- Introduction(5%)

In Lab5, we implemented a conditional seq2seq VAE for English tense conversion and generation. By using the seq2seq model in Lab4, we added additional 3 parts, i.e., blue words in Fig. 1, for CVAE. The totally architecture is shown in Fig. 1.

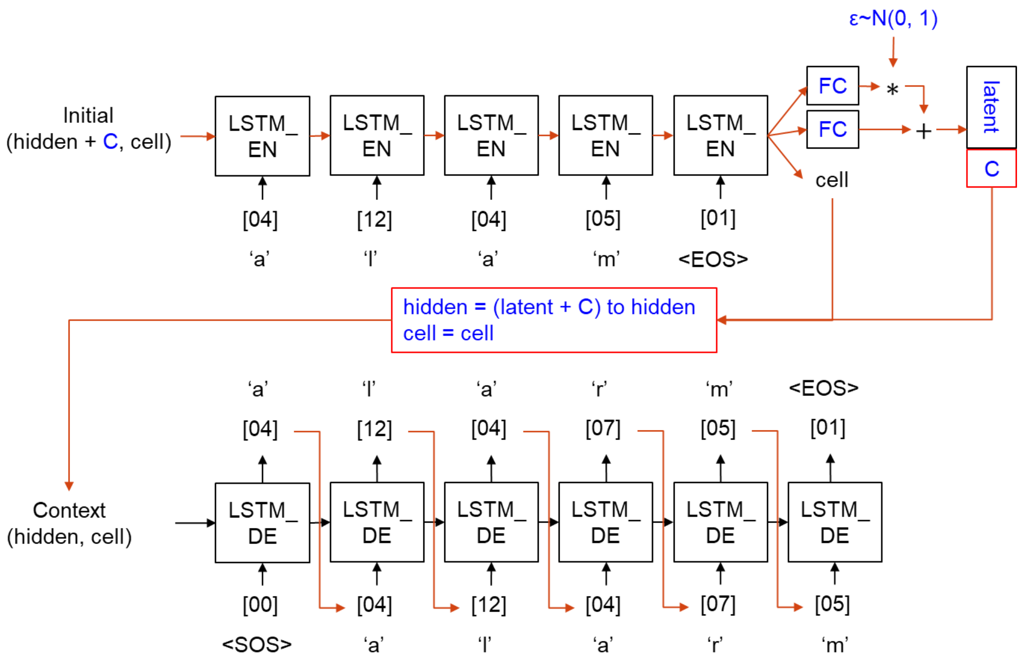


Fig. 1 The totally architecture of conditional seq2seq VAE

In training stage, the input of LSTM encoder is an index sequence of a word. The initial hidden state includes 'hidden + tense' and 'cell'. The output of LSTM encoder is ignore. The latest hidden inputs to two fully connected network to form a latent by normal sample ε . The input of LSTM decoder is <SOS>. The initial hidden state is hidden ('encoder latent + tense' to hidden dim) and cell (random initial value). The output of LSTM decoder is the predicted result which should be the same as the input.

In testing stage, we can input a word and tense and then get the word with the specific tense. Furthermore, because we restricted the latent be a sample of normal distribution, we can generate words with 4 tense by entering a Gaussian noise to LSTM decoder.

- Derivation of CVAE(5%)

Derivation of Conditional VAE：

在 Supervised learning 中，inference 就是在算 $P(z|x,c)$．但因為 $P(z|x,c)$ 通常是 interactable，或者計算複雜度太高，此時就可以找一個 $q(z|c)$ 去逼近 $p(z|x,c)$，讓 $P(z|x,c)$ 轉變成最佳化問題．這裏用 KL-divergence 來度量 $q(z|c)$ 和 $p(z|x,c)$ 的距離．

∴我們的目標就是希望 $KL(q(z|c)\|p(z|x,c))$ 越小越好．

而
$$
\begin{aligned}
KL(q(z|c)\|p(z|x,c)) &= -\sum_z q(z|c)\log\frac{P(z|x,c)}{q(z|c)} \\
&= -\sum_z q(z|c)\left[\log\frac{P(x,z|c)}{P(x|c)}\cdot\frac{1}{q(z|c)}\right] \\
&= -\sum_z q(z|c)\left[\log\frac{P(x,z|c)}{q(z|c)} - \log P(x|c)\right] \\
&= -\sum_z q(z|c)\cdot\log\frac{P(x,z|c)}{q(z|c)} + \log P(x|c)
\end{aligned}
$$

$$
\begin{aligned}
\therefore \log P(x|c) &= KL(q(z|c)\|p(z|x,c)) + \sum_z q(z|c)\cdot\log\frac{P(x,z|c)}{q(z|c)} \\
&= KL(q(z|c)\|p(z|x,c)) + \mathcal{L}(q)
\end{aligned}
$$

因為 $\log P(x|c)$ 和我們要找的 $q(z|c)$ 無關，所以 $\log P(x|c)$ 是固定的．由於 $KL\geq 0$，所以讓 KL 愈小愈好，等同於讓 $\mathcal{L}(q)$ 越大越好．我們的目標就是藉由最大化 $\mathcal{L}(q)$ 來迫使 $q(z|c)$ 接近 $P(z|x,c)$

(我們也可以看出 $\mathcal{L}(q)$ 是 $\log P(x|c)$ 的 lower bound)

∴目標函式 $= \mathcal{L}(q)$，假設 $q(z|x,c)$ 由參數為 θ' 的 NN 所產出．

$$
\begin{aligned}
\therefore \mathcal{L}(x,\theta') &= E_{z\sim q(z|x,c;\theta')}\left[\log P(x,z|c) - \log q(z|x,c;\theta')\right] \\
&= E_{z\sim q(z|x,c;\theta')}\left[\log P(x|z,c) + \log P(z|c) - \log q(z|x,c;\theta')\right] \\
&= E_{z\sim q(z|x,c;\theta')}\left[\log P(x|z,c)\right] + E_{z\sim q(z|x,c;\theta')}\left[\log\frac{P(z|c)}{q(z|x,c;\theta')}\right] \\
&= E_{z\sim q(z|x,c;\theta')}\left[\log P(x|z,c)\right] - KL(q(z|x,c;\theta')\|P(z|c))
\end{aligned}
$$

假設 $P(z|c)$ 是由參數為 θ 的 NN 所產出，則

$$
\mathcal{L}(x,\theta,\theta') = E_{z\sim q(z|x,c;\theta')}\left[\log P(x|z,c,\theta)\right] - KL(q(z|x,c;\theta')\|p(z|c))
$$

- Implementation details(15%)
  - Describe how you implement your model. (e.g. dataloader, encoder, decoder, etc)
    1. Dataloader

       Because we need a unique index per letter(字母) to use as the inputs and targets of the networks later, we built *Vocabulary* class to collect letters, transform a word to sequence of letters with unique index, and reverse transformation.

```
class Vocabulary(object):
    def __init__(self):
        self.char2index = {'SOS': 0, 'EOS': 1, 'PAD': 2, 'UNK': 3}
        self.char2count = {}
        self.index2char = {0: 'SOS', 1: 'EOS', 2: 'PAD', 3: 'UNK'}
        self.n_chars = 4  # Count SOS and EOS

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

    def addWord(self, word):
        for char in self.split_sequence(word):
            self.addChar(char)

    def addChar(self, char):...

    def split_sequence(self, word):...

    def sequence_to_indices(self, sequence, add_eos=False, add_sos=False):...

    def indices_to_sequence(self, indices):...
```

Then, we implemented dataloader for loading training data and testing data from text files.

```
class wordsDataset(data.Dataset):
    def __init__(self, train=True):
        if train:
            f = 'train.txt'
        else:
            f = 'test.txt'
        self.datas = np.loadtxt(f, dtype=np.str)

        if train:
            self.datas = self.datas.reshape(-1)
        else:
            '''
            sp -> p
            sp -> pg
            sp -> tp
            sp -> tp
            p  -> tp
            sp -> pg
            p  -> sp
            pg -> sp
            pg -> p
            pg -> tp
            '''
```

```python
        self.targets = np.array([
            [0, 3],
            [0, 2],
            [0, 1],
            [0, 1],
            [3, 1],
            [0, 2],
            [3, 0],
            [2, 0],
            [2, 3],
            [2, 1],
        ])

    # self.tenses = ['sp', 'tp', 'pg', 'p']
    self.tenses = [
        'simple-present',
        'third-person',
        'present-progressive',
        'simple-past'
    ]
    self.charvocab = Vocabulary()

    self.train = train
```

```python
def __len__(self):
    return len(self.datas)

def __getitem__(self, index):
    if self.train:
        c = index % len(self.tenses)
        # c_one_hot = np.zeros(len(self.tenses), dtype=int)
        # c_one_hot[c] = 1
        return self.charvocab.sequence_to_indices(self.datas[index], add_eos=True), c
    else:
        inp = self.charvocab.sequence_to_indices(self.datas[index, 0], add_eos=True)
        cinp = self.targets[index, 0]
        out = self.charvocab.sequence_to_indices(self.datas[index, 1], add_eos=True)
        cout = self.targets[index, 1]

        return inp, cinp, out, cout

def indices_to_word(self, indexes):
    word = self.charvocab.indices_to_sequence(indexes)
    return word
```

2. Encoder Class

   Encoder includes embedding, LSTM, FCs, and reparameterization trick. Embedding component embedded input to a fixed length vector. Then the fixed length vector and initial (hidden + C, cell) would be feed into LSTM. The output of LSTM includes predicted output and (hidden, cell), in which the former one is dropped and others would be used to form a (latent, cell). The outputs of FCs were operated with Gaussian noise to get latent.

```python
class EncoderRNN(nn.Module):
    def __init__(self, input_size, embedding_size, hidden_size, condition_size, condi_embed_size, latent_size,
                 num_layers=1, dropout=0.0):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.condition_size = condition_size
        self.condi_embed_size = condi_embed_size
        self.latent_size = latent_size
        self.num_layers = num_layers

        self.embedding = nn.Embedding(input_size, embedding_size)
        self.condi_embedding = nn.Embedding(condition_size, condi_embed_size)
        # nn.init.normal_(self.embedding.weight, 0.0, 0.2)
        # nn.init.normal_(self.condi_embedding.weight, 0.0, 0.2)

        self.lstm = nn.LSTM(embedding_size, hidden_size, num_layers, dropout=dropout)
        # self.dropout = nn.Dropout(dropout)
        self.mean = nn.Linear(hidden_size, latent_size)
        self.logvar = nn.Linear(hidden_size, latent_size)

    def forward(self, inp, hidden):
        # input = [input len, batch size]
        # embedded = self.dropout(self.embedding(input)).view(1, 1, -1)
        input_embedded = self.embedding(inp).view(1, 1, -1)
```

```python
        # input_embedded = [input len, batch size, emb dim]
        # hidden = (hidden_condition, cell_condition)
        output, hidden = self.lstm(input_embedded, hidden)

        # get (1, 1, hidden_size)
        m_hidd = self.mean(hidden[0])
        logvar_hidd = self.logvar(hidden[0])

        normal_sample = self.sample_latent()
        z_hidd = normal_sample * torch.exp(logvar_hidd) ** 0.5 + m_hidd

        m = m_hidd
        logvar = logvar_hidd
        z = z_hidd

        return output, hidden, z, m, logvar
```

```python
    def initHidden(self, condition):
        c = torch.LongTensor([condition]).to(device)
        condi_embedded = self.condi_embedding(c).view(1, 1, -1)

        h0 = torch.zeros(1, 1, (self.hidden_size - self.condi_embed_size)).to(device)
        c0 = torch.zeros(1, 1, self.hidden_size)

        h0 = torch.cat((h0, condi_embedded), dim=2)

        # hidden = (Variable(nn.Parameter(h0, requires_grad=True)).to(device),
        #           Variable(nn.Parameter(c0, requires_grad=True)).to(device))
        hidden = (Variable(h0).to(device),
                  Variable(c0).to(device))

        return hidden

    def sample_latent(self):
        return torch.normal(
            torch.FloatTensor([0] * self.latent_size),
            torch.FloatTensor([1] * self.latent_size)
        ).to(device)
```

3. Decoder Class

   Decoder includes embedding, LSTM, Linear components. Embedding component embedded input to a fixed length vector. Then the fixed length vector and encoder (‘latent + C’ to hidden, cell) would be feed into LSTM. The Linear output would be the predicted probability of each class.

```
class DecoderRNN(nn.Module):
    def __init__(self, output_size, embedding_size, hidden_size, latent_size, condi_embed_size,
                 num_layers=1, dropout=0.0):
        super(DecoderRNN, self).__init__()
        self.latent_size = latent_size
        self.condi_embed_size = condi_embed_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.embedding_size = embedding_size
        self.num_layers = num_layers

        self.latent_to_hidden = nn.Linear(
            latent_size + condi_embed_size, hidden_size
        )

        self.condi_embedding = nn.Embedding(condition_size, condi_embed_size)

        self.embedding = nn.Embedding(output_size, embedding_size)
        nn.init.normal_(self.embedding.weight, 0.0, 0.2)
        self.lstm = nn.LSTM(embedding_size, hidden_size, num_layers, dropout=dropout)
        self.out = nn.Linear(hidden_size, output_size)
```

```
    def forward(self, inp, hidden):
        # input = [1, batch size]
        # output = self.dropout(self.embedding(input)).view(1, 1, -1)
        output = self.embedding(inp).view(1, 1, -1)
        # embedded = [1, batch size, emb dim]
        output = F.relu(output)
        output, hidden = self.lstm(output, hidden)
        # output = self.softmax(self.out(output[0]))
        output = self.out(output[0])

        return output, hidden

    def initHidden(self, z, condition):
        c = torch.LongTensor([condition]).to(device)
        condi_embedded = self.condi_embedding(c).view(1,1,-1)

        latent_hidd = torch.cat((z, condi_embedded), dim=2)

        hidd = self.latent_to_hidden(latent_hidd)
        cell = torch.zeros(1, 1, self.hidden_size)
        cell = torch.FloatTensor(cell).to(device)

        hidden = (hidd, cell)
```

4. KL divergence loss

From the paper homework 1, we know the equation of KL(p1||p2) is list below if p1~N(μ_1, Σ_1) and p2~N(μ_2, Σ_2).

$$\mathfrak{D}_{\mathrm{KL}}[p_1 \,||\, p_2] = \frac{1}{2}\left[\log\frac{|\Sigma_2|}{|\Sigma_1|} - n + \mathrm{tr}\{\Sigma_2^{-1}\Sigma_1\} + (\mu_2 - \mu_1)^T\Sigma_2^{-1}(\mu_2 - \mu_1)\right]$$

Now, p1~N(m, var), p2~N(0,1), and n = 1. Therefore, we can use the equation of μ_1 and Σ_1 to calculate KL divergence.

```
def KL_loss(m, logvar):
    kldiv = torch.sum(0.5 * (-logvar + (m ** 2) + torch.exp(logvar) - 1))
    return kldiv
```

5. Train function (seq2seq part)

   In Train function, we implemented seq2seq model. Firstly, we initial encoder parameters and then feed each letter of input to encoder. The final output of Encoder includes predicted output, (hidden, cell), encoder latent which would be as hidden input of decoder, encoder mean, and encoder log variance. Then the loss of KL diverse was calculated for backforward later.

```
def train(input_tensor, condition, target_tensor, encoder, decoder, encoder_optimizer, decoder_optimizer,
          criterion, kldiv_weights, teacher_forcing_ratio=0.5):

    encoder.train()
    decoder.train()

    encoder_hidden = encoder.initHidden(condition)

    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()

    input_length = input_tensor.size(0)
    target_length = target_tensor.size(0)

    loss = 0
    # kldiv_loss = 0

    for ei in range(input_length):
        output, encoder_hidden, encoder_latent, encoder_mean, encoder_logvar = encoder(
            input_tensor[ei], encoder_hidden)

    kldiv_loss = KL_loss(encoder_mean, encoder_logvar)
```

   Secondly, we considered the decoder. The first input of decoder is the letter of <SOS>. If use_teacher_forcing is true, we use the target letter as next input. Otherwise, the predicted output would be used as next input of decoder. Here the loss of cross entropy was calculated for backforward later.

```
    decoder_input = torch.tensor([[SOS_token]], device=device)
    decoder_hidden = decoder.initHidden(encoder_latent, condition)

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

    if use_teacher_forcing:
        # Teacher forcing: Feed the target as the next input
        for di in range(target_length):
            decoder_output, decoder_hidden = decoder(
                decoder_input, decoder_hidden)

            loss += criterion(decoder_output, target_tensor[di].view(-1))
            decoder_input = target_tensor[di]  # Teacher forcing

    else:
        # Without teacher forcing: use its own predictions as the next input
        for di in range(target_length):
            decoder_output, decoder_hidden = decoder(
                decoder_input, decoder_hidden)
            topv, topi = decoder_output.topk(1)
            decoder_input = topi.squeeze().detach()  # detach from history as input

            loss += criterion(decoder_output, target_tensor[di].view(-1))
            if decoder_input.item() == EOS_token:
                break
```

   The cross entropy loss and weighted KL diverse loss were summated for back probagation.

```
    (loss + (kldiv_weights * kldiv_loss)).backward()

    encoder_optimizer.step()
    decoder_optimizer.step()

    return loss.item(), kldiv_loss.item()
```

6. Train Iterations function

   Here, we defined optimizer, loss function, and how to use data to train.

```
def trainIters(encoder, decoder, epochs, print_every=1000, learning_rate=0.01):
    start = time.time()
    ary_losses = []
    ary_kldlosses = []
    ary_bleu_score = []
    print_loss_total = 0  # Reset every print_every
    print_kldivloss_total = 0  # Reset every plot_every

    encoder_optimizer = optim.SGD(encoder.parameters(), lr=learning_rate)
    decoder_optimizer = optim.SGD(decoder.parameters(), lr=learning_rate)

    criterion = nn.CrossEntropyLoss(size_average=False)

    if not os.path.isdir('model'):
        os.mkdir('model')
    if not os.path.isdir('history'):
        os.mkdir('history')

    fp = open('history/history.txt', 'w')
    fp.close()
    data_len = len(train_dataset)
```

```
    for epoch in range(1, epochs+1, 1):
        kldiv_weights = KLD_cost_annealing(type=1, iteration=epoch)
        teacher_forcing_ratio = Teacher_Forcing_Ratio_Fcn(epoch)

        for iter in range(data_len):
            data = train_dataset[iter]
            inputs, condition = data

            input_tensor = torch.LongTensor(inputs).to(device)
            target_tensor = torch.LongTensor(inputs).to(device)

            loss, kldiv_loss = train(input_tensor, condition, target_tensor, encoder,
                                     decoder, encoder_optimizer, decoder_optimizer, criterion, kldiv_weights,
                                     teacher_forcing_ratio)
            print_loss_total += loss
            print_kldivloss_total += kldiv_loss
```

7. Evaluation functions

   BLEU-4 score would be used to evaluate the model results. BLEU (bilingual evaluation understudy) is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. The range of BLEU-4 score is (0, 1), while 1 means the best result.

```python
def compute_bleu(output, reference):
    cc = SmoothingFunction()
    if len(reference) == 3:
        weights = (0.33,0.33,0.33)
    else:
        weights = (0.25,0.25,0.25,0.25)
    return sentence_bleu([reference], output, weights=weights, smoothing_function=cc.method1)
```

Gaussian_score

```python
def Gaussian_score(words):
    words_list = []
    score = 0
    path = 'train.txt'#should be your directory of train.txt
    with open(path,'r') as fp:
        for line in fp:
            word = line.split(' ')
            word[3] = word[3].strip('\n')
            words_list.extend([word])
        for t in words:
            for i in words_list:
                if t == i:
                    score += 1
    return score/len(words)
```

Inference function

```python
def inference(encoder, decoder, input_tensor, input_condition, target_condition, target_len):
    with torch.no_grad():
        input_length = input_tensor.size()[0]
        encoder_hidden = encoder.initHidden(input_condition)

        for ei in range(input_length):
            output, encoder_hidden, encoder_latent, encoder_mean, encoder_logvar = encoder(
                input_tensor[ei], encoder_hidden)

        decoder_input = torch.tensor([[SOS_token]], device=device)  # SOS

        encoder_latent = encoder_latent.view(1, 1, -1)
        decoder_hidden = decoder.initHidden(encoder_latent, target_condition)
        decoded_words = []
        for di in range(target_len):
            decoder_output, decoder_hidden = decoder(
                decoder_input, decoder_hidden)
            topv, topi = decoder_output.data.topk(1)
            if topi.item() == EOS_token:
                decoded_words.append('<EOS>')
                break
            else:
                decoded_words.append(train_dataset.charvocab.index2char[topi.item()])

            decoder_input = topi.squeeze().detach()

        return decoded_words
```

Evaluated function

```python
def evaluate_bleu_score(encoder, decoder, mode='test'):
    bleu_score = 0.0
    if mode == 'test':
        print('======================')
    for idx in range(len(test_dataset)):
        data = test_dataset[idx]
        if test_dataset.train:
            inputs, input_condition = data
            targets = inputs
            target_condition = input_condition
        else:
            inputs, input_condition, targets, target_condition = data

        if mode == 'test':
            print('input:  ', test_dataset.indices_to_word(inputs))
            print('target:', test_dataset.indices_to_word(targets))

        input_tensor = torch.LongTensor(inputs).to(device)
        output_words = inference(encoder, decoder, input_tensor, input_condition, target_condition, len(targets))
        output_word = ''
        for k in range(len(output_words) - 1):
            output_word += str(output_words[k])
        if mode == 'test':
            print('pred:   ', output_word)
            print('======================')

        bleu_score += compute_bleu(output_word, test_dataset.indices_to_word(targets))
```

```python
    bleu_score /= len(test_dataset)
    return bleu_score
```

8. Steps of how to train the dataset:
    1. Create data loader by functions of dataloader.
    2. Create encoder instance (encoder1) by encoder class.
    3. Create decoder instance (decoder1) by decoder class.
    4. Call training iterations to train data by encoder1 and decoder1.

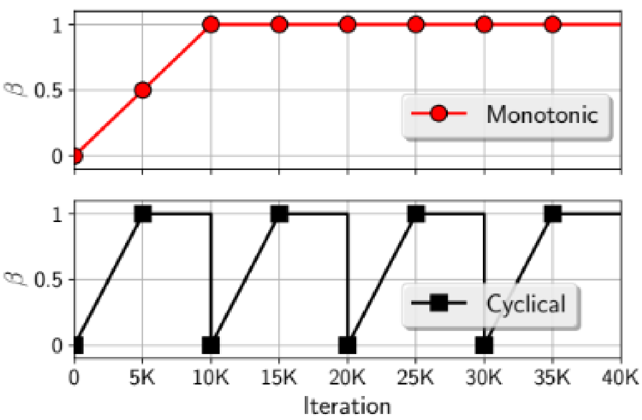
```python
train_dataset = wordsDataset(train=True)
test_dataset = wordsDataset(train=False)

embedding_size = 256
hidden_size = 256
condition_size = 4
condi_embed_size = 8
latent_size = 32
LR=1e-3
encoder1 = EncoderRNN(train_dataset.charvocab.n_chars, embedding_size, hidden_size,
                      condition_size, condi_embed_size, latent_size).to(device)
decoder1 = DecoderRNN(train_dataset.charvocab.n_chars, embedding_size, hidden_size,
                      latent_size, condi_embed_size).to(device)
trainIters(encoder1, decoder1, 500, print_every=1000, learning_rate=LR)
```

9. Steps of how to test the dataset:
    1. Load model
    2. Evaluate testing dataset

```python
def load_modle_and_evaluate(mode='test'):

    encoder1.load_state_dict(torch.load('encoder_best.dict'))
    decoder1.load_state_dict(torch.load('decoder_best.dict'))

    return evaluate_bleu_score(encoder1, decoder1, mode)
```

```
train_dataset = wordsDataset(train=True)
test_dataset = wordsDataset(train=False)

embedding_size = 256
hidden_size = 256
condition_size = 4
condi_embed_size = 8
latent_size = 32
LR=1e-3
encoder1 = EncoderRNN(train_dataset.charvocab.n_chars, embedding_size, hidden_size,
                      condition_size, condi_embed_size, latent_size).to(device)
decoder1 = DecoderRNN(train_dataset.charvocab.n_chars, embedding_size, hidden_size,
                      latent_size, condi_embed_size).to(device)

score_test = load_modle_and_evaluate(mode='test')
```

- Specify the hyperparameters (KL weight, teacher forcing ratio, etc.)
  1. embedding_size = 256
  2. hidden_size = 256
  3. condition_size = 4
  4. condi_embed_size = 8
  5. latent_size = 32
  6. LR=1e-3
  7. For KL cost annealing, we implemented two kinds curve of KL weight, the monotonic and cyclical, which are list below.



The code is implemented like following picture. Type==0 means using monotonic curve; type==1 means using cyclical curve. The slope is a key for getting high bleu score.

```python
def KLD_cost_annealing(type, iteration):
    # Monotonic
    if type == 0:
        slope = 0.001

        w = slope * iteration

        if w > 1.0:
            w = 1.0
    # Cyclic
    else:
        slope = 0.005
        period = 1.0 / slope * 2

        w = slope * (iteration % period)

        if w > 1.0:
            w = 1.0

    return w
```

8. Teacher forcing ratio

   The ratio is decreasing from 1.0 to 0.0.

```python
def Teacher_Forcing_Ratio_Fcn(iteration):
    # from 1.0 to 0.0
    slope = 0.01
    level = 10
    w = 1.0 - (slope * (iteration // level))
    if w <= 0.0:
        w = 0.0

    return w
```

- Notice: You must prove that your text generation is produced by Gaussian noise (paste/screenshot your code)

  The following codes are the parts of generating words with 4 tense.

```python
def decode_inference(decoder, z, target_condition, target_len):
    with torch.no_grad():
        decoder_input = torch.tensor([[SOS_token]], device=device)  # SOS

        z = z.view(1, 1, -1)
        decoder_hidden = decoder.initHidden(z, target_condition)

        decoded_words = []

        for di in range(target_len):
            decoder_output, decoder_hidden = decoder(
                decoder_input, decoder_hidden)
            topv, topi = decoder_output.data.topk(1)
            if topi.item() == EOS_token:
                decoded_words.append('<EOS>')
                break
            else:
                decoded_words.append(train_dataset.charvocab.index2char[topi.item()])

            decoder_input = topi.squeeze().detach()

        return decoded_words
```

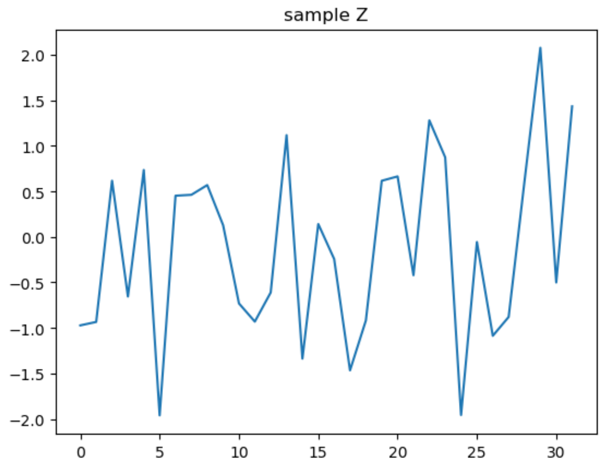
```python
def generate_word(decoder, z, condition, maxlen=20):
    decoder.eval()

    output_words = decode_inference(
        decoder, z, condition, target_len=maxlen
    )

    output_word = ''
    for k in range(len(output_words) - 1):
        output_word += str(output_words[k])

    return output_word


def show_noise(noise):
    plt.title('sample Z')
    plt.plot(list(noise))
    plt.show()
```

```python
def generate_test(decoder, noise):

    show_noise(noise)

    strs = []
    for i in range(len(train_dataset.tenses)):
        output_str = generate_word(decoder, noise, i)
        strs.append(output_str)

    return strs
```

Noise is created from the sample_latent function of EncoderRNN class.

```python
words = []
for k in range(100):
    noise = encoder1.sample_latent()
    four_tense_str = generate_test(decoder1, noise)
    print(four_tense_str)
    words.append(four_tense_str)

avg_gaussian_score = Gaussian_score(words)
print('Gaussian Score: ' + str(avg_gaussian_score))
```

Example of Gaussian noise.

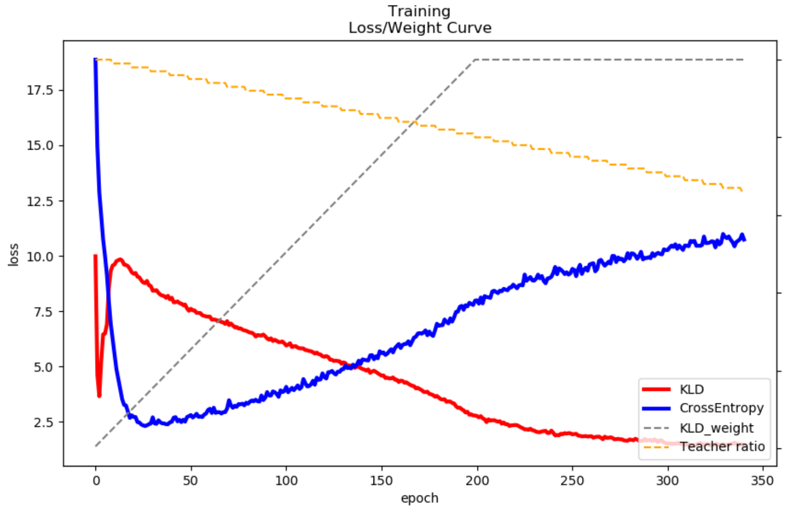


- Results and discussion(25%)
  - Plot the loss and KL loss curve while training and discuss the results. (5%)

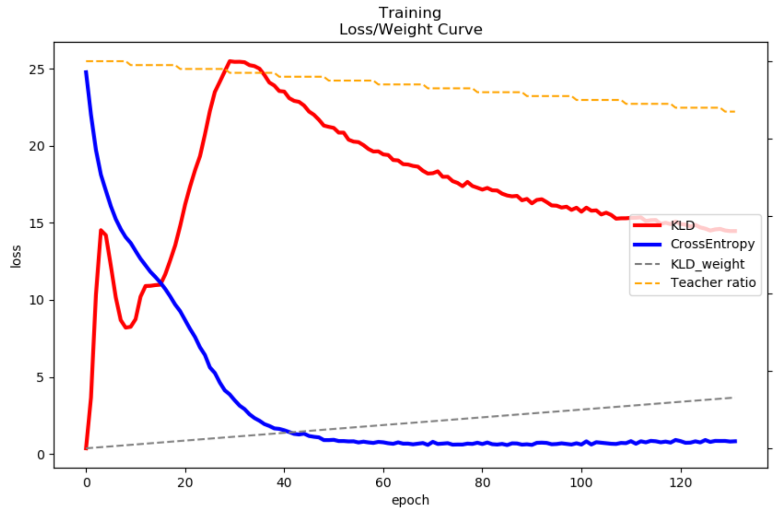
    1. Monotonic

       下圖是讓 KL weight 在第 200 個 epoch 就升到 1.0，可以看到，CrossEntropy 的 loss 在大約 30 個 epoch 就漸漸升高，根據後面的

實驗數據可以發現，這對 BLEU-4 score 相對不利

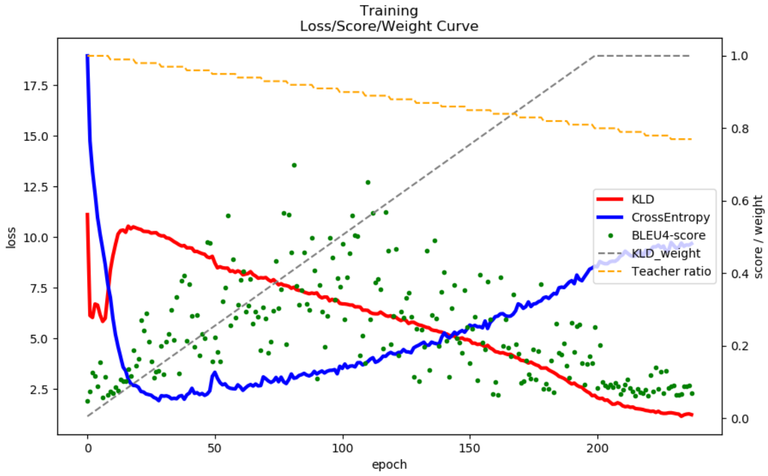


下圖是讓 KL weight 在第 1000 個 epoch 才升到 1.0，可以看到，CrossEntropy 的 loss 持續下降，但 KL loss 緩慢下降，根據後面的實驗數據可以發現，這對 BLEU-4 score 很有利，但對 Gaussian score 相對不利（表示目前的 q(z|c)還離 Gaussian distribution 很遠，不能當 p(z|x,c)的近似值）

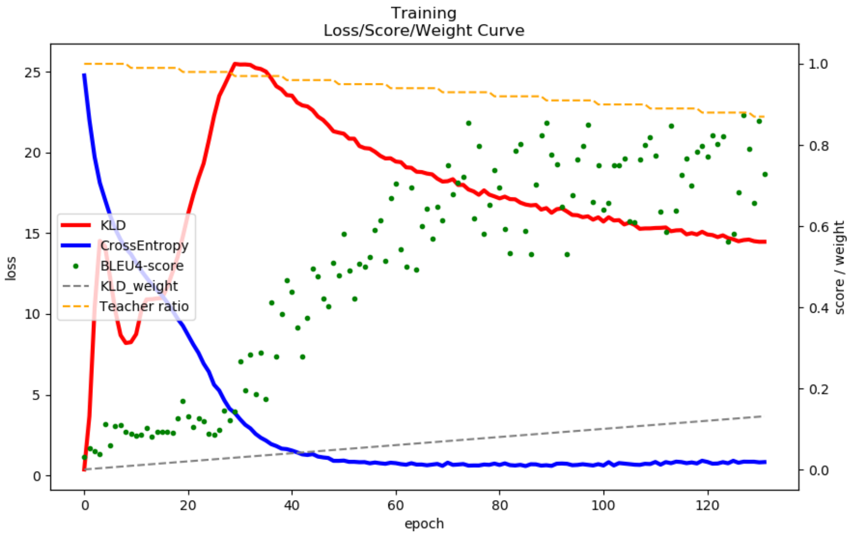


2. Cyclical

   Cyclical 的部分，因為時間關係尚未跑完，所以還未出現重複的波形，上網搜尋研究後了解，應會出現週期性的 pattern，且 BLEU-4 數值越來越高。目前我的結果還在第一個週期，最高的 BLEU-4 數值是 0.7002。Cyclical 的好處是，讓 CrossEntropy 有機會重新下降。



■ Plot the BLEU-4 score of your testing data while training and discuss the result. (20%)



最高的 BLEU-4 Score 會出現在 CrossEntropy 降得較低時，但由於我們限制 q 是 Gaussian distribution，而實際上又不是，所以此時 KL loss 通常值還很大。假設我們採用最高的 BLEU-4 Score，作為我們的最佳模型，在訓練的時候，可以得到 0.8608 的高分，但此時 Gaussian score 有 0.03 分，所以實際應用時需要找到兩者的平衡點。下面兩個範例分別為最佳 BLEU-4 Score 和兩者平衡的結果

範例 1:

範例 2:

```
C:\Users\bayul\AppData\Local
========================
input:  abandon
target: abandoned
pred:   abandoned
========================
input:  abet
target: abetting
pred:   batting
========================
input:  begin
target: begins
pred:   begins
========================
input:  expend
target: expends
pred:   expends
========================
input:  sent
target: sends
pred:   sents
========================
input:  split
target: splitting
pred:   splitting
========================
```

```
========================
input:  flared
target: flare
pred:   flare
========================
input:  functioning
target: function
pred:   finchuri
========================
input:  functioning
target: functioned
pred:   functioned
========================
input:  healing
target: heals
pred:   heals
========================
BLEU-4 score (test):  0.7939321233585442
```

```
['chormard', 'chormalds', 'chormaining', 'chormard']
['goude', 'goudgs', 'gouging', 'gouded']
['invet', 'instears', 'aiteeting', 'avetted']
['flearoat', 'frannows', 'flearoating', 'franker']
['spurt', 'spurs', 'spurting', 'spurt']
['megu', 'mets', 'megking', 'megut']
['intermeath', 'entertains', 'enterceating', 'entertained']
['single', 'sathers', 'sizzning', 'sathered']
['beloget', 'beloges', 'beloghing', 'beloghered']
['glink', 'glinks', 'glinking', 'jearkned']
['ghimble', 'ghickles', 'ghimblasting', 'ghimbled']
['wanse', 'wanses', 'wadening', 'wadeed']
['exply', 'explys', 'expinding', 'expayd']
Gaussian Score: 0.02
```

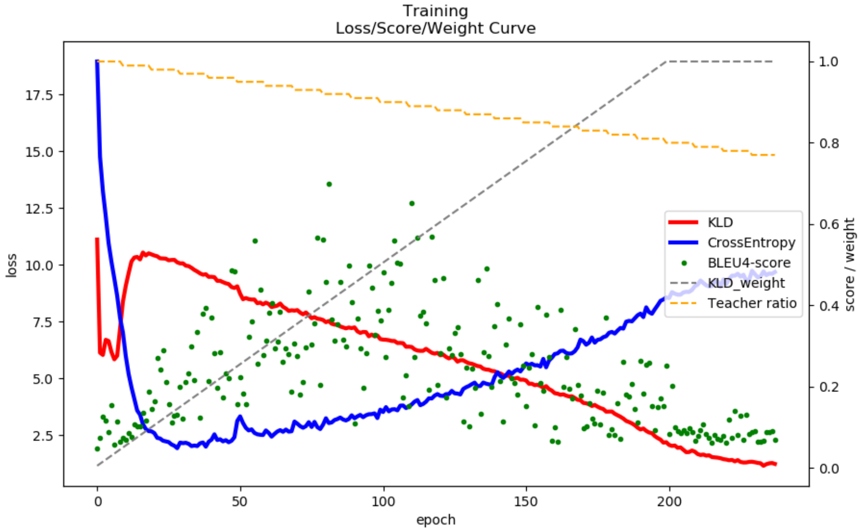
範例 2:

```
C:\Users\bayul\AppData\Local\Program
========================
input:  abandon
target: abandoned
pred:   abandoned
========================
input:  abet
target: abetting
pred:   hearing
========================
input:  begin
target: begins
pred:   begins
========================
input:  expend
target: expends
pred:   expends
========================
input:  sent
target: sends
pred:   slees
========================
input:  split
target: splitting
pred:   splitting
========================
input:  flared
target: flare
pred:   flare
```

```
========================
input:  functioning
target: functioned
pred:   found
========================
input:  healing
target: heals
pred:   heave
========================
BLEU-4 score (test):  0.5622219932331934
```

```
['cost', 'filches', 'feinting', 'cost']
['emit', 'emits', 'emitting', 'read']
['measure', 'wrinks', 'wrinkling', 'measured']
['foreshow', 'foreshows', 'foreshowing', 'fored']
['begin', 'begins', 'beginning', 'beginnled']
['run', 'runs', 'bulging', 'run']
['shook', 'shooks', 'outshoning', 'shook']
['misdeal', 'misdeals', 'adjing', 'adjudged']
['undergo', 'understands', 'undergoing', 'understood']
['pend', 'pledges', 'pledging', 'pled']
Gaussian Score: 0.22
```

Cyclical 的部分，因為時間關係尚未跑完，所以還未出現重複的波形，上網搜尋研究後了解，應會出現週期性的 pattern，且 BLEU-4 數值越來越高。目前我的結果還在第一個週期，最高的 BLEU-4 數值是 0.7002



- Notice: This part mainly focuses on your discussion, if you simply just paste your results, you will get a low score