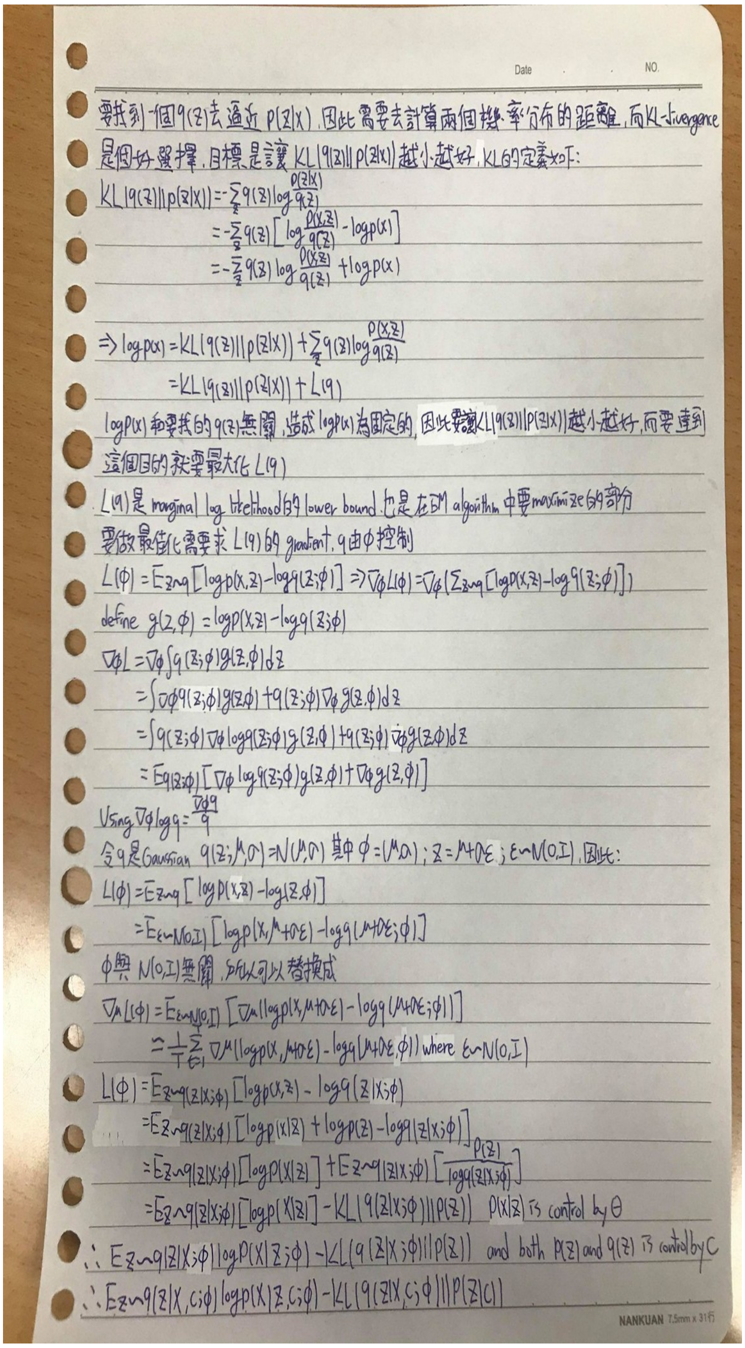
Lab 5 Report
309551064 張凱翔

- Introduction

  Implement a conditional seq2seq VAE for English tense conversion and generation and regard tense as condition. In training phase, input and output are the same, however, in the evaluating phase the output is conditional on the output tense to generate the tense conversion word. And there are some constraints and requirements for this lab:
  - The output of reparameterization trick should be log variance
  - Text generation should produced by Gaussian noise
  - Implement two method of kl cost annealing and compare
  - Show the training process including bleu score, kl loss and cross entropy loss
- Derivation of CVAE



Date . . NO.

要找到一個$q(z)$去逼近$p(z|x)$，因此需要去計算兩個機率分布的距離，而KL-divergence是個好選擇，目標是讓$KL(q(z)||p(z|x))$越小越好，KL的定義如下：

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

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

$\log p(x)$和要找的$q(z)$無關，造成$\log p(x)$為固定的，因此要讓$KL(q(z)||p(z|x))$越小越好，而要達到這個目的就要最大化$L(q)$

$L(q)$是marginal log likelihood的lower bound，也是在EM algorithm中要maximize的部分

要做最佳化需要求$L(q)$的gradient，q由ϕ控制

$$L(\phi) = E_{z\sim q}[\log p(x,z) - \log q(z;\phi)] \Rightarrow \nabla_\phi L(\phi) = \nabla_\phi(\Sigma_{z\sim q}[\log p(x,z) - \log q(z;\phi)])$$

define $g(z,\phi) = \log p(x,z) - \log q(z;\phi)$

$$\nabla_\phi L = \nabla_\phi \int q(z;\phi) g(z,\phi)\,dz$$
$$= \int \nabla_\phi q(z;\phi) g(z,\phi) + q(z;\phi)\nabla_\phi g(z,\phi)\,dz$$
$$= \int q(z;\phi)\nabla_\phi \log q(z;\phi) g(z,\phi) + q(z;\phi)\nabla_\phi g(z,\phi)\,dz$$
$$= E_{q(z;\phi)}[\nabla_\phi \log q(z;\phi) g(z,\phi) + \nabla_\phi g(z,\phi)]$$

Using $\nabla_\phi \log q = \frac{\nabla_\phi q}{q}$

令q是Gaussian $q(z;\mu,\sigma) = N(\mu,\sigma)$ 其中 $\phi = (\mu,\sigma)$；$z = \mu + \sigma\varepsilon$；$\varepsilon \sim N(0,I)$，因此：

$$L(\phi) = E_{z\sim q}[\log p(x,z) - \log(z,\phi)]$$
$$= E_{\varepsilon\sim N(0,I)}[\log p(x,\mu+\sigma\varepsilon) - \log q(\mu+\sigma\varepsilon;\phi)]$$

ϕ與$N(0,I)$無關，所以可以替換成

$$\nabla_\mu L(\phi) = E_{\varepsilon\sim N(0,I)}[\nabla_\mu(\log p(x,\mu+\sigma\varepsilon) - \log q(\mu+\sigma\varepsilon;\phi))]$$
$$\simeq \frac{1}{T}\sum_{t=1}^{T}\nabla_\mu(\log p(x,\mu+\sigma\varepsilon) - \log q(\mu+\sigma\varepsilon,\phi)) \text{ where } \varepsilon\sim N(0,I)$$

$$L(\phi) = E_{z\sim q(z|x;\phi)}[\log p(x,z) - \log q(z|x;\phi)$$
$$= E_{z\sim q(z|x;\phi)}[\log p(x|z) + \log p(z) - \log q(z|x;\phi)]$$
$$= E_{z\sim q(z|x;\phi)}[\log p(x|z)] + E_{z\sim q(z|x;\phi)}\left[\frac{p(z)}{\log q(z|x;\phi)}\right]$$
$$= E_{z\sim q(z|x;\phi)}[\log p(x|z)] - KL(q(z|x;\phi)||p(z))$$ $p(x|z)$ is control by θ

$\therefore E_{z\sim q(z|x;\phi)}\log p(x|z;\phi) - KL(q(z|x;\phi)||p(z))$ and both $p(z)$ and $q(z)$ is control by c

$\therefore E_{z\sim q(z|x,c;\phi)}\log p(x|z,c;\phi) - KL(q(z|x,c;\phi)||p(z|c))$

NANKUAN 7.5mm x 31行

- Implementation details

- Describe how you implement your model

1. **Train dataset**

First, read the training file and use it as input to the train dataset.

```
train_pairs = pd.read_csv('./lab5_dataset/train.txt', header=None).values
train_dataset = TrainTenseDataset(train_pairs)
```

The dataset splits the line into four words of four different tenses and then transforms them into number (2~27) and add EOS_TOKEN (1) at the end of words. Final, append all the numbers transformed from words and their tenses (0~3) together then it is possible to get them by index.

```
class TrainTenseDataset(Dataset):
    def __init__(self, data):
        self.data = []
        self.label = []
        for i in range(len(data)):
            sp, tp, pg, p = data[i][0].split(' ')
            sp = Word2Number(sp)
            tp = Word2Number(tp)
            pg = Word2Number(pg)
            p = Word2Number(p)
            data_pairs = [sp, tp, pg, p]
            for j in range(len(data_pairs)):
                self.data.append(data_pairs[j])
                self.label.append(j)

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

    def __getitem__(self, index):
        return self.data[index], self.label[index]
```

```
def Word2Number(word):
    number = []
    for i in range(len(word)):
        number.append(ord(word[i]) - 97 + 2)
    number.append(EOS_TOKEN)
    return np.array(number)
```

2. **Test dataset**

First, read the testing file and use it as input to the test dataset.

```
test_pairs = pd.read_csv('./lab5_dataset/test.txt', header=None).values
test_dataset = TestTenseDataset(test_pairs)
```

The dataset first specifies two words' tenses as a list in testing data. Then, split the line into two words of input words and target words and transforms them into number (2~27) and add EOS_TOKEN (1) at the end of words. Final, append all input words and target words and also their tenses together then it is possible to get them by index. (an index of data consists of two words and two tenses)

```
class TestTenseDataset(Dataset):
    def __init__(self, data):
        self.data = []
        self.label = []
        tense = [(0, 3), (0, 2), (0, 1), (0, 1), (3, 1), (0, 2), (3, 0), (2, 0), (2, 3), (2, 1)]
        for i in range(len(data)):
            data1, data2 = data[i][0].split(' ')
            self.data.append([Word2Number(data1), Word2Number(data2)])
            self.label.append(tense[i])

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

    def __getitem__(self, index):
        return self.data[index], self.label[index]
```

3. CVAE

The CVAE can be divided into five parts, **initialization of encoder state**, **encoder**, **latent** (middle), **initialization of decoder state** and **decoder** and I will explain each part below.

```python
def __init__(self, vocab_size, hidden_size, latent_size, condition_embedding_size):
    super(VAE, self).__init__()

    self.vocab_size = vocab_size
    self.hidden_size = hidden_size
    self.latent_size = latent_size
    self.condition_embedding_size = condition_embedding_size

    self.tense_embedding = nn.Embedding(4, condition_embedding_size)

    self.encoder = self.Encoder(vocab_size, hidden_size)

    self.hidden2mean = nn.Linear(hidden_size, latent_size)
    self.hidden2variance = nn.Linear(hidden_size, latent_size)

    self.latent2hidden = nn.Linear(latent_size + condition_embedding_size, hidden_size)

    self.decoder = self.Decoder(hidden_size, vocab_size)
```

```python
def forward(self, word, tense, use_teacher_forcing):
    # encoder initial state
    tense = self.tense_embedding(tense).unsqueeze(1) #add one dimension
    encoder_initial_hidden_state = self.encoder.init_hidden_state(self.hidden_size - self.condition_embedding_size)
    encoder_initial_hidden_state = torch.cat([encoder_initial_hidden_state, tense], dim=-1)
    encoder_initial_cell_state = self.encoder.init_cell_state()

    # encoder
    _, hidden_state, cell_state = self.encoder(word, encoder_initial_hidden_state, encoder_initial_cell_state)

    # middle
    mean = self.hidden2mean(hidden_state)
    variance = self.hidden2variance(hidden_state)
    latent = self.reparameterize(mean, variance)

    # decoder initial state
    decoder_initial_hidden_state = torch.cat([latent, tense], dim=-1)
    decoder_initial_hidden_state = self.latent2hidden(decoder_initial_hidden_state)
    decoder_initial_cell_state = self.decoder.init_cell_state()

    decoder_input = torch.tensor([[SOS_TOKEN]], device=device)
    pred_distribution = torch.zeros(word.size(1), self.vocab_size, device=device)

    # decoder
    decoder_hidden_state = decoder_initial_hidden_state
    decoder_cell_state = decoder_initial_cell_state
    pred_output = []
    for i in range(word.size(1)):
        output, decoder_hidden_state, decoder_cell_state = self.decoder(decoder_input, decoder_hidden_state, decode
        pred_distribution[i] = output[0]

        if use_teacher_forcing:
            decoder_input = torch.tensor([[word[0][i]]], device=device)
        else:
            if torch.argmax(output).cpu().detach().numpy() == EOS_TOKEN:
                break
            decoder_input = torch.argmax(output).unsqueeze(0).unsqueeze(0)
        pred_output.append(torch.argmax(output).cpu().detach().numpy().item())

    return pred_output, pred_distribution, mean, variance
```

1) initialization of encoder state

First, convert tense into embedding vector and add one dimension to satisfy the requirement of LSTM. Then, initial a torch with all 0 with size (1, 1, hidden_size - conditional_embedding_size) and concatenate with embedding vector to form LSTM initial hidden state. Final, initial a torch with all 0 with size (1, 1, hidden_size) as LSTM initial cell state.

2) encoder

Take word, initial hidden state and initial cell state as input to encode the word. First, convert each number of alphabet into embedding vector and permute the vector to satisfy the requirement of input of LSTM. Then, use embedding vector, initial hidden state and initial cell state as input to get the final output, hidden_state and cell_state and return.

```python
class Encoder(nn.Module):
    def __init__(self, vocab_size, hidden_size):
        super(VAE.Encoder, self).__init__()

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

    def forward(self, x, initial_hidden_state, initial_cell_state):
        word_embedding = self.embedding(x)
        word_embedding = word_embedding.permute(1, 0, 2)
        output, (hidden_state, cell_state) = self.lstm(word_embedding, (initial_hidden_state, initial_cell_state))
        return output, hidden_state, cell_state

    def init_hidden_state(self, size):
        return torch.zeros(1, 1, size, device=device)

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

3) latent

After get the hidden state, I use two linear layers to transform hidden state to mean and log variance and use reparameterize to get latent. In function reparameterize, it samples from the N(0, 1) and calculates together with mean and log variance to get latent.

```
def reparameterize(self, mu, logvar):
    std = torch.exp(0.5 * logvar)
    eps = torch.normal(torch.FloatTensor([0] * self.latent_size),
                       torch.FloatTensor([1] * self.latent_size)).to(device)
    return mu + eps * std
```

4) initialization of decoder state

First, concatenate latent with embedding vector of tense and use a linear layer to downsize the dimension to hidden size as the initial hidden state of LSTM of decoder. Then, initial a torch with all 0 with size (1, 1, hidden_size) as initial cell state of LSTM of decoder. Final, claim the first input as SOS_TOKEN (0).

5) decoder

Unlike the encoder use all number of alphabet as input all at once, decoder use one number of alphabet at a time. Take word, initial hidden state and initial cell state as input to get the original input.

First, convert number of alphabet into embedding vector and go through a ReLU function. Then, use embedding vector, initial hidden state and initial cell state as input to get the final output and downsize it to the probability of each token.

```
class Decoder(nn.Module):
    def __init__(self, hidden_size, vocab_size):
        super(VAE.Decoder, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(vocab_size, hidden_size)
        self.lstm = nn.LSTM(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, vocab_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, x, hidden_state, cell_state):
        output = self.embedding(x)
        output = F.relu(output)
        output, (hidden_state, cell_state) = self.lstm(output, (hidden_state, cell_state))
        output = self.softmax(self.out(output[0]))

        return output, hidden_state, cell_state

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

After get the probability of each token, I will check whether using teacher forcing. If so, use the correct answer as the next input, if not, use the predict one as the next input. Final, return predict output, mean and variance to calculate loss.

4. Loss function

Loss function consists of two parts, cross entropy loss and KL loss. After calculate both losses, the final loss is calculated as shown in figure.

```
ce_loss, kl_loss = loss_function(distribution, word, mean, variance, len(output))
loss = ce_loss + kl_weight * kl_loss
```

The way calculating cross entropy loss is to use the probability of each token and the correct answer where the length of both is the output word length.

The way calculating kl loss is to use the mean and variance of the output of model. I find a formula at the appendix of this paper and use it to derive kl loss. Most important thing is that the spec specifies that the variance is log variance, so it should use exponential to get the origin variance.

$$-D_{KL}((q_\phi(\mathbf{z})||p_\theta(\mathbf{z})) = \int q_\theta(\mathbf{z}) \left(\log p_\theta(\mathbf{z}) - \log q_\theta(\mathbf{z})\right) \, d\mathbf{z}$$
$$= \frac{1}{2} \sum_{j=1}^{J} \left(1 + \log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2\right)$$

```
def loss_function(distribution, word, mean, variance, pred_len):
    criterion = nn.CrossEntropyLoss().to(device)
    ce_loss = criterion(distribution[:pred_len], word[0][:pred_len])

    kl_loss = -0.5 * torch.sum(1 + variance - mean.pow(2) - variance.exp())
    return ce_loss, kl_loss
```

5. Train

In each epoch, I will update teacher forcing ratio and kl weight. Using each word as input word to get the output. Using output to calculate the loss and update the parameters of model. Using predict words to see the model's performance. I will keep whatever I need for the report in training phase.

```
for epoch in tqdm(range(EPOCHS)):
    model.train()

    teacher_forcing_ratio = _teacher_forcing_ratio(epoch, EPOCHS)
    kl_weight = _kl_weight(epoch, KL_METHOD, KL_PERIOD)

    total_ce_loss = 0
    total_kl_loss = 0
    train_total_bleu_4 = 0

    for word, tense in train_loader:
        optimizer.zero_grad()

        word = word.to(device).long()
        tense = tense.to(device).long()

        use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False

        output, distribution, mean, variance = model(word, tense, use_teacher_forcing)

        ce_loss, kl_loss = loss_function(distribution, word, mean, variance, len(output))
        loss = ce_loss + kl_weight * kl_loss

        target_word = Number2Word(word[0][:-1].cpu().detach().numpy())
        pred_word = Number2Word(output)

        bleu_4 = compute_bleu(pred_word, target_word)

        loss.backward()
        optimizer.step()

        total_ce_loss += ce_loss
        total_kl_loss += kl_loss
        train_total_bleu_4 += bleu_4
```

6. **Evaluate**

This function is similar with the forward function except the initialization of decoder state and decoder parts. Instead of using input tense as initial hidden state, the function use the output tense to achieve tense conversion. And in the generate part, the length is not constrained and it will continuously generate until meeting EOS_TOKEN and the next input is always predicted token.

```
def evaluate(self, input_word, input_tense, output_tense):
    # encoder initial state
    input_tense = self.tense_embedding(input_tense).unsqueeze(1) #add one dimension
    output_tense = self.tense_embedding(output_tense).unsqueeze(1)
    encoder_initial_hidden_state = self.encoder.init_hidden_state(self.hidden_size - self.condition_embedding_size)
    encoder_initial_hidden_state = torch.cat([encoder_initial_hidden_state, input_tense], dim=-1)
    encoder_initial_cell_state = self.encoder.init_cell_state()

    #encoder
    _, hidden_state, cell_state = self.encoder(input_word, encoder_initial_hidden_state, encoder_initial_cell_state

    # middle
    mean = self.hidden2mean(hidden_state)
    variance = self.hidden2variance(hidden_state)
    latent = self.reparameterize(mean, variance)

    # decoder initial state
    decoder_initial_hidden_state = torch.cat([latent, output_tense], dim=-1)
    decoder_initial_hidden_state = self.latent2hidden(decoder_initial_hidden_state)
    decoder_initial_cell_state = self.decoder.init_cell_state()

    decoder_input = torch.tensor([[SOS_TOKEN]], device=device)

    decoder_hidden_state = decoder_initial_hidden_state
    decoder_cell_state = decoder_initial_cell_state
    pred_output = []

    while True:
        output, decoder_hidden_state, decoder_cell_state = self.decoder(decoder_input, decoder_hidden_state, decode

        if torch.argmax(output).cpu().detach().numpy() == EOS_TOKEN:
            break
        pred_output.append(torch.argmax(output).cpu().detach().numpy().item())
        decoder_input = torch.argmax(output).unsqueeze(0).unsqueeze(0)

    return pred_output
```

7. **Generate from Gaussian noise**

This function is similar with the evaluate function except the whole encoder and the initialization of decoder state. Instead of using output of encoder to derive latent, this function directly use Gaussian noise as latent. The rest is the same.

```
def generate_gaussian(self, latent, tense):
    # encoder initial state
    tense = self.tense_embedding(tense).unsqueeze(1)

    # decoder initial state
    decoder_initial_hidden_state = torch.cat([latent, tense], dim=-1)
    decoder_initial_hidden_state = self.latent2hidden(decoder_initial_hidden_state)
    decoder_initial_cell_state = self.decoder.init_cell_state()

    decoder_input = torch.tensor([[SOS_TOKEN]], device=device)

    decoder_hidden_state = decoder_initial_hidden_state
    decoder_cell_state = decoder_initial_cell_state
    pred_output = []

    while True:
        output, decoder_hidden_state, decoder_cell_state = self.decoder(decoder_input, decoder_hidden_state, decod

        if torch.argmax(output).cpu().detach().numpy() == EOS_TOKEN:
            break
        pred_output.append(torch.argmax(output).cpu().detach().numpy().item())
        decoder_input = torch.argmax(output).unsqueeze(0).unsqueeze(0)

    return pred_output
```

- Specify the hyperparameters

1. **EPOCHS: 300**
2. **LR: 0.05**
3. **HIDDEN_SIZE: 256**
4. **LATENT_SIZE: 32**
5. **VOCAB_SIZE: 28**
6. **CONDITION_EMBEDDING_SIZE: 8**
7. **TEACHER_FORCING_RATIO**

I let the teacher forcing ratio linear decreasing along with the epoch.

```
def _teacher_forcing_ratio(epoch, total_epoch):
    return 1 - epoch / total_epoch
```

8. **KL_WEIGHT**

Monotonic method is linear increasing along with the epoch until period (200). After period, KL weight will become 1.

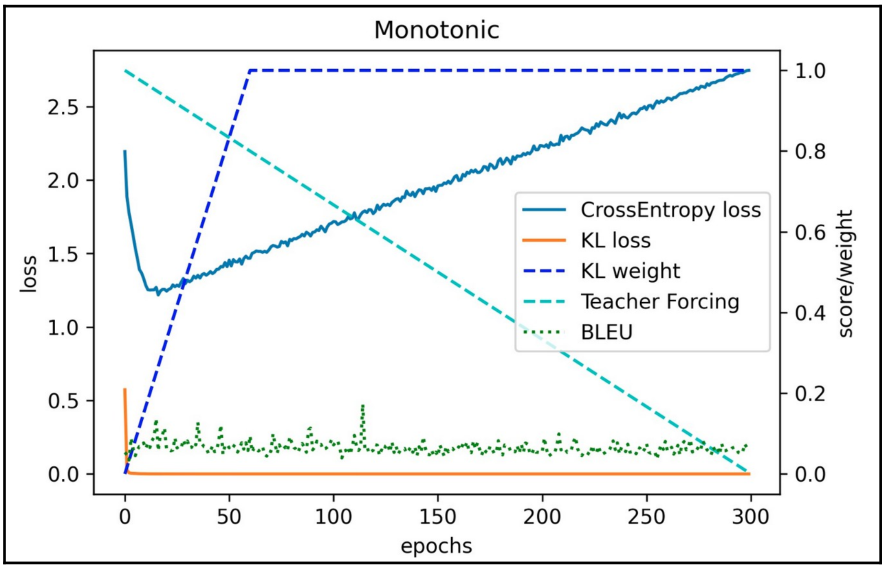
Cyclical method is a sigmoid like function between a period (0~200) and use constants to manipulate the slope (-10, 20). In next period (200~400), KL weight will become 1.

```
def _kl_weight(epoch, method, period):
    if method == 'Monotonic':
        return min(1, epoch / period)
    if method == 'Cyclical':
        if int(epoch / period) % 2 == 1:
            return 1
        else:
            return sigmoid(-10 + (epoch % period) * 20 / period)
```

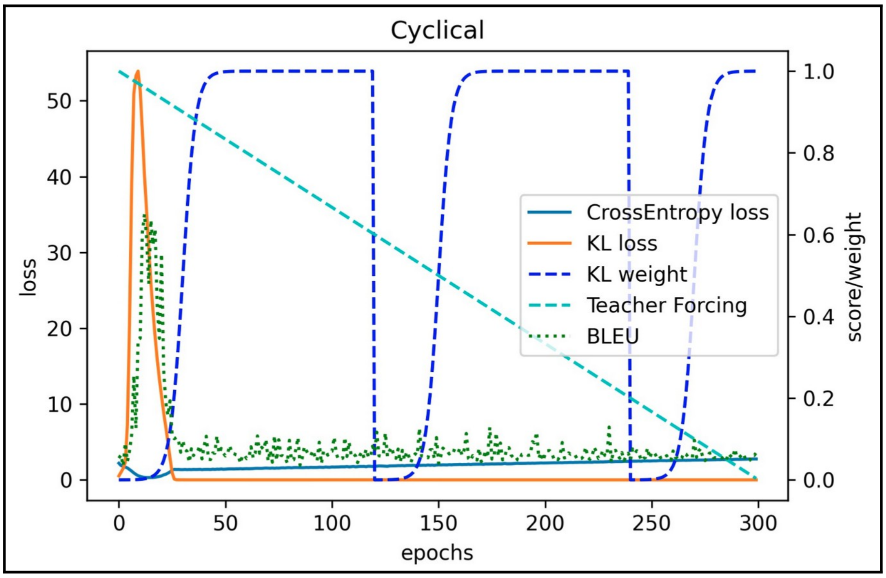
- Results and discussion

- Show your results of tense conversion and generation and Plot the Crossentropy loss, KL loss and BLEU-4 score curves during training

1. **Monotonic**



2. **Cyclical**



```
input:abandon
target:abandoned
prediction:abandoned

input:abet
target:abetting
prediction:abetting

input:begin
target:begins
prediction:begins

input:expend
target:expends
prediction:expends

input:sent
target:sends
prediction:sents

input:split
target:splitting
prediction:splitting

input:flared
target:flare
prediction:flare

input:functioning
target:function
prediction:function

input:functioning
target:functioned
prediction:functioned

input:healing
target:heals
prediction:heals

Average BLEU-4 score : 0.928574404296988
```

```
Gaussian score : 0.32
['outrug', 'outruns', 'outrugging', 'outsell']
['thank', 'thanks', 'thanking', 'thanked']
['total', 'totals', 'totalling', 'totalled']
['adjus', 'adjusts', 'adjoining', 'adjoined']
['advocate', 'advocates', 'advocating', 'advocated']
['caroate', 'convinces', 'caroping', 'captured']
['furge', 'furns', 'furgning', 'flaunted']
['obtrust', 'officiates', 'overcaming', 'objected']
['excuse', 'impresses', 'imetreting', 'improvised']
['enact', 'enacts', 'enacting', 'enacted']
['doubt', 'doubts', 'decoming', 'doubted']
['enclose', 'encourages', 'enclosing', 'encouraged']
['mistrust', 'misdeals', 'stamming', 'mistrusted']
['obtact', 'adopts', 'absorbing', 'adopted']
['embre', 'emerges', 'emplying', 'forsweared']
['haunt', 'haunts', 'haunting', 'haunted']
['excanate', 'excanates', 'exacuring', 'excame']
['jurk', 'jurks', 'infuriating', 'jurked']
['befit', 'befits', 'befitting', 'befitted']
['snore', 'snorts', 'struggling', 'snowed']
['figure', 'figures', 'figuring', 'forswieed']
['necrease', 'necessits', 'necreiving', 'necreased']
['enjoy', 'enjoys', 'enjoying', 'enjoyed']
['adjurt', 'adjuds', 'adjuring', 'adjudged']
['deminate', 'defits', 'deminating', 'deminded']
['waste', 'wastes', 'wasting', 'vaughteed']
['occur', 'occurs', 'occurring', 'occurred']
['improvise', 'improvises', 'improvising', 'improvised']
['heas', 'alurs', 'allotting', 'allotted']
['pares', 'pares', 'pared', 'pared']
['demonstrate', 'demonstrates', 'demonstrating', 'demonstrated']
['burst', 'bursts', 'bursting', 'funttipled']
['yell', 'yells', 'eliving', 'yell']
['delude', 'deludes', 'deluding', 'deluded']
['smail', 'smails', 'smashing', 'smacked']
['thrash', 'arches', 'telling', 'thinkeed']
['occept', 'improvises', 'appointing', 'improvised']
['yield', 'arlishes', 'arlishing', 'arlished']
['bury', 'burys', 'burying', 'buryed']
['stiffen', 'stiffens', 'skids', 'skid']
['withdraw', 'widens', 'widening', 'widened']
['exhilarate', 'exhilarates', 'exhilarating', 'exhilarated']
```

```
['jenk', 'jenks', 'jenkrning', 'jenkrned']
['began', 'began', 'begains', 'begins']
['forgive', 'forgives', 'forgiving', 'forgive']
['upset', 'upsets', 'upsetting', 'upset']
['snafk', 'snafks', 'snapping', 'snapped']
['balk', 'balks', 'bluttering', 'bluttered']
['imitate', 'imitates', 'imitating', 'imitated']
['switch', 'swings', 'switching', 'switched']
['splung', 'splungs', 'splunging', 'spluted']
['adjourn', 'adjourns', 'advocating', 'advocated']
['deny', 'streeks', 'streeking', 'streeked']
['mitter', 'mitters', 'mittering', 'mitstabled']
['bind', 'bindles', 'yincing', 'finhered']
['kneel', 'kneels', 'neglieting', 'shrugged']
['giggle', 'giggles', 'giggling', 'giggled']
['invent', 'invents', 'inventing', 'invorted']
['disturb', 'disturbs', 'disturbing', 'disturbed']
['yearn', 'yearns', 'yearning', 'yearned']
['enclose', 'encloses', 'enclosing', 'enclosed']
['request', 'requests', 'trembling', 'requested']
['tors', 'affords', 'arousing', 'adjourned']
['perceive', 'preaches', 'perceiving', 'perceived']
['yield', 'yields', 'approximating', 'yielded']
['furnish', 'furnishes', 'furnishing', 'furnished']
['kneep', 'kneels', 'kneeping', 'knowed']
['wiel', 'knows', 'jecking', 'jecked']
['total', 'totals', 'totalling', 'totalled']
['stammer', 'stammers', 'stammering', 'snatched']
['doug', 'dougs', 'doubling', 'doubled']
['declare', 'declares', 'declaring', 'declared']
['thrust', 'thrusts', 'thrusting', 'thrilled']
['abdicate', 'abdicates', 'abdicating', 'abducted']
['number', 'numbers', 'numbering', 'numbered']
['withdraw', 'withdraws', 'wettifying', 'withdrawed']
['inform', 'informs', 'informing', 'informed']
['inflit', 'inlays', 'inlaying', 'inlay']
['asphalt', 'appeases', 'asphalting', 'appealt']
['yearn', 'yearns', 'yearning', 'yearned']
['withstand', 'withstands', 'withdrawing', 'withdeased']
['constrain', 'constrains', 'constraining', 'constrained']
['furnish', 'furnishes', 'furnishing', 'furnished']
```

```
['feign', 'feigns', 'feigning', 'feigned']
['necry', 'nectoins', 'necipitating', 'enclosed']
['install', 'installs', 'installing', 'installed']
['disqualify', 'dismembers', 'dismembering', 'dismembered']
['yeach', 'yeacts', 'bending', 'jent']
['crush', 'crushes', 'crushing', 'crushed']
['jim', 'jims', 'mispleating', 'misproved']
['peck', 'jerks', 'pecking', 'proceeded']
['enable', 'enables', 'enabling', 'enabled']
['record', 'records', 'recovering', 'recorded']
['arrsay', 'arrsags', 'arresting', 'arrested']
['worry', 'wrotes', 'wrotihned', 'forgetted']
['expend', 'expends', 'expanding', 'expanded']
['institute', 'infuriates', 'instituting', 'insuired']
['putter', 'emits', 'assuring', 'putted']
['groan', 'groans', 'groaning', 'groaned']
['progress', 'progresses', 'progressing', 'progressed']
```

- Discuss the results according to your setting of teacher forcing ratio, KL weight, and learning rate

After training the monotonic method, I found out that both bleu score of training and testing was low. Therefore, I manually tried different kl weight and I realized that if the kl weight is small in the first 20 epochs, then the bleu score of training and testing will be higher. That is the reason why I define the Cyclical method as a sigmoid like function.

The bleu score mostly depends on the cross entropy loss. The bleu score is higher when the kl loss is high that means the kl weight should be small and let the cross entropy loss be more lower. However, the kl weight is increasing during the training process, which would lead to small kl loss and the latent space would be more similar with N(0, I). Therefore, the best model for bleu score is from the beginning of the training process and the best model for Gaussian score is from the end of the training process.

In my opinion, my setting of teacher forcing ratio does not significantly affect the performance of the model. However, it is still helpful in the beginning of training to force the decoder to learn something.