HW5 Report Conditional Sequence-to-Sequence VAE

tags: `DL and Practice`

電控碩一 黃柏叡 309512074

Report

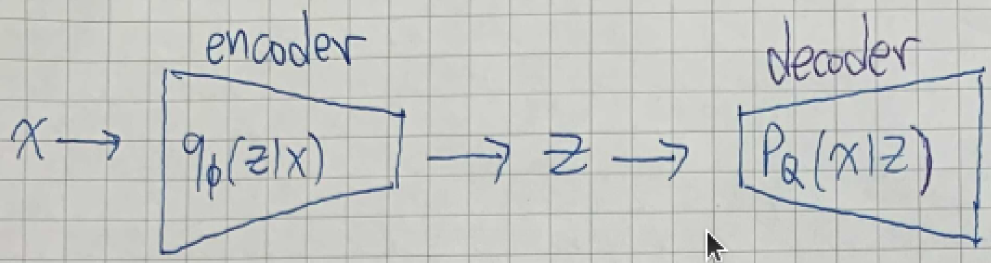
Introduction

Every English words has tenses such as simple present, simple past etc. To convert different tense between input words and target words, we use tense as condition and English words as input and target

Requriemetns:

- Implement a seq2seq CVAE model
- Plot cross-entropy loss and KL loss curves during training
- Plot BLEU-4 score curves during training
- Show the generated words with 4 tense by Gaussian normal distribution

Derivation of CVAE



$$q_\phi(z|x) = N(\mu_\phi(x), \Sigma_\phi(x))$$

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

$$= \frac{1}{2}\{\underbrace{Tr(\Sigma_\phi(x))}_{\text{sum of } \Sigma_\phi(x)} + \mu_\phi(x)^T \mu_\phi(x) - \underbrace{k}_{\text{dim of Gaussian}} - \log\underbrace{|\Sigma_\phi(x)|}_{\text{Diagonal matrix}}\}$$

$$= \frac{1}{2}\sum_k [\Sigma_\phi(x) + \mu_\phi^2(x) - 1 - \log\Sigma_\phi(x)]$$

$$L(\theta,\phi) = -E_{z\sim q_\phi(z|x)}(\log p_\phi(x|z)) + \frac{1}{2}\sum_k(\Sigma_\phi(x) + \mu_\phi^2(x) - 1 - \log\Sigma_\phi(x))$$

$$N(\mu, \sigma e^{6}) = \mu + \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} e^{6}$$

$$\downarrow$$

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

Implementation details

A. Describe how you implement your model

- **Dataloader**
  - To feed character into model, I convert charcter to number by dictionary(SOS, EOS, a,b,c, ..., z applies to 0~27)
  - funciton to convert string to long tensor

```
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

    def string2tensor(self,string,add_eos=True):
        """
        :param add_eox: True or False
        :return: (time1,1) tensor
        """
        indices=[self.char2idx[char] for char in string]
        if add_eos:
            indices.append(self.char2idx['EOS'])
        return torch.tensor(indices,dtype=torch.long).view(-1,1)

    def tense2tensor(self,tense):
        """
        :param tense: 0~3
        :return: (1) tensor
        """
        return torch.tensor([tense],dtype=torch.long)

    def tensor2string(self,tensor):
        """
        :param tensor: (time1,1) tensor
        :return: string (not contain 'EOS')
        """
        re=""
        string_length=tensor.size(0)
        for i in range(string_length):
            char=self.idx2char[tensor[i].item()]
            if char=='EOS':
                break
            re+=char
        return re

    def get_dataset(self,path,is_train):
        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']
                for test_tense in test_tenses:
                    tenses.append([self.tense2idx[tense] for tense in test_tense])
        return words,tenses
```

- Then, I create my own dataset to load training and testing data. In **getitem**, I return one word and its tense condition during training, and return two words(input and target) with its tenses condition during testing

```
class MyDataSet(data.Dataset):
    def __init__(self,path,is_train):
        self.is_train = is_train
        self.dataTransformer=DataTransformer()
        self.words,self.tenses=self.dataTransformer.get_dataset(os.path.join('dataset'
        self.max_length=self.get_max_length(self.words)
        self.string2tensor=self.dataTransformer.string2tensor  # output=(time1,1) tens
        self.tense2tensor=self.dataTransformer.tense2tensor    # output=(1) tensor
        self.tensor2string=self.dataTransformer.tensor2string  # input=(time1,1) tenso
        assert len(self.words)==len(self.tenses),'word list is not compatible with ten

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

    def __getitem__(self, idx):
        if self.is_train:
            return self.string2tensor(self.words[idx],add_eos=True),self.tense2tensor(
        else:
            return self.string2tensor(self.words[idx][0],add_eos=True),self.tense2tens
                   self.string2tensor(self.words[idx][1],add_eos=True),self.tense2tens

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

CVAE is build by 3 part: Encoder + sample part + Decoder

- **Encoder**

hidden state 會先將input embedding成向量，在放入LSTM跑，輸出output, hidden_state, cell_state

```
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)
```

```
        self.rnn = nn.LSTM(hidden_size, hidden_size)

    def forward(self, input, hidden_state, cell_state):
        embedded = self.embedding(input).view(1,1,-1)  # view(1,1,-1) due to input of
        output, (hidden_state, cell_state) = self.rnn(embedded, (hidden_state, cell_st
        return output, hidden_state, cell_state
```

- **sample part**

將encoder輸出的hidden_state透過fully connected layer得到mean跟log variance。根據定義，variance為正值，然而fully connected layer可能會輸出負數，所以使用log variance來解決這個問題

```
    ### middle part forward
    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)
```

- **Reparameterizaton trick**

sample a point from N(mean, exp(logvariance)). In compute graph perspective, it only multiply a constant for mean and log variance. Therefore it can caluculate gradient for parameters of encoder

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

- **Gaussian noise**

generate 100words with 4 tense by gaussian noise

```
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
```

- **Decoder**

輸入的hidden state為中間sample part的輸出

```
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_stat
        output = self.softmax(self.out(output[0]))
        return output, hidden_state, cell_state
```

B. Specify the hyperparameters

- epoch: 1000
- learning rate: 0.01
- teacher forcing ratio
  - The decoder depend on previous output, if the previous output is totally wrong, it will make training very hard. To solve this problem, mandatory input ground truth can help fix it
  - value: from 1.0 to 0.0

```
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
```

- KL weight annealing
  - monotonic, cycle 兩種schedule
  - time for monotonic(number of epoch for kl_weight to reach 1.0)

- time for cycle: number of cycle

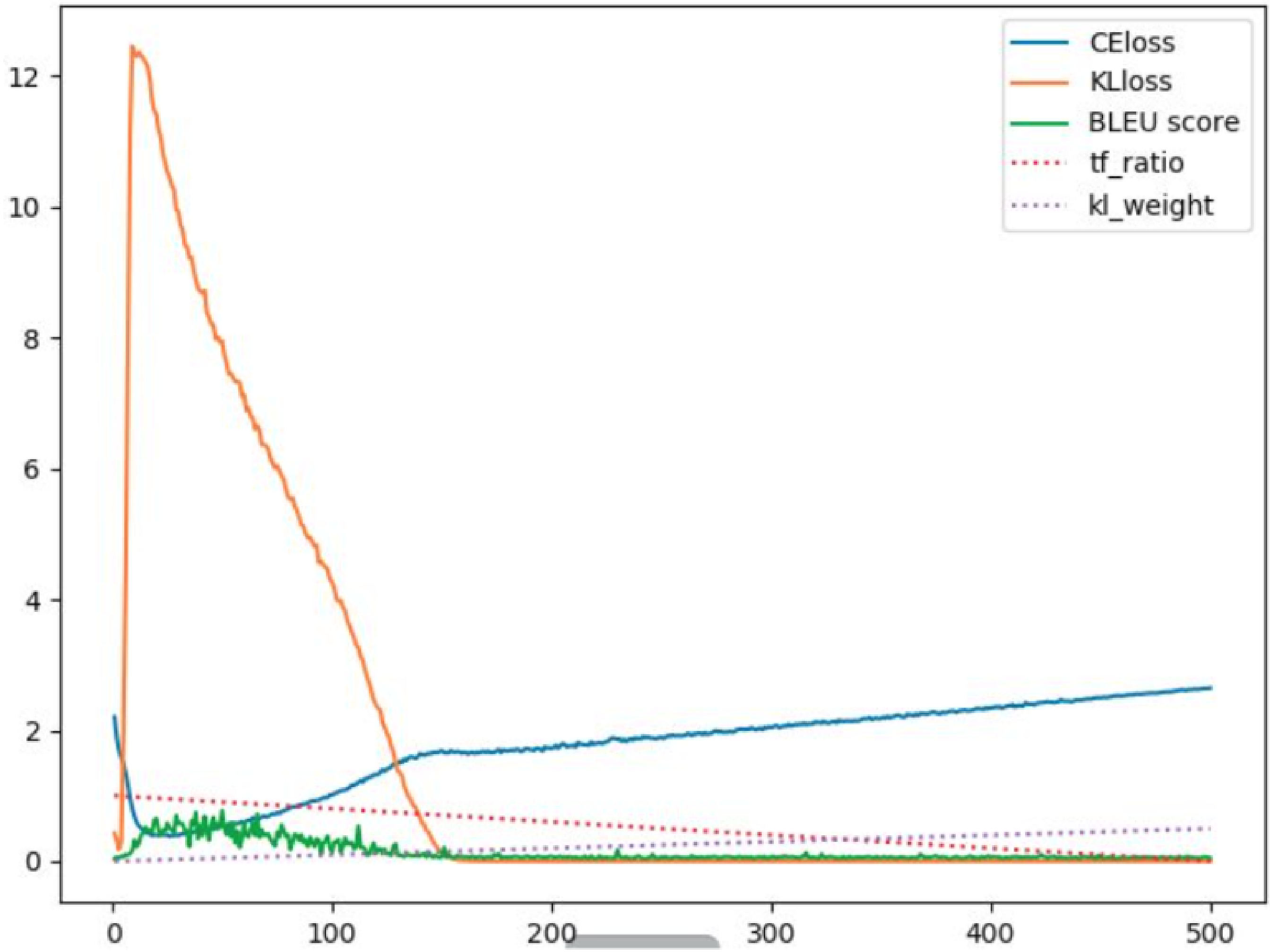
```
def get_kl_weight(epoch,epochs,kl_annealing_type,time):
    assert kl_annealing_type=='monotonic' or kl_annealing_type=='cycle','kl_annealing_
    if kl_annealing_type == 'monotonic':
        return (1./(time-1))*(epoch-1) if epoch<time else 1.

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

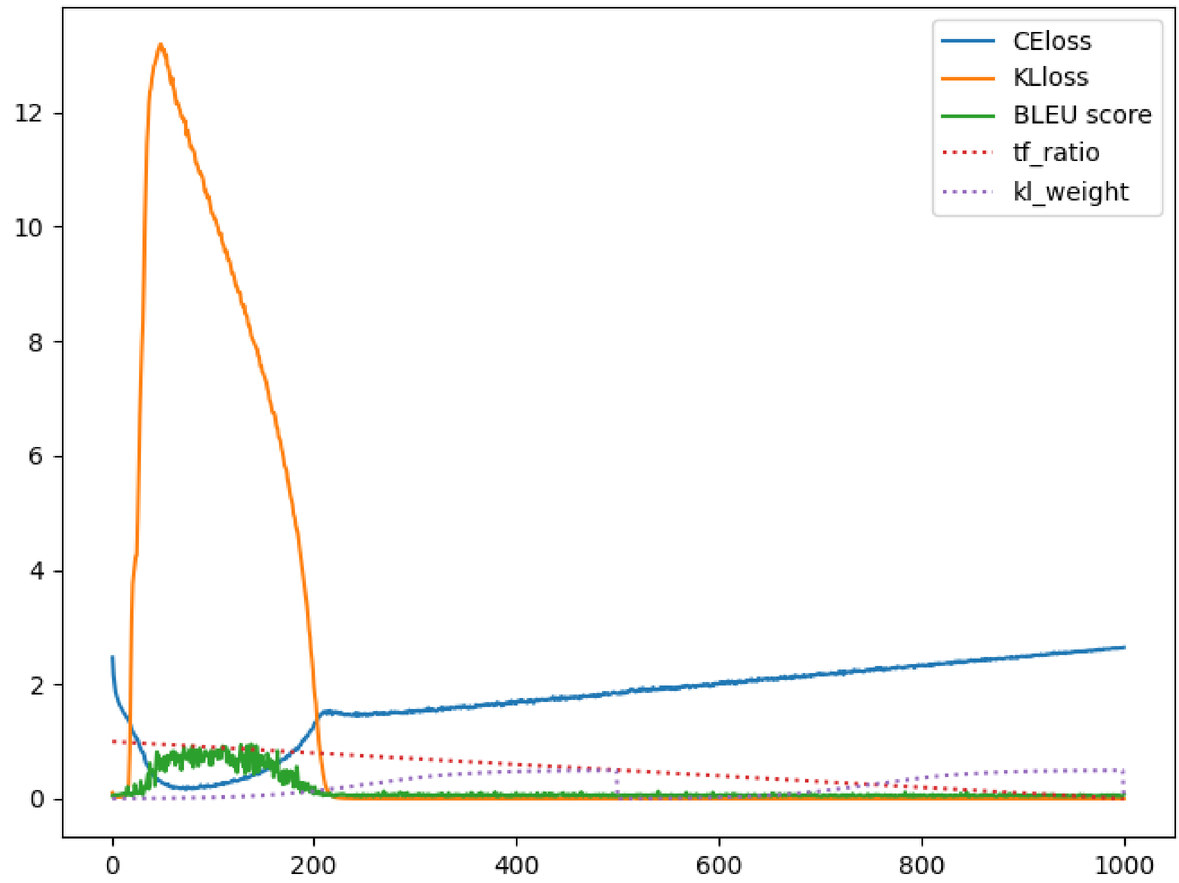
Results and Discussion

A. Show your results of tense conversion and generation and plot the curve during training

monotonic schedule: 500個epoch，teaching_forcing_ratio從1.0線性降至0，KL_weight從0.0線性升至1



cycle schedule: 1000個epoch, teaching_forcing_ratio從1.0線性降至0, 1~500epoch間與501~1000epoch間KL_weight從0.0線性升至0.5



tense conversion -- test.txt prediction:

```
test.txt prediction:
[['abandon', 'abandoned', 'abandoned'], ['abet', 'abetting', 'averting'], ['begin', 'begins', 'begins'], ['expend', 'expends', 'expends'], ['sent', 'sends', 'sents'], ['split', 'splitting', 'splitti
ng'], ['flared', 'flare', 'flare'], ['functioning', 'function', 'function'], ['functioning', 'functioned', 'functioned'], ['healing', 'heals', 'heals']]
```

avg(10)BLEU score: 0.85

Gaussian distribution生成100 words:

```
generate 100 words with 4 different tenses:
[['predict', 'misrepresents', 'predicting', 'presisted'], ['twist', 'gists', 'stivering', 'paved'], ['signal', 'sirlences', 'signalling', 'sirled'], ['confide', 'chid', 'confiding', 'confided'], ['d
iscleave', 'discleaves', 'discleaving', 'disclosed'], ['proceed', 'proceeds', 'proceeding', 'proceeded'], ['bough', 'boughs', 'bought', 'bought'], ['creep', 'trembles', 'trembling', 'crept'], ['knit
', 'knits', 'knitting', 'knitted'], ['esteem', 'tears', 'esteeming', 'peaced'], ['resemble', 'resembles', 'reseating', 'resembled'], ['disguise', 'disguises', 'disguising', 'disguised'], ['hear', 'h
ears', 'hearing', 'heard'], ['belch', 'belches', 'belching', 'belched'], ['prohe', 'prohes', 'prohesting', 'prohed'], ['pract', 'practles', 'practing', 'practed'], ['mention', 'mentions', 'mentionin
g', 'met'], ['await', 'awords', 'awaiting', 'awaited'], ['gaze', 'gosses', 'gazing', 'gazed'], ['tell', 'tells', 'telling', 'theld'], ['intermarry', 'adjerts', 'intermarriing', 'intermarried'], ['mu
ltiply', 'multiplies', 'multiplying', 'multiplied'], ['practle', 'travels', 'prattling', 'practled'], ['challenge', 'chortles', 'challowing', 'challenged'], ['inspire', 'inspires', 'inspiring', 'ins
pired'], ['manifest', 'manifests', 'manifesting', 'manifested'], ['entitle', 'entits', 'entituting', 'entitled'], ['parody', 'masts', 'parodiing', 'parode'], ['gather', 'hears', 'gathering', 'healed
'], ['unscrew', 'unscrews', 'unscrewing', 'unscrewed'], ['drift', 'drifts', 'drifting', 'drifted'], ['snear', 'denys', 'snearing', 'snearched'], ['vignal', 'vignals', 'visenging', 'vignaled'], ['exp
ress', 'expresses', 'expressing', 'expressed'], ['penie', 'testifies', 'peniting', 'testified'], ['erign', 'erigns', 'ericting', 'erigned'], ['buid', 'buidsts', 'budding', 'buided'], ['delude', 'del
udes', 'deluding', 'deluded'], ['brun', 'brungs', 'brunging', 'constrused'], ['restore', 'inserts', 'inserting', 'inserted'], ['trod', 'trod', 'troding', 'trod'], ['stole', 'stoles', 'stollegraphing
', 'stole'], ['yrime', 'firms', 'firming', 'firmed'], ['blank', 'blinks', 'blanking', 'blanked'], ['run', 'runs', 'graduating', 'graduated'], ['think', 'thinks', 'thinking', 'thickened'], ['decide',
 'deals', 'pretending', 'pretaced'], ['dislike', 'provides', 'disliking', 'disliked'], ['scrutinize', 'scrutinizes', 'scrutinizing', 'scrutinized'], ['steal', 'stightens', 'stealing', 'stigled'], ['
pace', 'paces', 'paceding', 'paced'], ['avell', 'avells', 'avelling', 'avowed'], ['expiate', 'expiates', 'expanding', 'expiated'], ['abser', 'accipates', 'abserging', 'accipped'], ['interview', 'int
erviews', 'interving', 'interviewed'], ['retort', 'retorts', 'retorting', 'retorted'], ['lease', 'regleases', 'leading', 'leaded'], ['alter', 'streaks', 'altering', 'streaked'], ['nive', 'invoices',
 'invoicing', 'nvided'], ['unlock', 'unlocks', 'unlowing', 'unlowed'], ['velcome', 'velcomes', 'regouncing', 'regounced'], ['disquiet', 'resounds', 'resounding', 'disquieted'], ['precipitate', 'prec
ipitates', 'preceding', 'precipitated'], ['purge', 'purges', 'purging', 'purged'], ['seize', 'seizes', 'seizing', 'seized'], ['wax', 'harasses', 'harassing', 'waxed'], ['reqiire', 'reqists', 'reqiir
ing', 'remained'], ['paint', 'eats', 'eating', 'painted'], ['retouch', 'earns', 'retouching', 'retouched'], ['decrease', 'decries', 'destroying', 'decreased'], ['awoke', 'awokes', 'awoking', 'awoke'
], ['obsell', 'obsells', 'obselling', 'obselled'], ['pack', 'packs', 'applying', 'palked'], ['skid', 'skids', 'skidding', 'skidded'], ['hire', 'hires', 'hiring', 'hired'], ['yank', 'yearns', 'yearin
g', 'yeard'], ['melt', 'melts', 'melting', 'melted'], ['appose', 'apposes', 'apposing', 'apposed'], ['envolve', 'envolves', 'envolving', 'encomes'], ['undid', 'undids', 'undiging', 'hissed'], ['go',
 'goins', 'going', 'going'], ['bent', 'bents', 'bentwing', 'bent'], ['bear', 'bears', 'bearning', 'bearded'], ['manifest', 'manufactures', 'manifesting', 'manifested'], ['elicit', 'elicits', 'elicit
ing', 'elicited'], ['eacquire', 'eats', 'eating', 'abetted'], ['adjust', 'adjusts', 'assuming', 'assumed'], ['acquire', 'acquires', 'acquiring', 'acquired'], ['relieve', 'rollers', 'relieving', 'rel
ieved'], ['pace', 'approves', 'pacing', 'pacipated'], ['precipitate', 'progists', 'preceding', 'precipitated'], ['prevail', 'prevails', 'prevailing', 'prevailed'], ['remonstrate', 'remonstrates', 'r
emonstrating', 'remonstrated'], ['ntotter', 'ntotters', 'ntotting', 'ntotted'], ['slug', 'slugs', 'slugging', 'slugged'], ['ask', 'adopts', 'adopting', 'adopted'], ['overhai', 'troubles', 'owerging'
, 'troubled'], ['elact', 'elaxes', 'elaxing', 'elacted'], ['avent', 'avents', 'astening', 'acqeed'], ['chiry', 'chires', 'chiring', 'chired']]
```

avg(10)Gaussian score: 0.3

B. Discuss the results according to teaching forcing ratio, KL weight and learning rate

一開始CE LOSS大於KL LOSS，model表現很差，BLEU的分數也很低 大約從第10個epoch開始，CE LOSS逐漸下降，代表英文字的reconstruction效果越來越好，BLEU分數提高，但此時latent vector與Gaussion normal distribution越來越不像，因此KL LOSS上升。 大約到第50個epoch，由於KL weight

變大，KL_weight * KL LOSS開始主導loss funciton，因此KL LOSS會變小，CE LOSS連帶上升導致BLEU分數下降 當epoch更大時，由於KL_weight持續提高，使KL LOSS持續降低，CE LOSS持續升高，導致BLEU分數很低