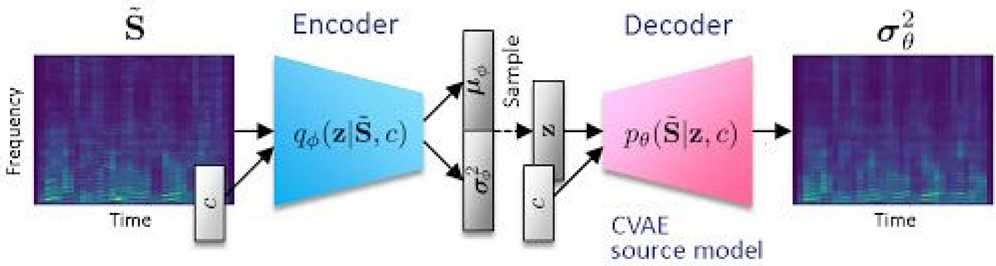
lab5

0856105 資工碩一 吳承翰

1.Introduction

利用CVAE處理英文word的時態轉換問題，例如： abandon(sp) -> abandoned(p)
condition有：simple present(sp), third person(tp), present progressive(pg), simple past(p) 4種，在進入Encoder及Decoder時與data concate。



2.Derivation of CVAE

3.Implement details

CVAE是由三個部份所組成：Encoder+中間sample part+Decoder。

Encoder:
hidden_state加入會先把alphabet embedding成一個向量，再丟進LSTM去跑，最後輸出output,hidden_state,cell_state

```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sample part:
我們把encoder輸出的hidden_state透過fully connected layer變為32-dim的mean與log variance，知所以用log variance是因為variance皆為正值，但fully connected layer 可能會輸出負數。
有了mean與log variance後，我們就可以透過reparameterization trick sampling一個32-dim的latent，32-dim latent與8-dim condition concate後，在透過一個fully connected layer轉換為hidden_state的維度。

```
"""
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:
輸入的hidden_state為剛剛”中間sample part”的輸出，cell_state則初始化為0 tensor。

```
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N(mean,exp(log variane))中sample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
```

Text generation by Gaussian noise:
用torch.randn()隨機生成一個32-dim的latent tensor，再把這latent tensor與tense tensor concate，並作為decoder的hidden_state

```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
```

dataloader:
把SOS,EOS,a,b,c,....,z分別對應到0~27，以利將來torch.nn.embedding()

```
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
```

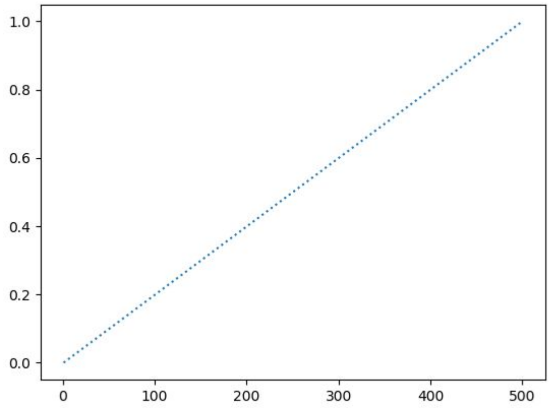
```
def get_dataset(self,path,is_train):
    """
    :returns:
    if(train):  words=[w1,w2,w3,.....,wn], tenses:[0,1,2,3,0,1,2,3...]
    if(test):  words=[[w1,w2],[w3,w4]....[wn-1,wn]], tense:[[sp,p],[sp,pg]...,[pg,tp]]
    """
    words=[]
    tenses=[]
    with open(path,'r') as file:
        if is_train: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

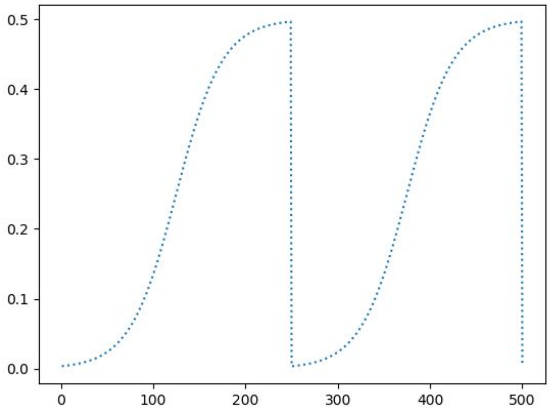
```
def __len__(self):
    return len(self.words)

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])
```

epoch設500，learning rate設0.05

KL weight:
有monotonic,cycle兩種schedule



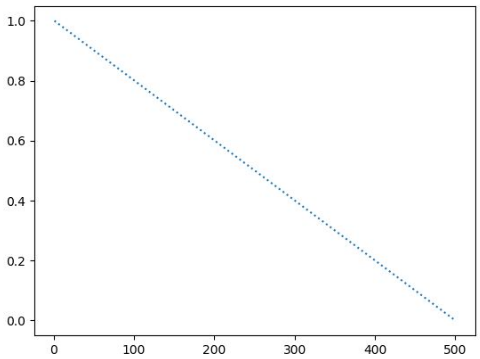



```python
def get_kl_weight(epoch,epochs,kl_annealing_type,time):
    """
    :param epoch: i-th epoch
    :param kl_annealing_type: 'monotonic' or 'cycle'
    :param time:
        if('monotonic'): # of epoch for kl_weight from 0.0 to reach 1.0
        if('cycle'):     # of cycle
    """
    assert kl_annealing_type=='monotonic' or kl_annealing_type=='cycle','kl_annealing_type not exist!'

    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))
        # KL_weight = (iter-10000) * 0.001
        KL_weight = min(1, max(0, KL_weight))
        return KL_weight
```

teacher forcing ratio:

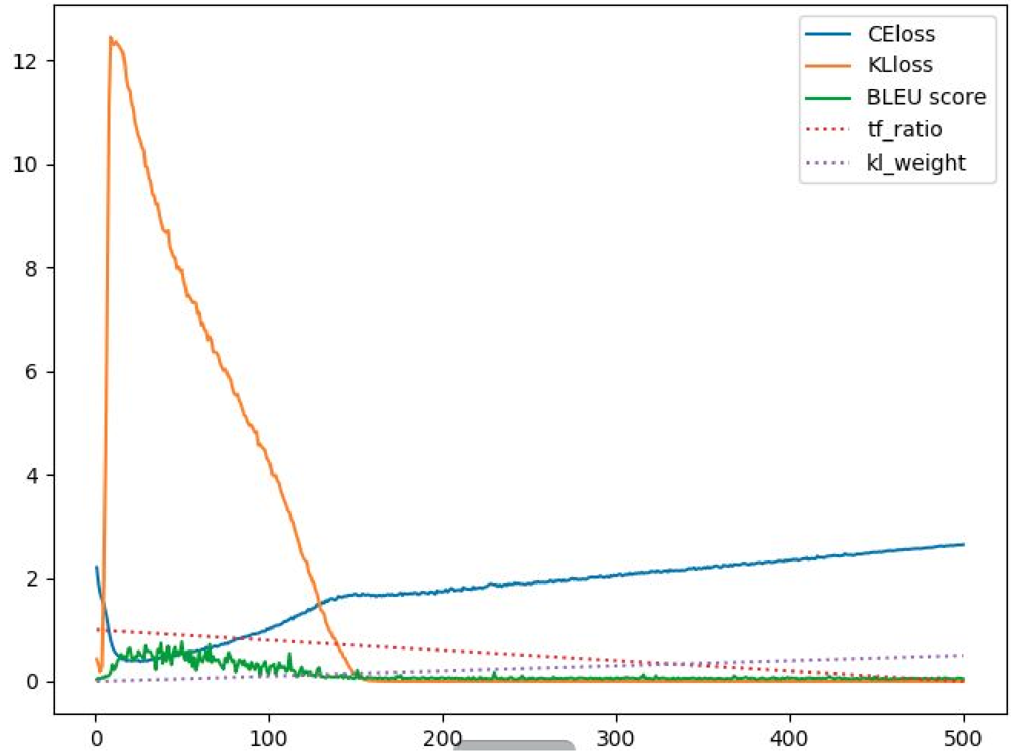



```python
def get_teacher_forcing_ratio(epoch,epochs):
    # from 1.0 to 0.0
    teacher_forcing_ratio = 1.-(1./(epochs-1))*(epoch-1)
    return teacher_forcing_ratio
```

4.Results and discussion

monotonic shedule:

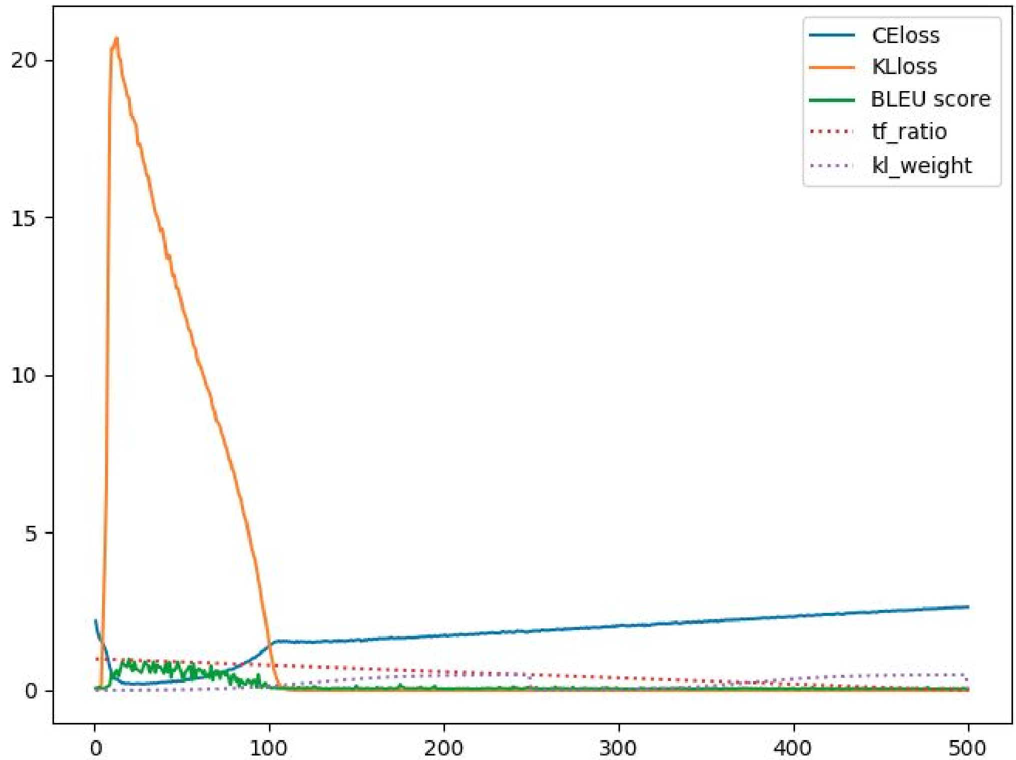
500個epoch， teacher_forcing_ratio從1.0線性降至0.0， kl_weight從0.0線性升至1.0



cyclical schedule:

500個epoch， teacher_forcing_ratio從1.0線性降至0.0，

第1到250個epoch間與第251到500個epoch間kl_weight從0.0線性升至0.5



一開始CE loss大於KL loss，model還沒學到任何東西，BLEU分數也很小。

在大約第6個epoch，時CE loss逐漸下降，代表word的reconstuction成功了，因此BLEU提昇;
但latent distribution與prior:N(0,1)長得越來越不像，因此KL loss上升。

大約到第20個epoch時，由於kl_weight變大，kl_wight*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下降。

在更後面的epoch，由於kl_weigth 一直提高，迫使KL loss一直都很低，BLEU也連帶無法升
高。

我發現KL loss升高時，BLEU也會升高。
在cycle shedule中，第250個epoch開始KL looss與BLEU都應該要上升才對，可能是我KL
weight的cycle頻率設的不對。

test.txt prediction:

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

```
['expend', 'expends', 'remonstrates'], ['sent', 'sends', 'senses'], ['split', 'splitting', 'splitting'],
```

```
['flared', 'flare', 'flare'], ['functioning', 'function', 'function'], ['functioning', 'functioned', 'functioned'], ['healing', 'heals', 'heals']]
```

可以看到epend與sent這兩個word的prediction錯誤
BLEU socre：0.82

Gaussian distribution生成100 word:

```
['pity', 'lingers', 'pitying', 'led'], ['slump', 'slams', 'slamming', 'slumped'], ['spring', 'springs', 'springing', 'spring'],
```

```
['pay', 'pays', 'pacing', 'paced'], ['mumble', 'mumbles', 'mumbling', 'mumbled'], ['insist', 'insists', 'imitating', 'insisted'],
```

```
['withhold', 'withholds', 'withholding', 'withdrew'], ['disrupt', 'thrashes', 'thrashing', 'thrashed'], ['wade', 'wades', 'wading', 'waded'],
```

```
['dive', 'dives', 'diving', 'dived'], ['jingle', 'jingles', 'jerking', 'bit'], ['snap', 'snaps', 'snapping', 'snapped'],
```

Gaussian score：0.42