves during training

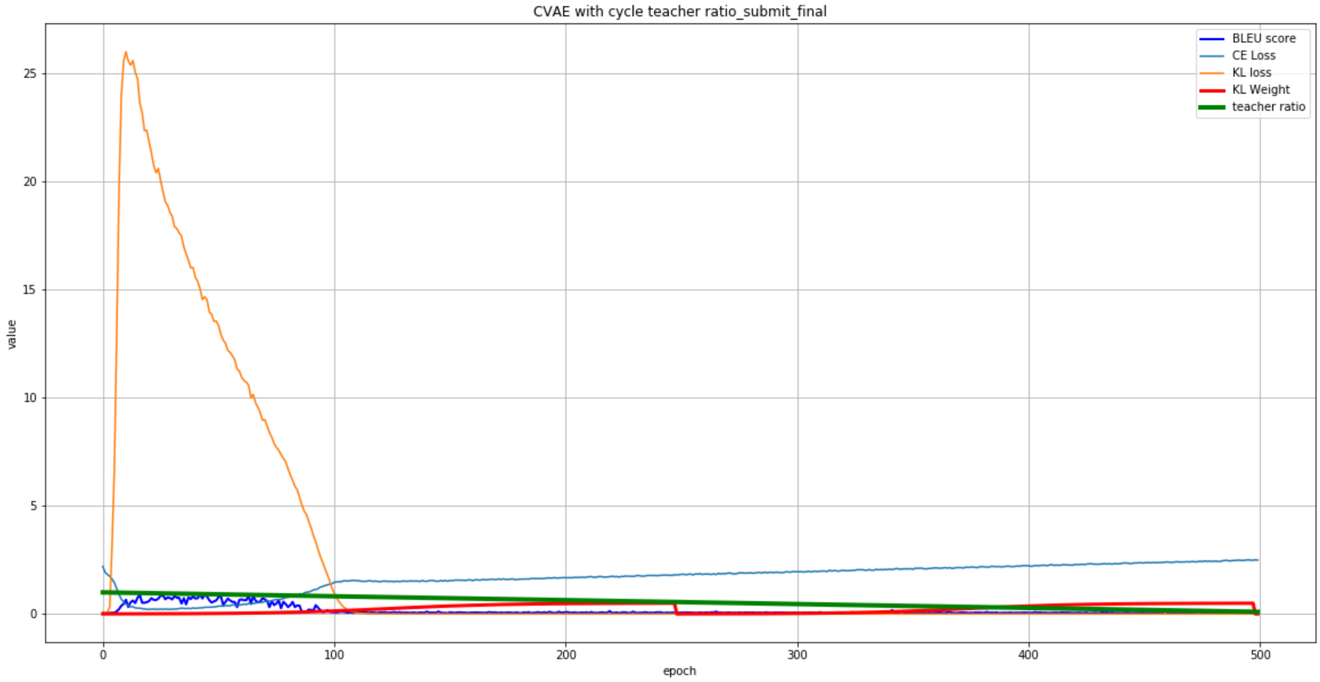


Figure 1

- 可發現隨著 CE loss 降低，BLEU-4 score 逐漸上升，表 reconstruction 成功，直到後面由於 KL weighted 越來越大，導致為了使 KL loss 下降而讓 CE loss 上升，使得 BLEU-4 無法繼續提升。
- teacher forcing ratio 一開始很高，以加快訓練速度，到後面越來越低。
- KL weight：cyclical *(with Sigmoid)*
- Epoch：500, Learning rate：0.05

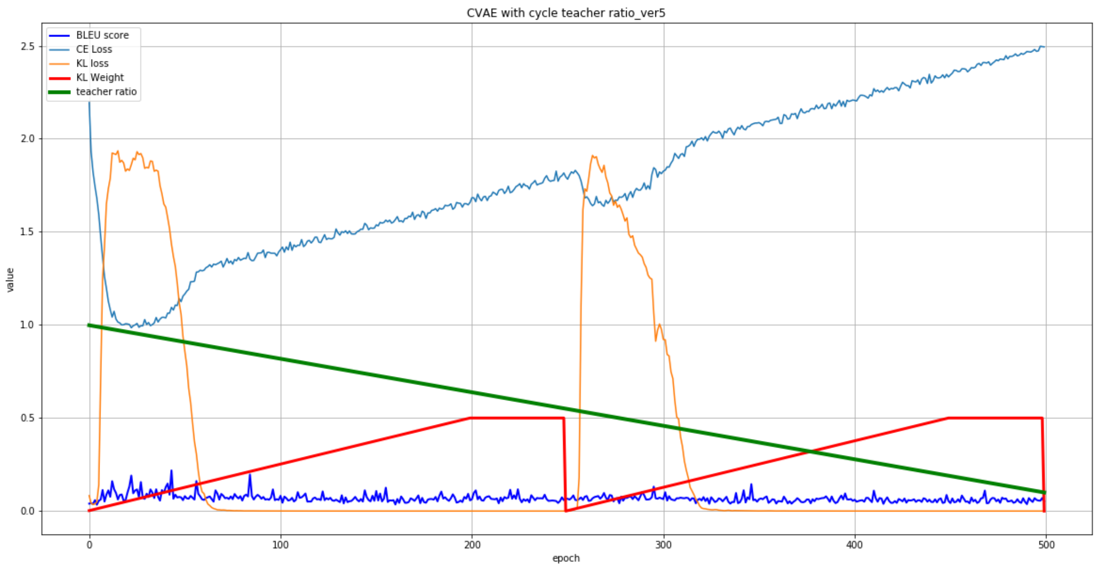


Figure 2

Figure 2 參數都與 Figure 1 相同，唯一不同是 KL weight 在 cyclical 部分是使用和 Figure 相同周期的梯形波改變 weight，可發現在 loss 部分表現很不好。

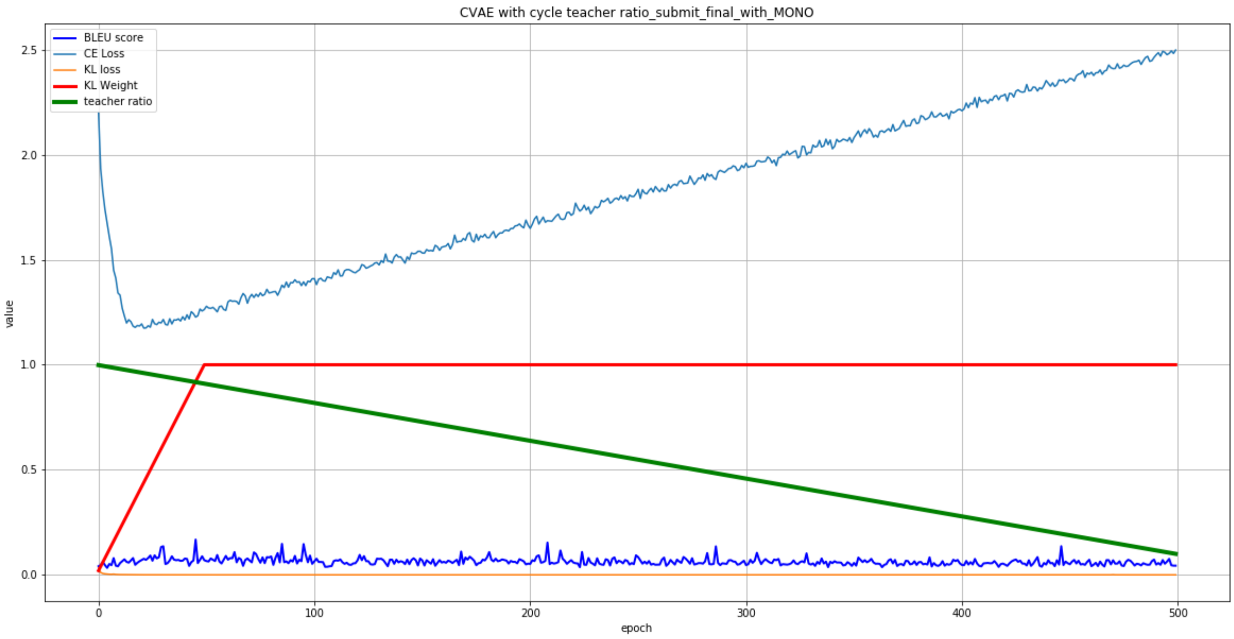


Figure 3

Figure 3 參數都與 Figure 1 相同，唯一不同是 KL weight 是使用了 monotonic 的給法，可以發現在 Cross entropy loss 表現不是很好，同時在 Gaussian 部分表現的也蠻差的。