Lab 3: Conditional Sequence-to-sequence VAE

Name:鄭謝廷揚 Student ID:A073501

Introduction

In this lab, we will implement Sequence-to-sequence VAE. With seq2seq VAE, we can transform word into different tense. In addition, we can generate a Gaussian noise vector and feed it with different tenses to the decoder and generate a word those tenses.

Implementation details

A. Model implement

1. Encoder implement

First, I convert condition one-hot to embedding vector and concatenate it with initial hidden input. Second, I convert inputs one-hot to embedding vector (use different embedding layer between condition). Then, I use nn.GRU function to implement LSTM and use *log variance* to generate *z*.

```python
class EncoderRNN(nn.Module):
    def __init__(
        self, word_size, hidden_size, latent_size,
        num_condition, condition_size
    ):
        super(EncoderRNN, self).__init__()
        self.word_size = word_size
        self.hidden_size = hidden_size
        self.condition_size = condition_size
        self.latent_size = latent_size

        self.condition_embedding = nn.Embedding(num_condition, condition_size)
        self.word_embedding = nn.Embedding(word_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)
        self.mean = nn.Linear(hidden_size, latent_size)
        self.logvar = nn.Linear(hidden_size, latent_size)

    def forward(self, inputs, init_hidden, input_condition):
        c = self.condition(input_condition)

        # shape = (1,1,hidden_size)
        hidden = torch.cat((init_hidden, c), dim=2)

        # shape = (seq, 1, hidden_size)
        x = self.word_embedding(inputs).view(-1, 1, self.hidden_size)

        # shape = (seq, 1, hidden_size), (1, 1, hidden_size)
        outputs, hidden = self.gru(x, hidden)

        # shape = (1, 1, hidden_size)
        m = self.mean(hidden)
        logvar = self.logvar(hidden)
        z = self.sample_z() * torch.exp(logvar/2) + m

        return z, m, logvar

    def initHidden(self):
        return torch.zeros(
            1, 1, self.hidden_size - self.condition_size,
            device=device
        )

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

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

2. Decoder implement

First, I use nn.Linear transform latent vector *z* and condition *c* to hidden size. Second, I input word-embedding *x* into nn.GRU to get the last output.

```
#Decoder
class DecoderRNN(nn.Module):
    def __init__(
        self, word_size, hidden_size, latent_size, condition_size
    ):
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.word_size = word_size

        self.latent_to_hidden = nn.Linear(latent_size+condition_size, hidden_size)
        self.word_embedding = nn.Embedding(word_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, word_size)

    def initHidden(self, z, c):
        latent = torch.cat((z, c), dim=2)
        return self.latent_to_hidden(latent)

    def forward(self, x, hidden):
        # shape = (1, 1, hidden_size)
        x = self.word_embedding(x).view(1, 1, self.hidden_size)

        # shape = (1, 1, hidden_size) (1, 1, hidden_size)
        output, hidden = self.gru(x, hidden)

        # shape = (1, word_size)
        output = self.out(output).view(-1, self.word_size)

        return output, hidden
```

3. Teacher forcing

In this function we input the ground truth into decoder model to get better training result.

```
def decode_teaching(decoder, z, c, maxlen, teacher=False, inputs=None):
    sos_token = train_dataset.chardict.word2index['SOS']
    eos_token = train_dataset.chardict.word2index['EOS']
    z = z.view(1,1,-1)
    i = 0

    outputs = []
    x = torch.LongTensor([sos_token]).to(device)
    hidden = decoder.initHidden(z, c)

    for i in range(maxlen):
        x = x.detach()
        output, hidden = decoder(x,hidden)
        outputs.append(output)
        output_onehot = torch.max(torch.softmax(output, dim=1), 1)[1]

        # meet EOS
        if output_onehot.item() == eos_token and not teacher:
            break

        if teacher:
            x = inputs[i+1:i+2]
        else:
            x = output_onehot

    if len(outputs) != 0:
        outputs = torch.cat(outputs, dim=0)
    else:
        outputs = torch.FloatTensor([]).view(0, word_size).to(device)

    return outputs
```

4. Text generation

Text generation is produced by Gaussian noise *Z*.

```
def decode_teaching(decoder, z, c, maxlen, teacher=False, inputs=None):
    sos_token = train_dataset.chardict.word2index['SOS']
    eos_token = train_dataset.chardict.word2index['EOS']
    z = z.view(1,1,-1)
    i = 0
```

And Z is defined as follow.

```
z = self.sample_z() * torch.exp(logvar/2) + m
```

5. Dataloding

```
class wordsDataset(Dataset):
    def __init__(self, train=True):
        if train:
            f = './train.txt'
        else:
            f = './test.txt'
        self.datas = np.genfromtxt(f,invalid_raise = False,dtype=np.str)
        print(self.datas)

        if train:
            self.datas = self.datas.reshape(-1)
        else:
            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 = [
            'simple-present',
            'third-person',
            'present-progressive',
            'simple-past'
        ]
        self.chardict = CharDict()

        self.train = train

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

    def __getitem__(self, index):
        if self.train:
            c = index % len(self.tenses)
            return self.chardict.longtensorFromString(self.datas[index]), c
        else:
            i = self.chardict.longtensorFromString(self.datas[index, 0])
            ci = self.targets[index, 0]
            o = self.chardict.longtensorFromString(self.datas[index, 1])
            co = self.targets[index, 1]
            return i, ci, o, co
```

B. Specify the hyperparameters

KLD weight and Teacher Forcing Ratio is decided by different epoch during the training process.

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

word_size = train_dataset.chardict.n_words
num_condition = len(train_dataset.tenses)
hidden_size = 256
latent_size = 32
condition_size = 8

teacher_forcing_ratio = 1.0
empty_input_ratio = 0.1
KLD_weight = 0.0
LR = 0.05

def KLD_weight_annealing_monotonic(epoch):
    slope = 0.001
    scope = (1.0 / slope)*2
    w = (epoch % scope) * slope
    if w > 1.0:
        w = 1.0
    return w
def KLD_weight_annealing_cyclical(epoch):
    if(epoch%10 == 0):
        return 0
    iterations = (epoch%10)*1000
    slope = 1/5000
    w = min(slope*iterations,1)
    return w
def Teacher_Forcing_Ratio(epoch):
    slope = 0.01
    level = 10
    w = 1.0 - (slope * (epoch//level))
    if w <= 0.0:
        w = 0.0

    return w
```

Results and discussion

A. Results of tense conversion and generation

1. Monotonic KL weight

- tense conversion

```
In [114]: evaluation(encoder, decoder, test_dataset)
```

```
abandon -> abandoned : abandoned
abet -> abetting : abasing
begin -> begins : begins
expend -> expends : owes
sent -> sends : sents
split -> splitting : splitting
flared -> flare : flare
functioning -> function : function
functioning -> functioned : functioned
healing -> heals : heals
BLEU-4 score : 0.7509077287947072
```

- generation

```
In [159]: words = []
for i in range(100):
    noise = encoder.sample_z()
    s = []
    for j in range(4):
        outputs = generate_word(encoder, decoder, noise, j)
        output_str = train_dataset.chardict.stringFromLongtensor(outputs)
        s.append(output_str)
    words.append(s)
print(words)
print(Gaussian_score(words))
```

```
[['embaaaa', 'embasains', 'embasaining', 'embasailed'], ['swit', 'swits', 'switcing', 'switcd'], ['regot', 'regots', 'regotting', 'regot'],
['burst', 'bursts', 'bursting', 'burst'], ['succe', 'succes', 'succsing', 'succeed'], ['bewail', 'bewails', 'bewailing', 'bewail'], ['wen',
'wents', 'wenting', 'went'], ['swing', 'singles', 'singling', 'singled'], ['festoon', 'reseots', 'festooning', 'festooned'], ['yell', 'yell
s', 'yelling', 'yeld'], ['surgi', 'survises', 'surving', 'surgived'], ['tell', 'tells', 'telling', 'telled'], ['swish', 'swishes', 'swishin
g', 'switched'], ['still', 'stiffs', 'stilling', 'stiched'], ['understand', 'undersses', 'undergoing', 'underseed'], ['truggle', 'tiggees',
'truggling', 'truggled'], ['wea', 'weads', 'wealing', 'wead'], ['sulk', 'sulks', 'sulking', 'sulked'], ['rade', 'rades', 'rading', 'rade'],
['whecy', 'whects', 'whesting', 'whelped'], ['seek', 'seeks', 'senking', 'swuwhed'], ['stot', 'stotes', 'streting', 'stote'], ['stitu', 'sti
tus', 'stituting', 'stituted'], ['stride', 'strides', 'striding', 'stride'], ['slum', 'slums', 'sluing', 'slung'], ['befi', 'befits', 'bendi
ng', 'bend'], ['brabe', 'brabs', 'brabing', 'brabed'], ['surgve', 'survives', 'surviving', 'survived'], ['foll', 'folld', 'follding', 'foll
d'], ['consun', 'consuets', 'consuing', 'consunted'], ['advise', 'arrives', 'advising', 'advised'], ['sweep', 'slashes', 'sweeping', 'swell
o'], ['upser', 'upsers', 'upsering', 'upsere'], ['soug', 'sough', 'soughing', 'sough'], ['drire', 'dripes', 'dripping', 'dripped'], ['preser
ve', 'preserves', 'preserving', 'preserve'], ['shriek', 'shrieks', 'shrieking', 'shrieked'], ['pret', 'peers', 'peering', 'peered'], ['summo
n', 'summons', 'summoning', 'summoned'], ['jonee', 'joneees', 'jingling', 'jengled'], ['sweep', 'sweeps', 'sweeping', 'swreok'], ['befit',
'befits', 'beginning', 'befied'], ['stalk', 'stalks', 'stalking', 'stalked'], ['beget', 'begets', 'begetn', 'beget'], ['steer', 'steers', 's
teering', 'steered'], ['withstand', 'withstands', 'withstanding', 'withstod'], ['yearn', 'yearns', 'yearning', 'yearned'], ['flank', 'flank
s', 'flanking', 'flanked'], ['achie', 'achies', 'aching', 'achied'], ['stalk', 'stalks', 'stalking', 'stalked'], ['spin', 'spins', 'spinnin
g', 'spin'], ['snor', 'snorts', 'snoring', 'snort'], ['stalk', 'stands', 'standing', 'stand'], ['stflat', 'stflects', 'stiffnnnig', 'stfalle
d'], ['protru', 'protrus', 'protruding', 'protruded'], ['chart', 'chartles', 'chartling', 'chartled'], ['scatter', 'scatisfies', 'scatterin
g', 'scauntd'], ['fashion', 'fashions', 'fashioning', 'fashioned'], ['wear', 'wears', 'wearing', 'weared'], ['switc', 'switches', 'switcin
g', 'assured'], ['gush', 'gushes', 'gushing', 'gushed'], ['subsort', 'subsorts', 'subsorting', 'snortred'], ['mestrod', 'mestures', 'mesturi
ng', 'murturned'], ['spray', 'sprays', 'spraying', 'sprawl'], ['sighte', 'sightees', 'sightening', 'sighered'], ['bankroll', 'bankrolls', 'b
ankrolling', 'bankroll'], ['lead', 'leads', 'leading', 'lend'], ['manifest', 'manifests', 'manifesting', 'manifested'], ['stimeme', 'stimemb
es', 'stimemeing', 'stimerged'], ['abast', 'abasts', 'abasting', 'availed'], ['stalk', 'talks', 'stalking', 'stalked'], ['crank', 'crawls',
'craining', 'crawled'], ['befeat', 'befeats', 'befeating', 'befeated'], ['stret', 'streihes', 'testsaying', 'testified'], ['swit', 'swits',
'switcing', 'switc'], ['sold', 'solds', 'solding', 'sold'], ['fatten', 'fattens', 'fattening', 'fattened'], ['permeal', 'permises', 'permisl
ing', 'permised'], ['stand', 'stands', 'satiaing', 'satiat'], ['stride', 'strides', 'striding', 'stride'], ['wugge', 'wuhsceens', 'wuggestin
g', 'wuggeer'], ['hurl', 'hurls', 'hurling', 'hurled'], ['persure', 'persures', 'persurring', 'persurbed'], ['get', 'gets', 'getting', 'gett
ed'], ['dush', 'buks', 'dushing', 'dushed'], ['see', 'seets', 'seeing', 'seet'], ['gainsay', 'gainsays', 'gainsaying', 'gainsayed'], ['wende
r', 'wenders', 'ensuering', 'weneered'], ['seer', 'seers', 'seering', 'seerap'], ['gainsay', 'gainsays', 'gainsaying', 'gainsayed'], ['swir
l', 'swirls', 'swirling', 'swirled'], ['arraign', 'arraigns', 'arraigning', 'arraigned'], ['tespend', 'tespends', 'tespending', 'tespette
d'], ['presert', 'presends', 'presending', 'presended'], ['stalk', 'stalks', 'stalking', 'stalked'], ['wrank', 'creaks', 'creaking', 'wranke
d'], ['perceive', 'perserves', 'perserving', 'perceived'], ['knit', 'kniists', 'knitcing', 'kniipat'], ['suffer', 'suffers', 'suffering', 's
uffered'], ['saw', 'saws', 'sawing', 'saw']]
0.21
```

2. Cyclical KL weight

- tense conversion

```
In [194]: evaluation(encoder, decoder, test_dataset)
```

```
abandon -> abandoned : boerered
abet -> abetting : boerering
begin -> begins : boessws
expend -> expends : boessws
sent -> sends : boeh
split -> splitting : borhceigg
flared -> flare : boerer
functioning -> function : boerer
functioning -> functioned : boerere
healing -> heals : boessws
BLEU-4 score : 0.05507959219588354
```

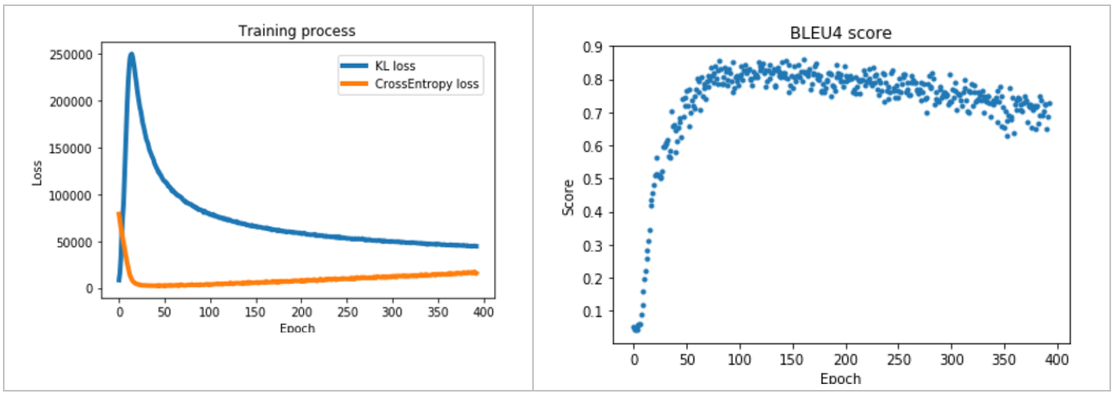
- generation

```
In [195]: words = []
for i in range(100):
    noise = encoder.sample_z()
    s = []
    for j in range(4):
        outputs = generate_word(encoder, decoder, noise, j)
        output_str = train_dataset.chardict.stringFromLongtensor(outputs)
        s.append(output_str)
    words.append(s)
print(words)
print(Gaussian_score(words))
```

```
[['boerer', 'boessws', 'borhceigg', 'boerered'], ['boerer', 'boessws', 'borhceigg', 'boerered'], ['boerer', 'boerses', 'borhceigg', 'boerere
d'], ['boerer', 'boessws', 'borhceigg', 'boerered'], ['boerer', 'boeh', 'borhceigg', 'boerere'], ['boerer', 'boeh', 'borhceigg', 'boerere
d'], ['boerer', 'boessws', 'borhceigg', 'boerered'], ['boerer', 'boessws', 'borhceigg', 'boerered'], ['boeh', 'boeh', 'boerering', 'boerere
d'], ['boerer', 'boessws', 'borhceigg', 'boerered'], ['boeh', 'boessws', 'borhceigg', 'boerered'], ['boeh', 'boeh', 'borhceigg', 'boerere
d'], ['boeh', 'boessws', 'borhceigg', 'boerered'], ['boerer', 'boeh', 'borhceigg', 'boerere'], ['boerer', 'boerses', 'borhceigg', 'boerer
e'], ['boeh', 'boeh', 'borhceigg', 'boerered'], ['boerer', 'boessws', 'borhceigg', 'boerered'], ['boerer', 'boeh', 'borhceigg', 'boerered'],
['boerer', 'boerers', 'boerering', 'boerere'], ['boerer', 'boerers', 'borhcegng', 'boerered'], ['boeh', 'boeh', 'borhceigg', 'boerere'], ['b
oerer', 'boeh', 'borhceigg', 'boerere'], ['boerer', 'boehces', 'borhceigg', 'boerere'], ['boeh', 'boessws', 'boerering', 'boerered'], ['boer
er', 'boerers', 'boerering', 'boerere'], ['boeh', 'boeh', 'boerering', 'boerered'], ['boerer', 'boessws', 'borhceigg', 'boerered'], ['boere
r', 'boeh', 'borhceigg', 'boerere'], ['boerer', 'boerers', 'borhceigg', 'boerered'], ['boeh', 'boeh', 'borhceigg', 'boerered'], ['boerer',
'boessws', 'boerering', 'boerered'], ['boerer', 'boessws', 'borhceigg', 'boerered'], ['boerer', 'boerses', 'borhceigg', 'boerered'], ['boe
h', 'boeh', 'borhceigg', 'boerere'], ['boerer', 'boessws', 'borhcegng', 'boerered'], ['boerer', 'boehces', 'borhceigg', 'boerered'], ['boe
h', 'boeh', 'borhcegng', 'boerered'], ['boeh', 'boessws', 'borhceigg', 'boerered'], ['boeh', 'boeh', 'borhceigg', 'boerere'], ['boeh', 'boe
h', 'boerering', 'boerered'], ['boerer', 'boehces', 'borhceigg', 'boerered'], ['boerer', 'boessws', 'borhceigg', 'boerered'], ['boerer', 'bo
essws', 'borhceigg', 'boerered'], ['boerer', 'boerers', 'borhceigg', 'boerere'], ['boerer', 'boerses', 'borhceigg', 'boerered'], ['boerer',
'boerers', 'boerering', 'boerere'], ['boerer', 'boessws', 'boerering', 'boerered'], ['boerer', 'boerers', 'boerering', 'boerere'], ['boere
r', 'boeh', 'boerering', 'boerered'], ['boerer', 'boessws', 'boerering', 'boerered'], ['boerer', 'boessws', 'boerering', 'boerered'], ['boer
er', 'boerers', 'borhcegng', 'boerered'], ['boeh', 'boeh', 'borhceigg', 'boerered'], ['boerer', 'boessws', 'boerering', 'boerered'], ['boe
h', 'boeh', 'borhceigg', 'boerere'], ['boeh', 'boessws', 'boerering', 'boerered'], ['boeh', 'boessws', 'borhceigg', 'boerered'], ['boerer',
'boerers', 'borhceigg', 'boerered'], ['boeh', 'boessws', 'borhceigg', 'boerered'], ['boerer', 'boessws', 'borre', 'boerere'], ['boeh', 'boe
h', 'boeiong', 'boerere'], ['boerer', 'boessws', 'borhceigg', 'boerered'], ['boerer', 'boessws', 'borhceigg', 'boerered'], ['boerer', 'boess
ws', 'borhceigg', 'boerered'], ['boeh', 'boessws', 'borhcegng', 'boerered'], ['boerer', 'boeh', 'borhceigg', 'boerered'], ['boeh', 'boeh',
'borhceigg', 'boerered'], ['boeh', 'boehces', 'boerering', 'boerered'], ['boerer', 'boessws', 'borhceigg', 'boerered'], ['boeh', 'boeh', 'bo
rhceigg', 'boerered'], ['boeh', 'boehces', 'borhceigg', 'boerere'], ['boerer', 'boessws', 'borhceigg', 'boerered'], ['boeh', 'boessws', 'bor
hceigg', 'boerered'], ['boeh', 'boeh', 'boerering', 'boerered'], ['boerer', 'boeh', 'borhceigg', 'boerere'], ['boeh', 'boeh', 'borhceigg',
'boerered'], ['boerer', 'boerers', 'boerering', 'boerered'], ['boeh', 'boeh', 'boeiong', 'boerered'], ['boerer', 'boessws', 'borhceigg', 'bo
erered'], ['boerer', 'boerses', 'boeiongg', 'boerere'], ['boerer', 'boerers', 'boerering', 'boerered'], ['boerer', 'boerers', 'borhceigg',
'boerered'], ['boeh', 'boeh', 'borhceigg', 'boerered'], ['boeh', 'boeh', 'boerering', 'boerered'], ['boerer', 'boessws', 'borhceigg', 'boere
red'], ['boerer', 'boessws', 'borre', 'boerered'], ['boeh', 'boeh', 'boeiong', 'boerere'], ['boerer', 'boehces', 'borhceigg', 'boerere'],
['boerer', 'boeh', 'borhceigg', 'boerered'], ['boeh', 'boeh', 'borhceigg', 'boerere'], ['boerer', 'boeh', 'borhcegng', 'boerere'], ['boeh',
'boeh', 'borhcegng', 'boerered'], ['boerer', 'boessws', 'borhceigg', 'boerered'], ['boerer', 'boerers', 'borhceigg', 'boerered'], ['boerer',
'boehces', 'borhcegng', 'boerere'], ['boeh', 'boess', 'borhceigg', 'boerered'], ['boerer', 'boessws', 'boeiongg', 'boerered'], ['boeh', 'boe
h', 'borhceigg', 'boerere'], ['boerer', 'boessws', 'boerering', 'boerered'], ['boeh', 'boeh', 'boeiongg', 'boerere']]
0.0
```

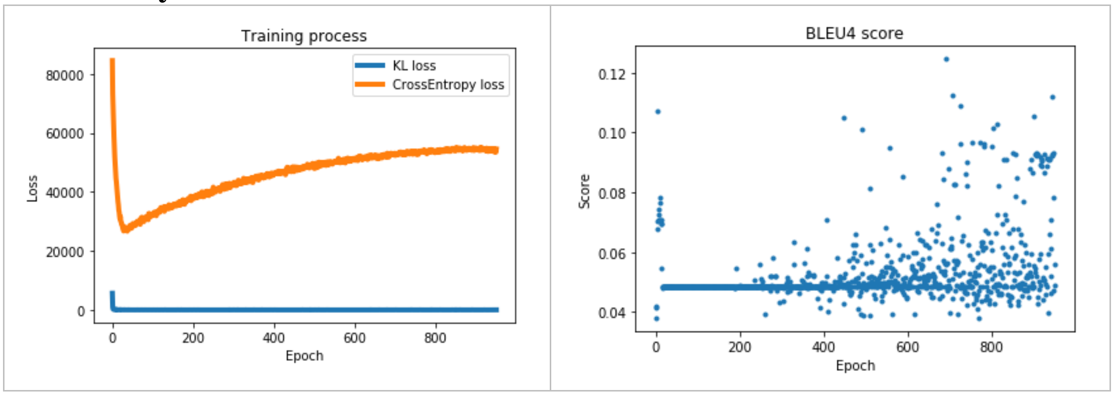
Detail result discussing

1. Monotonic KL result



The low KLD weight cause low cross entropy loss and high KLD loss. After KLD weight rising, cross entropy became higher and KLD loss became lower.

2. Cyclical KL result



With Cyclical KL weight we can not train model well. I think I have two approach to improve the model.

i. Lengthen the period of cyclical KL.

ii. Change teacher forcing ratio by cyclical KL