Deep Learning and Practice Lab 4

309554005 黃睿宇

1 Introduction

VAE has been applied to many NLP generation tasks such as text summarization and paraphrase. In this lab, we implement a conditional seq2seq VAE for English tense conversion and text generation. For example, when we input the input word "access" with the tense (the condition) "simple present" to the encoder, it will generate a latent vector z. Then, we take z with the tense "present progressive" as the input for the decoder and we expect that the output word should be "accessing". In addition, we can also manually generate a Gaussian noise vector and feed it with different tenses to the decoder and generate a word those tenses.

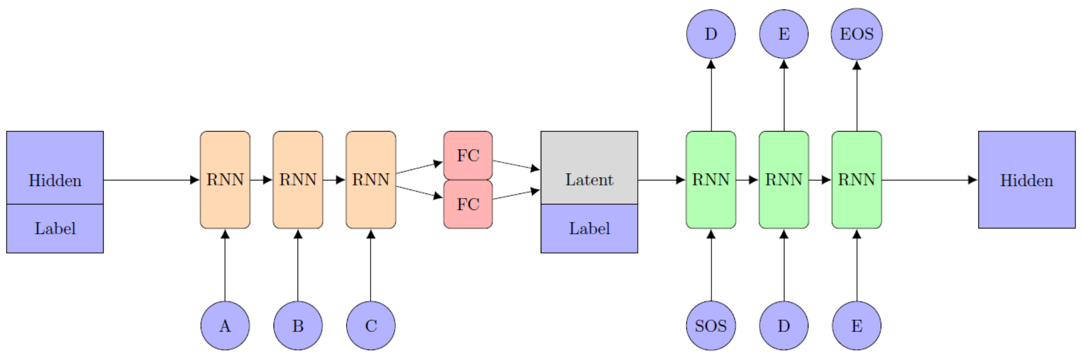


Figure 1: The illustration of sequence-to-sequence VAE architecture.

In training dataset, there are 1227 training pairs. Each training pair includes 4 words: simple present(sp), third person(tp), present progressive(pg), simple past(p). In validation dataset, there are 10 validating pairs. Each training pair includes 2 words with different combination of tenses.

2 Derivation of Conditional VAE

To see how the EM works, the chain rule of probability suggests
$\log p(\boldsymbol{X};\boldsymbol{\theta}) = \log p(\boldsymbol{X},\boldsymbol{Z};\boldsymbol{\theta}) - \log p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})$.

We next introduce an arbitrary distribution $q(\boldsymbol{Z})$ on both sides and integrate over $\boldsymbol{Z}$.

$$\int q(\boldsymbol{Z}) \log p(\boldsymbol{X};\boldsymbol{\theta})\, d\boldsymbol{Z} = \int q(\boldsymbol{Z}) \log p(\boldsymbol{X},\boldsymbol{Z};\boldsymbol{\theta})\, d\boldsymbol{Z} - \int q(\boldsymbol{Z}) \log p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})\, d\boldsymbol{Z}$$

$$= \int q(\boldsymbol{Z}) \log p(\boldsymbol{X},\boldsymbol{Z};\boldsymbol{\theta})\, d\boldsymbol{Z} - \int q(\boldsymbol{Z}) \log q(\boldsymbol{Z})\, d\boldsymbol{Z}$$

$$+\int q(\boldsymbol{Z}) \log q(\boldsymbol{Z})\, d\boldsymbol{Z} - \int q(\boldsymbol{Z}) \log p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})\, d\boldsymbol{Z}$$

to arrive at

$\log p(\boldsymbol{X};\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{X}, q, \boldsymbol{\theta}) + \mathrm{KL}\big(q(\boldsymbol{Z}) \parallel p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})\big)$

where

$$\mathcal{L}(\boldsymbol{X}, q, \boldsymbol{\theta}) = \int q(\boldsymbol{Z}) \log p(\boldsymbol{X},\boldsymbol{Z};\boldsymbol{\theta})\, d\boldsymbol{Z} - \int q(\boldsymbol{Z}) \log q(\boldsymbol{Z})\, d\boldsymbol{Z}$$

$$\mathrm{KL}\big(q(\boldsymbol{Z}) \parallel p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})\big) = \int q(\boldsymbol{Z}) \log \frac{q(\boldsymbol{Z})}{p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})}\, d\boldsymbol{Z}$$

Since the KL divergence is non-negative, $\mathrm{KL}(q \parallel p) \geq 0$, it follows that $\log p(\boldsymbol{X};\boldsymbol{\theta}) \geq \mathcal{L}(\boldsymbol{X}, q, \boldsymbol{\theta})$ with equality if and only if $q(\boldsymbol{Z}) = p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})$. In other words, $\mathcal{L}(\boldsymbol{X}, q, \boldsymbol{\theta})$ is a lower bound on $\log p(\boldsymbol{X};\boldsymbol{\theta})$.

A rearrangement gives

$\log p(\boldsymbol{X};\boldsymbol{\theta}) - \mathrm{KL}\big(q(\boldsymbol{Z}) \parallel p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})\big) = \mathcal{L}(\boldsymbol{X}, q, \boldsymbol{\theta})$.

As the equality holds for any choice of $q(\boldsymbol{Z})$, we introduce a distribution $q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')$ modeled by another neural network with parameter $\boldsymbol{\theta}'$ to obtain

$\log p(\boldsymbol{X};\boldsymbol{\theta}) - \mathrm{KL}\big(q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}') \parallel p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})\big) = \mathcal{L}(\boldsymbol{X}, q, \boldsymbol{\theta})$.

The right hand side can be spell out as

$$\begin{aligned}\mathcal{L}(\boldsymbol{X}, q, \boldsymbol{\theta}) &= E_{Z\sim q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')} \log p(\boldsymbol{X}|\boldsymbol{Z};\boldsymbol{\theta}) + E_{Z\sim q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')} \log p(\boldsymbol{Z}) \\ &\quad - E_{Z\sim q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')} \log q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}) \\ &= E_{Z\sim q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')} \log p(\boldsymbol{X}|\boldsymbol{Z};\boldsymbol{\theta}) - \mathrm{KL}\big(q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}') \parallel p(\boldsymbol{Z})\big)\end{aligned}$$

Now, instead of directly maximizing the intractable $p(\boldsymbol{X};\boldsymbol{\theta})$, we attempt to maximize

$\log p(\boldsymbol{X};\boldsymbol{\theta}) - \mathrm{KL}\big(q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}') \parallel p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})\big)$

which amounts to maximizing

$$E_{Z\sim q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')} \log p(\boldsymbol{X}|\boldsymbol{Z};\boldsymbol{\theta}) - \mathrm{KL}\big(q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}') \parallel p(\boldsymbol{Z})\big).$$

To make the KL divergence tractable, both $q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')$ and $p(\boldsymbol{Z})$ are assumed to be Gaussians.

Following the same line of derivations as for the unconditional case, the variational lower bound of $\log p(\boldsymbol{X}|c)$ for conditional VAE is given by

$$E_{Z\sim q(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta}')} \log p(\boldsymbol{X}|\boldsymbol{Z},c;\boldsymbol{\theta}) - \mathrm{KL}\big(q(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta}') \parallel p(\boldsymbol{Z}|c)\big).$$

3 Derivation of KL Divergence Loss

Given the prior $p(z) \sim N(0, I)$ and the posterior approximation $q(z|x;\theta) \sim N(\mu_\theta(x),\ \Sigma_\theta(x))$.

$$\because E[(x-\mu)^2] = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} (x-\mu)^2\, dx$$

$$= \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}} y^2\, dy \quad (\text{let } y = x-\mu)$$

$$= \frac{1}{\sqrt{2\pi}\sigma} y \left(-\sigma^2 e^{-\frac{y^2}{2\sigma^2}}\right)\Big|_{-\infty}^{\infty} - \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma}\left(-\sigma^2 e^{-\frac{y^2}{2\sigma^2}}\right) dy$$

$$(\text{let } u = y,\ dv = e^{-\frac{y^2}{2\sigma^2}} y dy \ \Rightarrow\ du = dy,\ v = -\sigma^2 e^{-\frac{y^2}{2\sigma^2}})$$

$$= \frac{-\sigma^2}{\sqrt{2\pi}\sigma} \frac{y}{e^{\frac{y^2}{2\sigma^2}}}\Big|_{-\infty}^{\infty} + \sigma^2 \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}}\, dy$$

$$= \frac{-\sigma}{\sqrt{2\pi}} \frac{1}{e^{\frac{y^2}{2\sigma^2}} \frac{y^2}{\sigma^2}}\Big|_{-\infty}^{\infty} + \sigma^2 \cdot 1$$

$$= 0 + \sigma^2$$

$$= \sigma^2$$

$$\therefore \mathrm{KL}\big(q(z|x;\theta) \parallel p(z)\big) = \mathrm{KL}\big(N(\mu, \sigma^2) \parallel\ N(0,1)\big)$$

$$= \int \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \left(\log \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}} / \sqrt{2\pi}\sigma}{e^{-\frac{x^2}{2}} / \sqrt{2\pi}}\right) dx$$

$$= \int \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \left(\log \frac{1}{\sigma} e^{\frac{1}{2}\left(x^2 - \frac{(x-\mu)^2}{\sigma^2}\right)}\right) dx$$

$$= \int \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \frac{1}{2}\left(-\log \sigma^2 + x^2 - \frac{(x-\mu)^2}{\sigma^2}\right) dx$$

$$= E\left[\frac{1}{2}\left(-\log \sigma^2 + x^2 - \frac{(x-\mu)^2}{\sigma^2}\right)\right]$$

$$= \frac{1}{2}\left(E[-\log \sigma^2] + E[x^2] - E\left[\frac{(x-\mu)^2}{\sigma^2}\right]\right)$$

$$= \frac{1}{2}\left(-\log \sigma^2 + (\sigma^2 + \mu^2) - \frac{E[(x-\mu)^2]}{\sigma^2}\right)$$

$$= \frac{1}{2}(-\log \sigma^2 + \sigma^2 + \mu^2 - 1)$$

is tractable

4 Implementation Details

4.1 Dataloader

To feed character into model, we added 26 English characters and 2 special token (<SOS> and <EOS>) into `Dictionary`. Then we could use `torch.nn.embedding` to deal with it.

```python
class DataTransformer:
    def __init__(self):
        self.char2idx = self.build_char2idx()
        self.idx2char = self.build_idx2char()
        self.tense2idx = {'sp': 0, 'tp': 1, 'pg': 2, 'p': 3}
        self.idx2tense = {0: 'sp', 1: 'tp', 2: 'pg', 3: 'p'}
        self.max_length = 0

    def build_char2idx(self):
        dictionary = {'SOS': 0, 'EOS': 1}
        dictionary.update([(chr(i+97), i+2) for i in range(0, 26)])
        return dictionary

    def build_idx2char(self):
        dictionary = {0: 'SOS', 1: 'EOS'}
        dictionary.update([(i+2, chr(i+97)) for i in range(0, 26)])
        return dictionary
```

Since training data and testing data are in different format, we returned one word and tense in training phase while returned two words and two tenses in testing phase.

```python
def __getitem__(self, idx):
    """
    if(train): word: (time1,1) tensor, tense: (1) tensor
    if(test): word1: (time1,1) tensor, tense1: (1) tensor,
              word2: (time2,1) tensor, tense2: (1) tensor
    """
    if self.is_train:
        return self.string2tensor(self.words[idx], add_eos=True),
               self.tense2tensor(self.tenses[idx])
    else:
        return self.string2tensor(self.words[idx][0], add_eos=True),
               self.tense2tensor(self.tenses[idx][0]),\
               self.string2tensor(self.words[idx][1], add_eos=True),
               self.tense2tensor(self.tenses[idx][1])
```

4.2 Encoder

In the begin, we convert the condition into one-hot vector and concatenate the condition part with the initial hidden part as input of encoder. Before the concatenation, we construct condition embeddings via projection. After embedding, we run LSTM model.

```
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.rnn = nn.LSTM(hidden_size, hidden_size)

    def forward(self, input, hidden_state, cell_state):
        embedded = self.embedding(input).view(1, 1, -1)
        output, (hidden_state, cell_state) = self.rnn(
            embedded, (hidden_state, cell_state))
        return output, hidden_state, cell_state
```

4.3 Reparameterization trick

We use reparameterization trick to solve the problem since we couldn't calculate gradient of parameters in encoder. First, we sample a z^* from multivariate normal distribution ~ $N(0, 1)$. Then we multiply z^* by $exp(logvar/2)$ and add mean. Since we only multiply and add a constant, we could calculate gradient of parameters in encoder directly.

```
def reparameterize(self, mean, logvar):
    std = torch.exp(0.5*logvar)
    eps = torch.randn_like(std)
    latent = mean + eps * std
    return latent
```

```python
# middle part forwarding
mean = self.hidden2mean(encoder_hidden_state)
logvar = self.hidden2logvar(encoder_hidden_state)
# sampling a point
latent = self.reparameterize(mean, logvar)
decoder_hidden_state = self.latentcondition2hidden(
    torch.cat((latent, c), dim=-1))
decoder_cell_state = self.decoder.init_c0()
decoder_input = torch.tensor([[SOS_token]], device=device)
```

4.4 Decoder

Decoder is similar to encoder. We concatenate the condition part with the latent vector *z* as input of decoder, then we do LSTM which is the same as in encoder.

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

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

4.5 Text generated by Gaussian noise

We use `torch.randn` to generate a latent tensor randomly. Than we concatenate latent tensor with tense tensor and make it be the hidden state of decoder.

```python
def generateWord(vae, latent_size, tensor2string):
    vae.eval()
    re = []
    with torch.no_grad():
        for i in range(100):
            latent = torch.randn(1, 1, latent_size).to(device)
            tmp = []
            for tense in range(4):
                word = tensor2string(vae.generate(latent, tense))
                tmp.append(word)

            re.append(tmp)
    return re
```

```python
def generate(self, latent, tense):
     tense_tensor = torch.tensor([tense]).to(device)
     c = self.tense_embedding(tense_tensor).view(1, 1, -1)
     decoder_hidden_state = self.latentcondition2hidden(
          torch.cat((latent, c), dim=-1))
     decoder_cell_state = self.decoder.init_c0()
     decoder_input = torch.tensor([[SOS_token]], device=device)

     # decoder forwarding
     predict_output = None
     for di in range(self.max_length):
          output, decoder_hidden_state, decoder_cell_state = self.decoder(
               decoder_input, decoder_hidden_state, decoder_cell_state)
          predict_class = output.topk(1)[1]
          predict_output = torch.cat((predict_output, predict_class)) if
               predict_output is not None else predict_class

          if predict_class.item() == EOS_token:
               break
          decoder_input = predict_class

     return predict_output
```

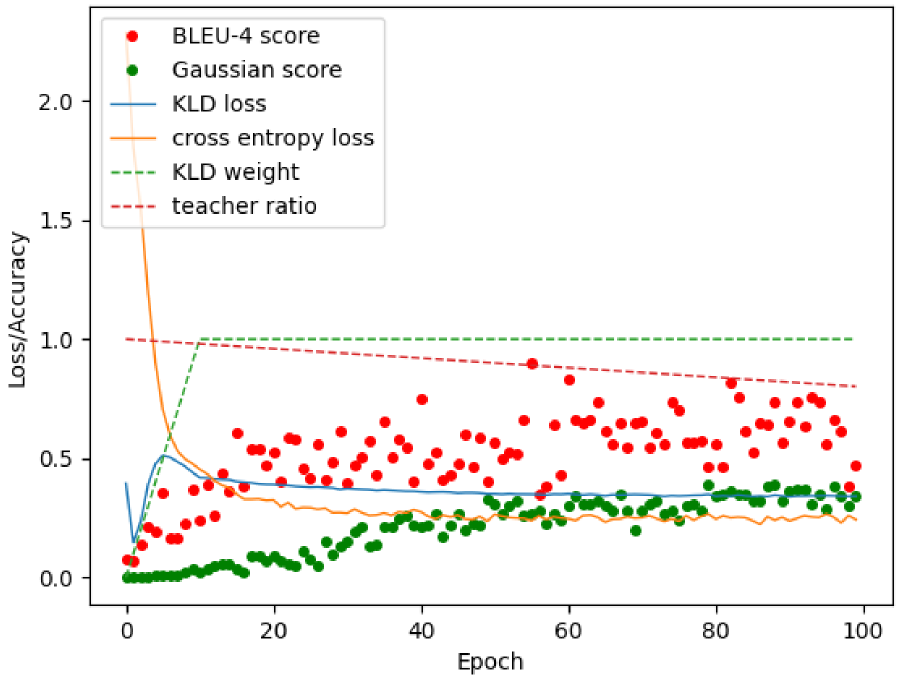
4.6 Hyperparameters

- Optimizer: SGD
- Loss function: cross entropy
- Epochs = 100
- RNN hidden size = 256
- Latent size = 32
- Condition embedding size = 8
- KL weight = 0
- Teacher forcing ratio = 1
- Learning rate = 0.05

5 Results and Discussion

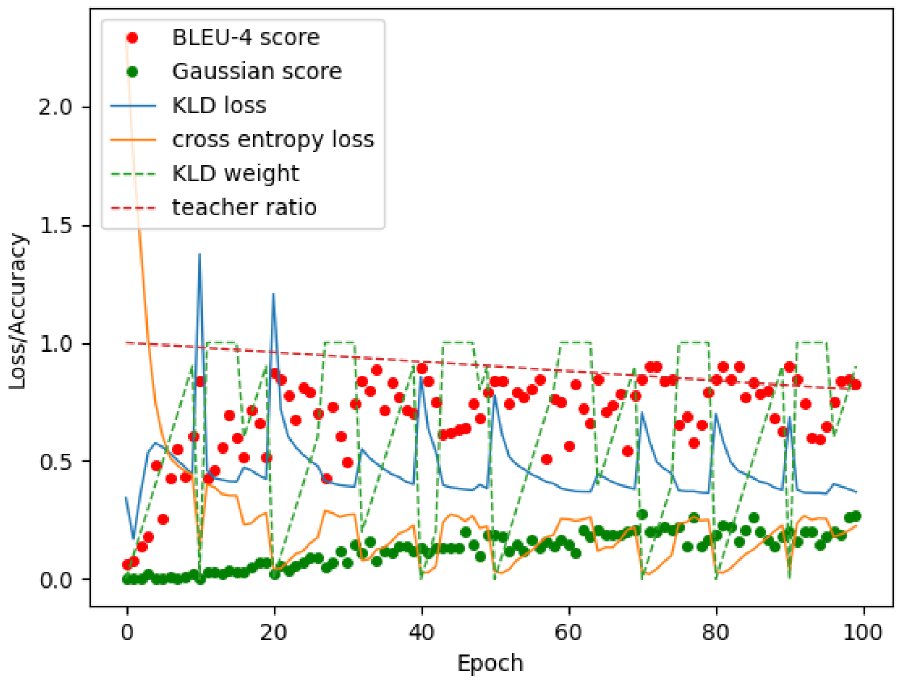
5.1 Training phase

- Monotonic



In the begin, since KL weight is low, cross entropy loss is low while KL loss is high. At about the 10th epoch, rising of KL weight leads to declining in KL loss. As the epoch increases, cross entropy decreases which means the word construction is successful, and thus BLEU-4 score raises.

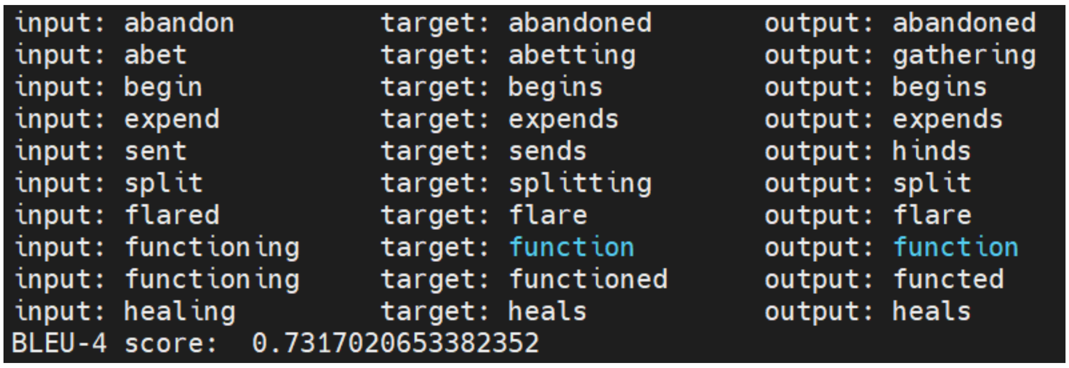
- Cyclical



In the result we could see that when KL weight rises, cross entropy loss rises as well while KL loss decreases. However, when cross entropy increases, BLEU-4 score falls. The overall score seems to be better than the monotonic.

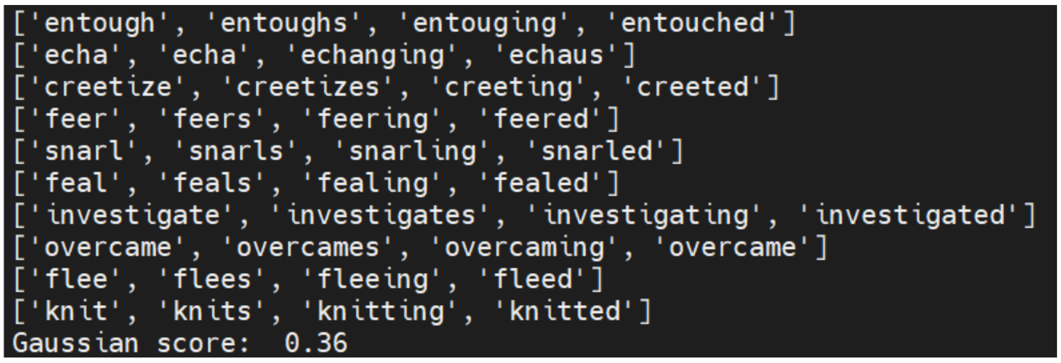
5.2 Testing phase

- English tense conversion with BLEU-4 score



```
input: abandon          target: abandoned        output: abandoned
input: abet             target: abetting         output: gathering
input: begin            target: begins           output: begins
input: expend           target: expends          output: expends
input: sent             target: sends            output: hinds
input: split            target: splitting        output: split
input: flared           target: flare            output: flare
input: functioning      target: function         output: function
input: functioning      target: functioned       output: functed
input: healing          target: heals            output: heals
BLEU-4 score:  0.7317020653382352
```

- Gaussian noise with 4 tenses with Gaussian score



```
['entough', 'entoughs', 'entouging', 'entouched']
['echa', 'echa', 'echanging', 'echaus']
['creetize', 'creetizes', 'creeting', 'creeted']
['feer', 'feers', 'feering', 'feered']
['snarl', 'snarls', 'snarling', 'snarled']
['feal', 'feals', 'fealing', 'fealed']
['investigate', 'investigates', 'investigating', 'investigated']
['overcame', 'overcames', 'overcaming', 'overcame']
['flee', 'flees', 'fleeing', 'fleed']
['knit', 'knits', 'knitting', 'knitted']
Gaussian score:  0.36
```