

Lab 5: Conditional Sequence-to-sequence VAE

● Lab objective

In this lab, you need to implement a conditional seq2seq VAE for English tense conversion.

● Important Date

1. Deadline: 5/9 (Thu) 11:59 a.m.
2. Demo date: 5/9 (Thu)

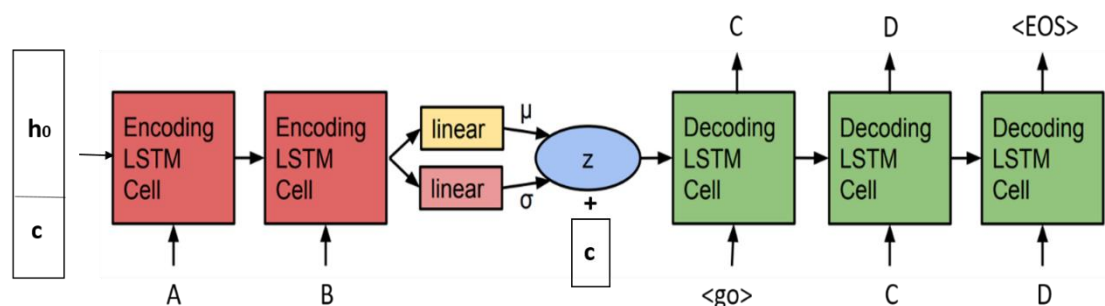
● Turn in

1. Experimental Report (.pdf)
2. Source code

Notice: zip all files in one file and name it like **DLP_LAB5_your studentID_name.zip**. e.g. DLP_LAB5_0756051_李仕柏.zip

● Lab Description

When we feed the input word ‘access’ with the tense (the condition) ‘simple present’ to the encoder, it will generate a latent vector z . Then, we feed z with the tense ‘present progressive’ to 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. The figure blow is the overall conditional seq2seq VAE model architecture modified from [Samuel R. Bowman et al. 2016].

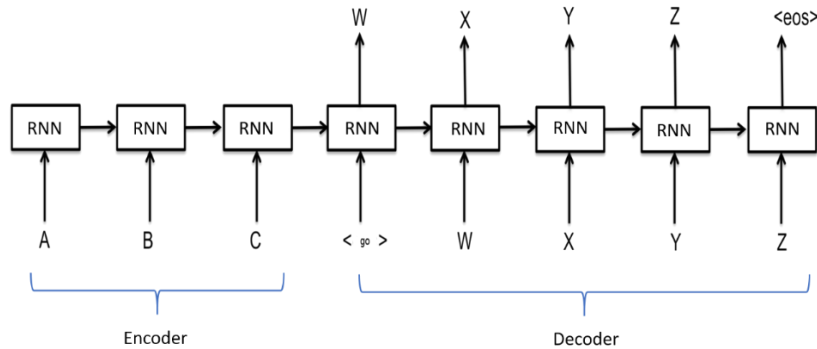


● Requirements

1. Implement a conditional seq2seq VAE.
 - A. Modify encoder, decoder, and training functions
 - B. Implement evaluation function, dataloader, and reparameterization trick.
2. Adopt teacher-forcing and kl loss annealing in your training processing.
3. Plot the loss and KL loss curve during training.
4. Plot the BLEU-4 score curve of the testing data during training.
5. Output the conversion results between tenses (from tense A to tense B)
6. Output the results generated by a Gaussian noise with 4 tenses.

● Implementation details

1. Seq2seq architecture



Each character in a word can be regarded as a vector. One simple approach to convert a character to a vector is encoding a character to a number and adopting Embedding Layer (see more information in [1]). In the decoder part, you will first feed the hidden output from the encoder and a <go> or <start of string> token to the decoder and stop generation process until you receive a <end of string> token or reach the last token of the target word (the token should also be <end of string>).

2. VAE

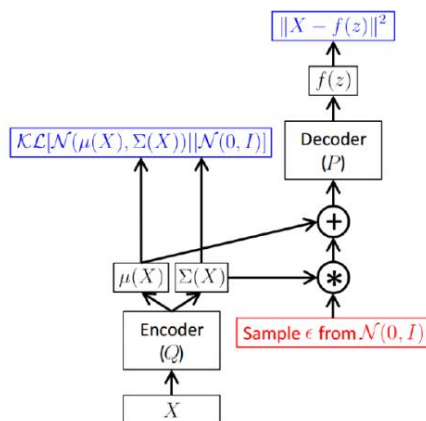
A. Recall that the loss function of VAE:

$$\mathcal{L}(X, q, \theta) = E_{Z \sim q}(Z|X; \phi) \log p(X|Z; \theta) - KL(q(Z|X; \phi) || p(Z))$$

where $q(Z|X; \phi)$ is considered as encoder and $p(X|Z; \theta)$ as decoder.

B. Reparameterization trick:

Train the encoder and decoder jointly.



$$\underbrace{E_{Z \sim q}(Z|X; \theta')}_{\text{Re-parameterization for end-to-end training}} - KL(q(Z|X; \theta') || p(Z))$$

C. Log variance:

The output of reparameterization trick should be **log variance** instead of variance directly. (see more information in [Diederik P. Kingma et al. 2014])

D. Conditional VAE:

$$E_{\mathbf{Z} \sim q(\mathbf{Z}|\mathbf{X}, c; \theta')} \log p(\mathbf{X}|\mathbf{Z}, c; \theta) - \text{KL}(q(\mathbf{Z}|\mathbf{X}, c; \theta') || p(\mathbf{Z}|c))$$

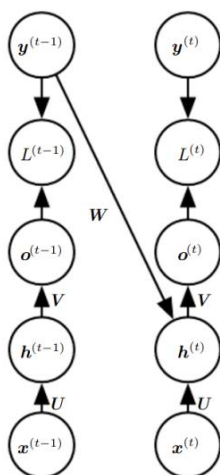
Both the encoder $q(\mathbf{Z}|\mathbf{X}, c; \phi)$ and the decoder $p(\mathbf{X}|\mathbf{Z}, c; \theta)$ need to take c as part of their input. There are several ways to add the conditional part to your VAE model. In the figure of model architecture, we concatenate the condition part with the initial hidden part as input of encoder. Similarly, we concatenate the condition part with the latent vector \mathbf{z} as input of decoder. Before the concatenation, we construct **condition embeddings** via projection. You can adopt `nn.Embedding` and decide the size of your condition embeddings. You can also try to convert your condition into one-hot vector.

E. KL cost annealing:

This is a simple approach adopted by [Samuel R. Bowman et al. 2016]. We add a variable weight to the KL term in the loss function. We initially set the weight to 0. The maximum value is 1. You can adjust the weight **in any way** you like while training (e.g. linear or non-linear.).

3. Teacher forcing

In the course, we have talked about teacher forcing technique, which feeds the correct target $y^{(t-1)}$ into $h^{(t)}$ during training. Thus, in this part, you will need to implement teacher forcing technique. Furthermore, according to [Samuel R. Bowman et al. 2016], we can do the **word dropout** to weaken the decoder by randomly replacing the input character tokens with the unknown token (defined by yourself). This forces the model only relying on the latent vector \mathbf{z} to make predictions.



4. Other implementation details

- ◆ The encoder and decoder are implemented by LSTM or GRU.
- ◆ The loss function is `nn.CrossEntropyLoss()`.
- ◆ Adopt BLEU-4 score function in NLTK [4]. Here is the reference code for BLEU-4.

```
from nltk.translate.bleu_score import SmoothingFunction, sentence_bleu
reference = [['this', 'is', 'small', 'test']]
candidate = ['this', 'is', 'a', 'test']
cc = SmoothingFunction()
sentence_bleu(reference, candidate, weights=(0.25, 0.25, 0.25, 0.25), smoothing_function=cc.method1)
```

● Dataset Descriptions

You can download the .zip file from new e3. There are three files in the .zip: readme.txt, train.txt, and test.txt. All the details of the dataset are in the readme.txt.

● Scoring Criteria

1. Report (60%)

- ◆ Introduction
- ◆ Implementation details
 - A. Describe how you implement your model. (e.g. dataloader, encoder, decoder, reparameterization trick, hidden size, condition embedding size, word dropout, rnn type, etc.)
 - B. Specify the hyperparameters (KL weight, learning rate, teacher forcing ratio, epochs, etc.)
- ◆ Results and discussion
 - A. Plot the loss and KL loss curve during training and discuss the results according to your setting of teacher forcing ratio, KL weight, and learning rate.
 - B. Plot the BLEU-4 score of your testing data while training and discuss the result.

2. Demo (40%)

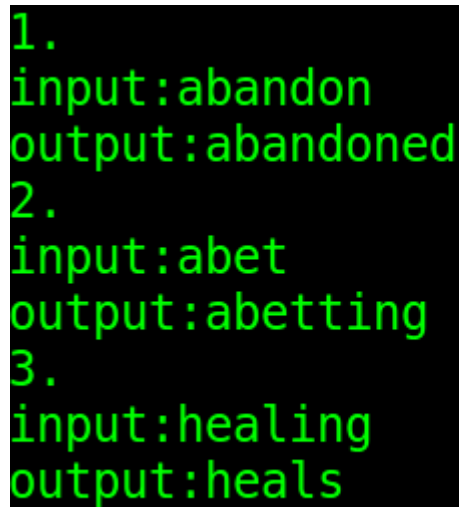
- A. Capability of tense conversion on testing data. (10%)
 - Score ≥ 0.7 ---- 100%
 - $0.7 > \text{score} \geq 0.6$ ---- 90%
 - $0.6 > \text{score} \geq 0.5$ ---- 80%
 - Score < 0.5 ---- 70%
- B. Capability of word generation. (Gaussian noise + tense) (10%)

You have 3 chances to randomly generate different Gaussian noises with 4 tenses. You can choose one of them as your final score.

 - 4 correct words ---- 100%
 - 3 correct words ---- 90%
 - 2 correct words ---- 80%
 - Otherwise ---- 0%
- C. Questions (20%)

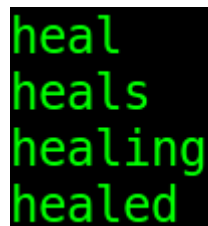
- **Output examples**

1. English tense conversion (test.txt)



```
1.  
input:abandon  
output:abandoned  
2.  
input:abet  
output:abetting  
3.  
input:healing  
output:heals
```

2. Gaussian noise with 4 tenses



```
heal  
heals  
healing  
healed
```

- **Hints**

1. While training, your input and output words should have the same tense. For example, if your input is 'accessing'+ 'progress', then your output should also be 'accessing'+ 'progress'.
2. Sequence-to-sequence model is very sensitive to the previous hidden input and hence, I **strongly** suggest you save your model weights after each epoch so that you can decide which weight you want to use.
3. The teacher forcing ratio and KL weight are very important for training this model and **significantly influence** the performance.

- **Reference**

1. Seq2seq reference code:
https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html
https://github.com/pytorch/tutorials/blob/master/intermediate_source/seq2seq_translation_tutorial.py
2. Generating Sentences from a Continuous Space [Samuel R. Bowman et al. 2016]
3. Auto-Encoding Variational Bayes [Diederik P. Kingma et al. 2014]
4. Natural Language Toolkit: <https://www.nltk.org/>