DLP HW4

310551053 Chieh-Ming Jiang

August 12th 2021

1 Introduction

In this assignment, I am going to implement a sequence-to-sequence CVAE for English tense conversion and generation. Moreover,the encoder and decoder in the model are implemented with LSTM model in pytorch.
The training data is composed with 1227 words with 4 English tense(tread,treads,treading,trod). On the other hand, the testing data is composed with 2 words, which are input word and predicted output word.(healing,heals).

2 Derivation of CVAE

$$p(x, z|c) = p(x|c)\, p(z|x, c)$$

$$\log p(x, z|c) = \log p(x|c) + \log p(z|x, c)$$

$$\log p(x|c) = \log p(x, z|c) - \log p(z|x, c)$$

同乘 $q(z|c, x; \theta')$ 並做積分

$$\int q(z|c, x; \theta') \log p(x|c; \theta)\, dz = \log p(x|c; \theta)$$

$$= \int q(z|c, x; \theta') \log p(x, z|c; \theta)\, dz - \int q(z|c, x; \theta') \log p(z|x, c; \theta)\, dz$$

$$= \int q(z|c, x; \theta') \log p(x, z|c; \theta)\, dz - \int q(z|c, x; \theta') \log q(z|c, x; \theta')$$

$$+ \int q(z|c, x; \theta') \log q(z|c, x; \theta') - \int q(z|c, x; \theta') \log p(z|x, c; \theta)\, dz$$

$$= \mathcal{L}(x, q, \theta|c) + KL(q(z|x, c; \theta') \,||\, p(z|x, c; \theta))$$

$$\mathcal{L}(x, q, \theta|c) = \log p(x|c; \theta) - KL(q(z|x, c; \theta') \,||\, p(z|x, c; \theta))$$

$$= \int q(z|c) \log p(x|z, c; \theta')\, dz + \int q(z|c) \log q(z|c; \theta)\, dz$$

$$+ KL(q(z|c) \,||\, p(z|x, c; \theta))$$

$$= E_{z \sim q(z|x,c,\theta')} \log p(x|z, c\theta) - KL(q(z|x, c; \theta') \,||\, p(z|x, c; \theta)) \quad \#$$

Figure 1: Derivation of CVAE

3 Derivation of KL Divergence loss

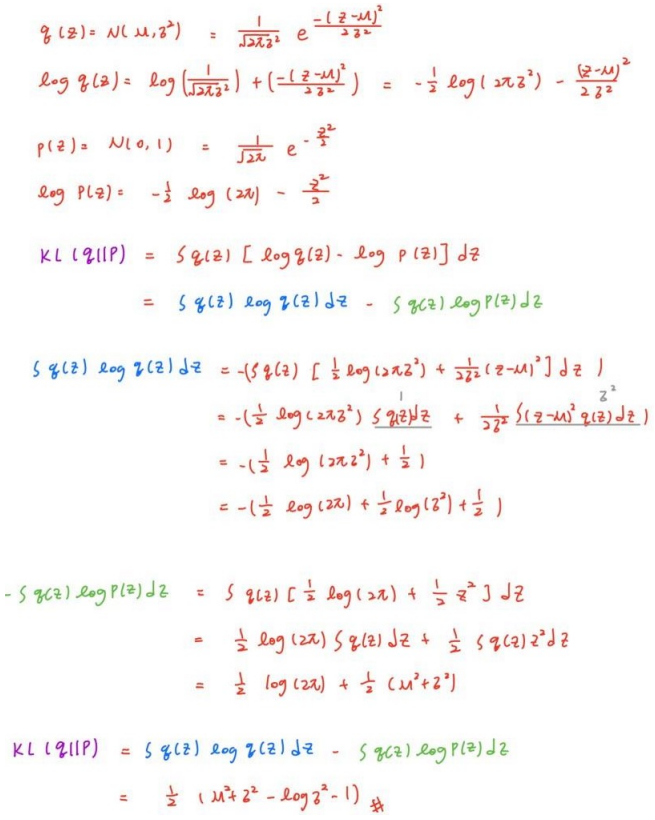

Figure 2: KL Divergence loss

4 Implementation details

1. Dataloader
   Firstly, create a dictionary class to map the English alphabets to numbers.

```python
class Dictionary:

    def __init__(self):
        self.char2idx = {'SOS': 0 , 'EOS': 1}
        self.idx2char = {0: 'SOS' , 1: 'EOS' }
        self.tense2idx={'sp':0 , 'tp':1, 'pg':2, 'p':3}
        self.idx2tense={0:'sp' , 1: 'tp', 2:'pg', 3:'p'}
        alphabets = 'abcdefghijklmnopqrstuvwxyz'
        for c in alphabets:
            if c in self.char2idx:
                continue
            idx = len(self.char2idx)
            self.char2idx[c] = idx
            self.idx2char[idx] = c

    def encode(self, w):
        return torch.tensor(
              [ self.char2idx[c] for c in w ]
            + [ EOS_token ],
        device=device).view(-1, 1)

    def decode(self, t):
        word = []
        for char in t.view(-1):
            word.append(self.idx2char[char.item()])
        return ''.join(word)
```

Figure 3: Dictionary

Then, create the train dataset and test dataset to get the words and tenses.

```python
class TrainDataset(Dataset):

    def __init__(self, path):
        self.data = np.loadtxt(path, dtype=np.str).reshape(-1)
        self.dict = Dictionary()

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

    def __getitem__(self, index):
        return self.data[index], index % len(self.dict.tense2idx)
```

Figure 4: Train dataset

```python
class TestDataset(Dataset):

    def __init__(self, path):
        self.data = np.loadtxt(path, dtype=np.str)
        self.dict = Dictionary()
        self.target = [
            ['sp',  'p'],
            ['sp',  'pg'],
            ['sp',  'tp'],
            ['sp',  'tp'],
            ['p',   'tp'],
            ['sp',  'pg'],
            ['p',   'sp'],
            ['pg', 'sp'],
            ['pg', 'p'],
            ['pg', 'tp']
        ]

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

    def __getitem__(self, index):

        return self.data[index][0], self.dict.tense2idx[self.target[index][0]],\
            self.data[index][1], self.dict.tense2idx[self.target[index][1]]
```

Figure 5: Test dataset

2. Encoder
   The encoder embeds the input to tensor and do LSTM to get the hidden state. The output of encoder is not important.

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

    def forward(self, input, hn, cn):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, (hn,cn) = self.lstm(output, (hn, cn))
        return output, hn, cn
```

Figure 6: Encoder

3. Decoder
   The decoder also embeds the input then do relu, LSTM, fully connected, and softmax to get the predicted output.

```python
#Decoder
class DecoderRNN(nn.Module):
    def __init__(self, intput_size, hidden_size ):
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(intput_size, hidden_size)
        self.lstm = nn.LSTM(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, intput_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hn, cn):
        output = self.embedding(input).view(1, 1, -1)
        output = F.relu(output)
        output, (hn, cn) = self.lstm(output, (hn, cn))
        output = self.out(output[0])
        output = self.softmax(output)
        return output, hn, cn
```

Figure 7: Decoder

4. Reparameterization trick
   When getting the mean and variance from encoder, I multiply a normal distribution on variance and shift it by mean to make it differentiable on the following steps.

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

def reparameterization(self, mean, var):
    return self.sampling() * torch.exp(log_var / 2) + mean
```

Figure 8: Reparameterization trick

5. Seq2seq-CVAE
   In sequence-to-sequence CVAE model, we wish to learn a model that can recognize the English word and tense to predict the correct output. First of all, we give the English word and tense to the encoder. So we can get the mean and log variance as latent. Here we take log variance instead of variance, I think it's because the variance could be a pretty small value

which is near to 0, so it may be hard update the weight during back-propahgation. Such little trick may help us train the model. Then we can give the latent state and SOStoken as input to the decoder to generate the corresponding word.

```
class CVAE(nn.Module):
    def __init__(self, input_size, hidden_size, condition_input_size , conditoin_output_size,  latent_size):
        super(CVAE, self).__init__()
        self.condition_embedding = nn.Embedding(condition_input_size, conditoin_output_size)
        self.encoder = EncoderRNN(input_size, hidden_size)
        self.mean = nn.Linear(hidden_size, latent_size)
        self.logvar = nn.Linear(hidden_size, latent_size)
        self.latent2hidden = nn.Linear(latent_size + conditoin_output_size, hidden_size)
        self.latent2cell = nn.Linear(latent_size + conditoin_output_size, hidden_size)
        self.decoder = DecoderRNN(input_size, hidden_size)
        self.hidden_size = hidden_size
        self.conditoin_output_size = conditoin_output_size
        self.condition_input_size = condition_input_size
        self.latent_size = latent_size
```

Figure 9: CVAE

```
    def initHidden(self, input_condition):
        h0 = torch.zeros(self.hidden_size - self.conditoin_output_size, device=device).view(1, 1, -1)
        c0 = torch.zeros(self.hidden_size, device=device).view(1, 1, -1)
        h0 = torch.cat((h0, self.condition(input_condition)), dim=-1)
        return h0, c0

    def KL_loss(self, mean, log_var):
        return torch.mean(0.5 * (-log_var + (mean ** 2) + torch.exp(log_var) - 1))

    def condition(self, c):
        return self.condition_embedding(torch.LongTensor([c]).to(device)).view(1, 1, -1)
```

Figure 10: CVAE

```python
def forward(self, input, input_condition, target, target_condtion, teacher_forcing=True):
    hn, cn = self.initHidden(input_condition)
    for i in range(input.size(0)):
        _, hn, cn = self.encoder(input[i], hn,cn )
    mean = self.mean(hn)
    log_var = self.logvar(hn)
    latent = self.sampling() * torch.exp(log_var / 2) + mean
    hn = self.latent2hidden(torch.cat((latent, self.condition(target_condtion)), dim=-1).reshape(-1)).view(1, 1, -1)
    cn = self.latent2cell(torch.cat((latent, self.condition(target_condtion)), dim=2).reshape(-1)).view(1, 1, -1)

    decoder_input = torch.tensor([[SOS_token]], device=device)
    prediction = []
    for i in range(MAX_LENGTH):
        output, hn, cn = self.decoder(decoder_input, hn, cn)
        if teacher_forcing:
            if i == target.size(0):
                break
            prediction.append(output)
            decoder_input = target[i]
        else:
            prediction.append(output)
            topv, topi = output.topk(1)
            decoder_input = topi.squeeze().detach()
            if decoder_input.item() == EOS_token:
                break
    return torch.stack(prediction), mean, log_var
```

Figure 11: CVAE

6. Genrate words
   To generate words, I use the sampling funtion in fig 8, which is a gaussian noise, to get the latent z to generate the words.

```python
def gen_word(self, dataset):
    self.eval()
    z = self.sampling()
    word = []
    with torch.no_grad():
        for tense in range(4):
            hn =  self.latent2hidden(torch.cat((z, self.condition(tense)), dim=2).reshape(-1)).view(1, 1, -1)
            cn =  self.latent2cell(torch.cat((z, self.condition(tense)), dim=2).reshape(-1)).view(1, 1, -1)

            input = torch.tensor([[SOS_token]], device=device)
            prediction = []
            for i in range(MAX_LENGTH):
                output, hn, cn = self.decoder(input, hn, cn)
                prediction.append(output)
                topv, topi = output.topk(1)
                input = topi.squeeze().detach()
                if input.item() == EOS_token:
                    break

            output = torch.stack(prediction)
            output = dataset.dict.decode(output.argmax(dim=2).view(-1, 1))
            word.append(output[:-3])
    return word
```

Figure 12: Genrate words

7. KL annealing

During the training procedure, there are cross entropy loss and kld loss. The kld loss depicts the distance of output distribution and N(0,1). Here I apply a kld weight to the loss function with two different methods.

$$loss = crossEntropyLoss + kldLoss * kldWeight$$

```python
def get_teacher_forcing_ratio(self, epoch):
    return 1 -(1 / (500))* (epoch)

def get_kl_weight(self, epoch, type_):
    if type_ == 'monotonic':
        return 0.1*epoch if epoch < 10 else 1

    elif type_ == 'cyclical':
        return 1 if epoch % 16 > 10  else (epoch % 10) * 0.1
```

Figure 13: TF ratio and KLD weight

8. hyperparameters
   hidden size : 256
   vocab size : 28
   conditoin output size : 8
   latent size : 32
   teacher forcing ratio : monotonic
   KLD weight : monotonic / cyclical
   learning rate : 0.05

5 Results and discussion

There are two loss terms in the model, which are cross entropy loss and KL loss. The cross entropy loss is used to calculate the loss of tense conversion, while KL loss is used to calculate the loss of words generation.In other words, if the cross entropy is small, the task of tense conversion performs better. If the KL loss is small, the task of word generation performs better, From the loss function:

$$loss = crossEntropyLoss + kldLoss * kldWeight$$

When the kld weight is large, we wish to map the output distribution to N(0,1), so the gaussian score will be higher, but it may influence the score in BLEU-4. As a consequence, deciding a proper kld weight is important to do the two tasks well at the same time.

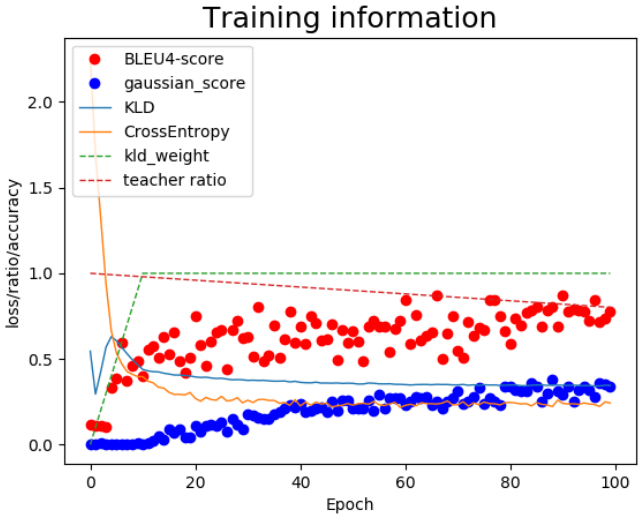

Figure 14: Monotonic result

Observing from the monotonic result, we can see the model try to minimize the cross entropy loss at first few epochs. So the score in BLEU-4 rises, but the score in gaussian is terrible. When the kld weight is large, the model try to map the output to N(0,1) and the gaussian score arises.

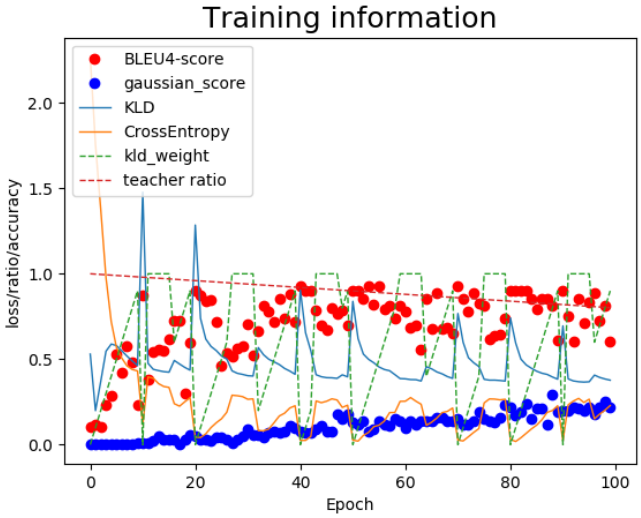

Figure 15: Cyclical result

Different from the monotonic method, the cyclical method adjusts the kld weight with a cycle. That is, it take turns to minimize cross entropy loss and kld loss. As we can see from the figure above, when kld weight is large, the loss of kld is decreasing. On the other hand, when kld weight is small, the loss of cross entropy is decreasing. After training several epochs, both cross entropy loss and kld loss are decreasing, and both scores of two task are increasing.

```
input: abandon             , target: abandoned           , output: abandoned
input: abet                , target: abetting            , output: abetting
input: begin               , target: begins              , output: begins
input: expend              , target: expends             , output: expends
input: sent                , target: sends               , output: sends
input: split               , target: splitting           , output: splitting
input: flared              , target: flare               , output: flare
input: functioning         , target: function            , output: function
input: functioning         , target: functioned          , output: unfolded
input: healing             , target: heals               , output: heals
BLEU4 score:  0.907159692505125
```

Figure 16: BLEU-4

The line starts with an arrow mark means the generated words are in training data and predicted correctly.

```
          ['distribute', 'distributes', 'distribing', 'distributed']
          ['send', 'sends', 'sensing', 'sensed']
          ['exite', 'exits', 'exiting', 'exited']
          ['flegge', 'flegges', 'fleggning', 'flegged']
          ['degain', 'degances', 'deganing', 'deganced']
          ['blect', 'blets', 'blekting', 'blek']
          ['descond', 'descones', 'desconing', 'desconed']
          ['glue', 'gleaps', 'glaring', 'gleaped']
          ['pinch', 'pinchs', 'pinching', 'pinched']
          ['stell', 'stells', 'stelling', 'stlealed']
          ['wiile', 'wicks', 'wiilling', 'wicked']
          ['kisse', 'hinds', 'hoping', 'hopined']
--->      ['convert', 'converts', 'converting', 'converted']
--->      ['resist', 'resists', 'resisting', 'resisted']
--->      ['invent', 'invents', 'inventing', 'invented']
          ['crumble', 'crumbles', 'charactering', 'characterized']
--->      ['bewail', 'bewails', 'bewailing', 'bewailed']
          ['acquit', 'acquits', 'acquitting', 'assuried']
          ['climb', 'clases', 'clasping', 'cleased']
          ['sccust', 'sccusts', 'sccusing', 'sccusted']
          ['strive', 'straves', 'striving', 'straved']
          ['confound', 'conforms', 'conforming', 'confounded']
          ['acherie', 'acheries', 'acheriening', 'acheried']
--->      ['preserve', 'preserves', 'preserving', 'preserved']
          ['exaberate', 'exabers', 'exaberating', 'exasped']
--->      ['stoop', 'stoops', 'stooping', 'stooped']
          ['adjoin', 'adjoins', 'abdicing', 'abdicated']
          ['pretire', 'preties', 'pretiring', 'pretiered']
          ['subside', 'hints', 'hinting', 'hinted']
          ['plan', 'plans', 'planing', 'plan']
          ['fin', 'fins', 'poining', 'finked']
          ['instact', 'instacts', 'instacting', 'instacted']
--->      ['investigate', 'investigates', 'investigating', 'investigated']
          ['learch', 'learches', 'learning', 'learched']
```

Figure 17: Gaussian score

```
--->     ['shove', 'shoves', 'shoving', 'shoved']
         ['bestit', 'bumbles', 'bestowing', 'bumbled']
--->     ['blast', 'blasts', 'blasting', 'blasted']
         ['fathing', 'fathers', 'fathering', 'fatheried']
--->     ['stimulate', 'stimulates', 'stimulating', 'stimulated']
--->     ['flick', 'flicks', 'flicking', 'flicked']
--->     ['push', 'pushes', 'pushing', 'pushed']
         ['indearate', 'indears', 'indearing', 'indeared']
         ['patch', 'patches', 'patching', 'patched']
--->     ['click', 'clicks', 'clicking', 'clicked']
         ['feat', 'feats', 'feating', 'feated']
--->     ['broaden', 'broadens', 'broadening', 'broadened']
         ['improte', 'improtes', 'improving', 'imported']
--->     ['thicken', 'thickens', 'thickening', 'thickened']
         ['counter', 'counters', 'countering', 'counsel']
--->     ['sneak', 'sneaks', 'sneaking', 'sneaked']
         ['awake', 'awakes', 'awaking', 'awaked']
         ['praye', 'prays', 'praying', 'prayed']
         ['wield', 'wielces', 'wielding', 'wielded']
--->     ['croon', 'croons', 'crooning', 'crooned']
         ['skim', 'skims', 'skimming', 'skimulated']
         ['proticie', 'proticies', 'proticing', 'proticied']
         ['gate', 'gates', 'gatping', 'gate']
         ['unfoct', 'unfocts', 'unfocing', 'unfocted']
         ['creat', 'creates', 'creating', 'created']
         ['tug', 'tugs', 'tugging', 'teard']
         ['frainwash', 'frainwashes', 'frainwashing', 'flourished']
--->     ['wiggle', 'wiggles', 'wiggling', 'wiggled']
         ['finish', 'finishes', 'finulating', 'finished']
         ['sawl', 'sawls', 'sawling', 'sawl']
         ['allow', 'allows', 'allowing', 'abhorred']
--->     ['await', 'awaits', 'awaiting', 'awaited']
--->     ['brain', 'brains', 'braining', 'brained']
```

Figure 18: Gaussian score

```
         ['insend', 'insends', 'indining', 'insisted']
         ['exprase', 'exprases', 'exprasing', 'exprased']
         ['associate', 'associates', 'associating', 'appointed']
--->     ['clamber', 'clambers', 'clambering', 'clambered']
         ['backforr', 'backforrs', 'backforring', 'backforred']
         ['cease', 'chastens', 'chastening', 'chastened']
         ['poose', 'poozes', 'poozing', 'poozed']
--->     ['despise', 'despises', 'despising', 'despised']
--->     ['approximate', 'approximates', 'approximating', 'approximated']
         ['reasit', 'exclaims', 'reasing', 'reasisted']
--->     ['obstruct', 'obstructs', 'obstructing', 'obstructed']
         ['endure', 'endears', 'enduring', 'endured']
         ['lasp', 'lashs', 'lasping', 'lashed']
         ['inlay', 'inlays', 'congealing', 'inlayed']
--->     ['entwine', 'entwines', 'entwining', 'entwined']
         ['leer', 'lends', 'learing', 'learned']
--->     ['preserve', 'preserves', 'preserving', 'preserved']
         ['misdeal', 'misdeals', 'misdealing', 'misdealed']
--->     ['regain', 'regains', 'regaining', 'regained']
         ['puldet', 'pulties', 'puldieting', 'pultied']
         ['clash', 'chings', 'chining', 'ching']
         ['install', 'instains', 'instaling', 'installed']
         ['spurr', 'spurses', 'spuring', 'spurred']
--->     ['interest', 'interests', 'interesting', 'interested']
         ['beheld', 'behelds', 'behaving', 'beheld']
         ['winci', 'winches', 'wincing', 'sived']
--->     ['identify', 'identifies', 'identifying', 'identified']
         ['crink', 'crinks', 'crowding', 'crowded']
         ['foretilize', 'foretilizes', 'foretilizing', 'foretilized']
--->     ['tremble', 'trembles', 'trembling', 'trembled']
         ['send', 'sends', 'sending', 'sensed']
         ['fight', 'fidgets', 'fidging', 'fidged']
         ['fedute', 'feduts', 'festuring', 'festured']
Gaussian score:  0.33
```

Figure 19: Gaussian score