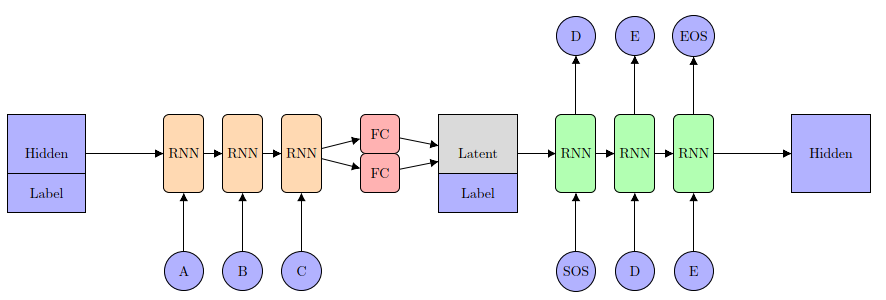
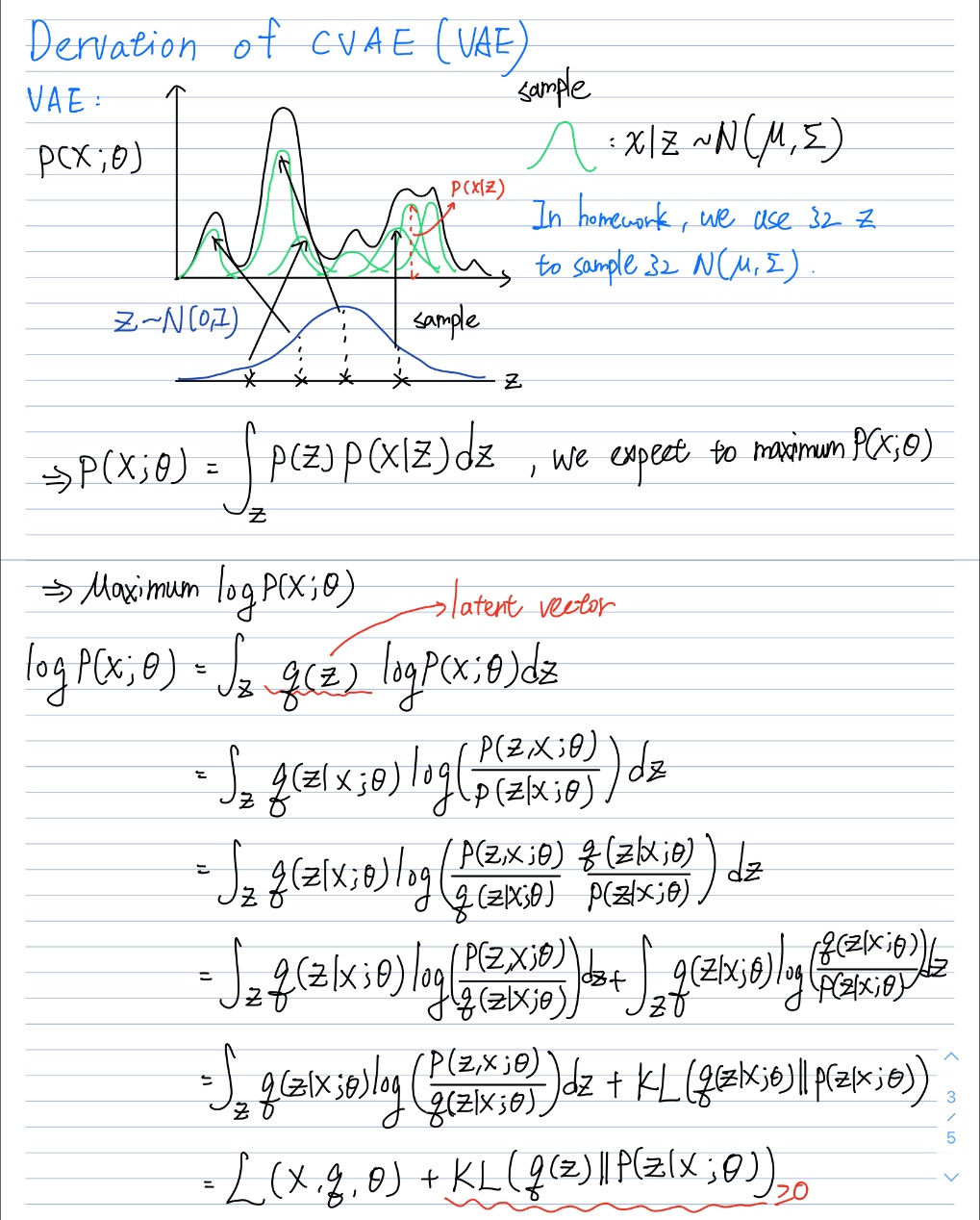
Lab 5 Conditional Sequence to Sequense VAE

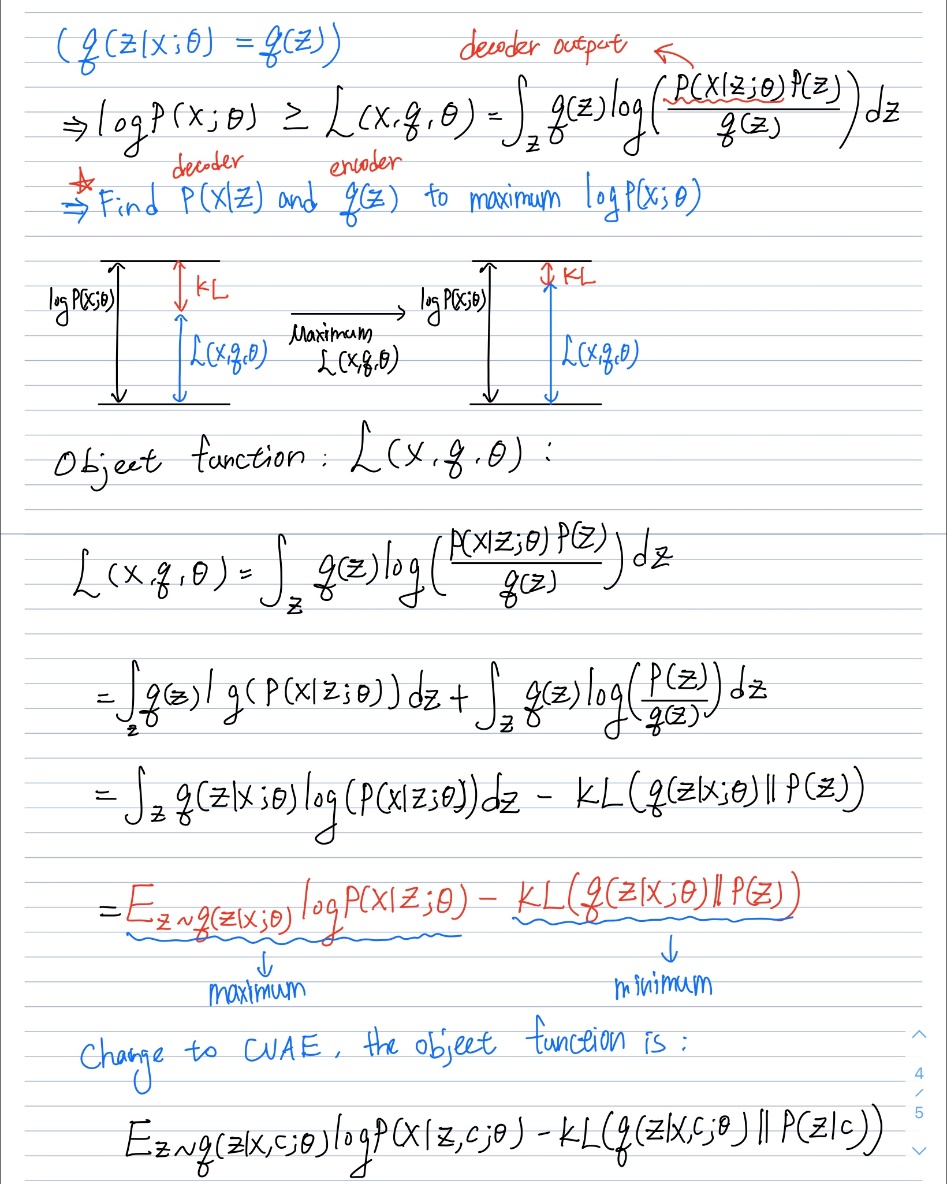
1. Introduction

Using CVAE to change the tense of the verbs , ex. abandon -> abandoned , input “abandon” to encoder with its tense(sp) , and use output and the target tense(p) as decoder input , and expect the verb “abandoned” as output .



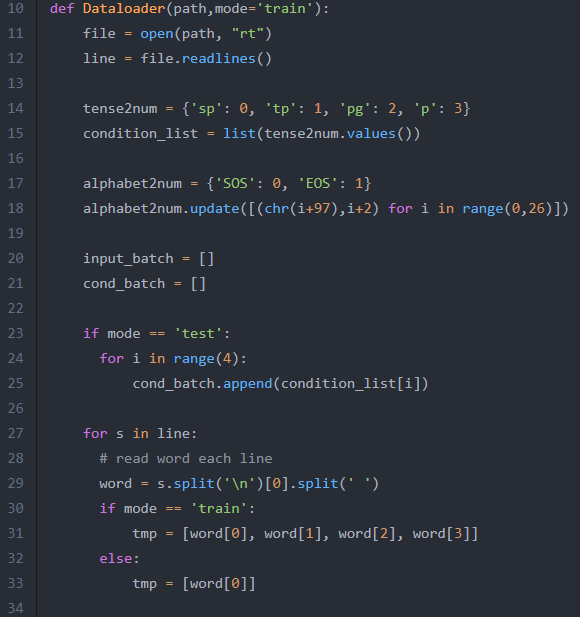
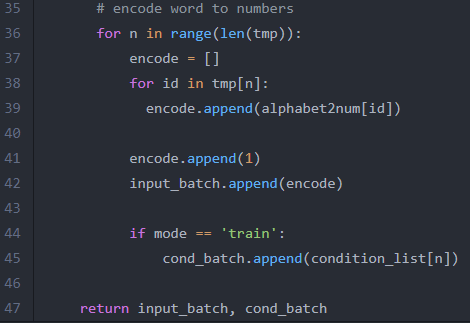
1. Derivation of CVAE





1. Implementation details

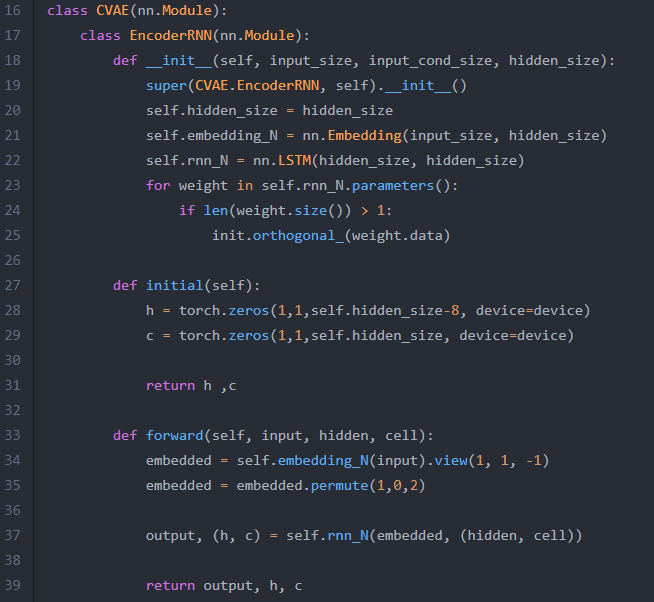
Dataloader (dataloader.py):

I encode the “SOS” , “EOS” and 26 alphabet to 0~27 (input size is 28 ) and tense encoded “sp” , “tp” , “pg” , “p” to 0~3 . My dataloader will return two object , “input\_batch” , which contains 4908 verbs for training or 10 for testing ; “cond\_batch” , 4908 tense for training or 10 for testing .

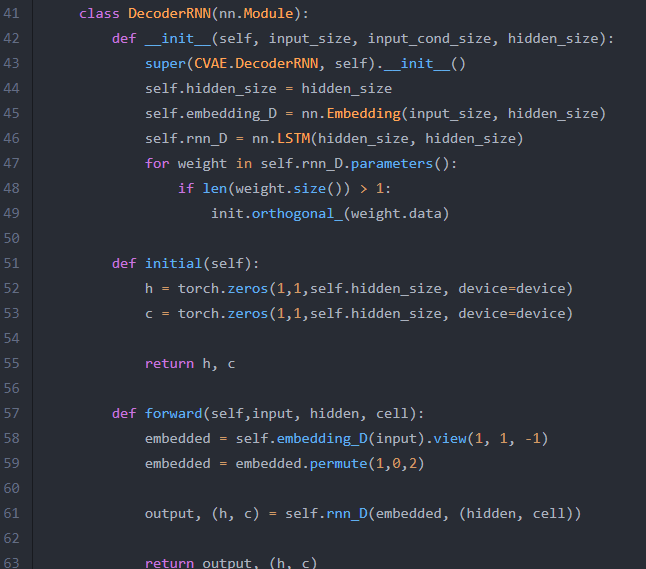
CVAE (model.py):

**Encoder** :



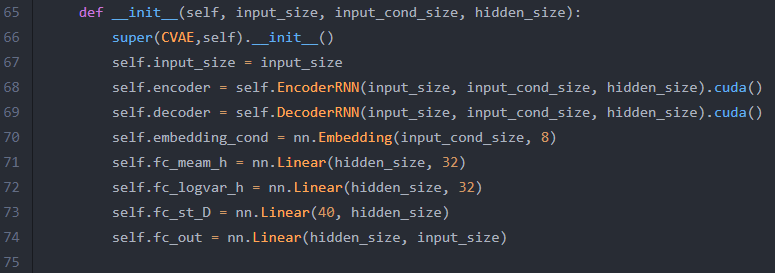
Includes one embedding layer and one LSTM layer , and I found that initial an orthogonal weight will get the better result .

**Decoder** :



Includes one embedding layer and one LSTM layer , and I found that initial an orthogonal weight will get the better result .

**Other layers** :



**self.embedding\_cond** : Embeds the conditional to size 8 .

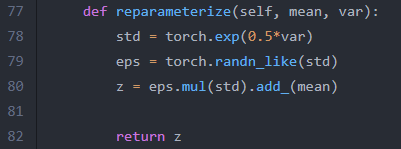
**self.mean\_h** : A fully connection layer to get size 32 mean from final hidden unit of encoder .

**self.logvar\_h** : A fully connection layer to get size 32 logvar from final hidden unit of encoder .

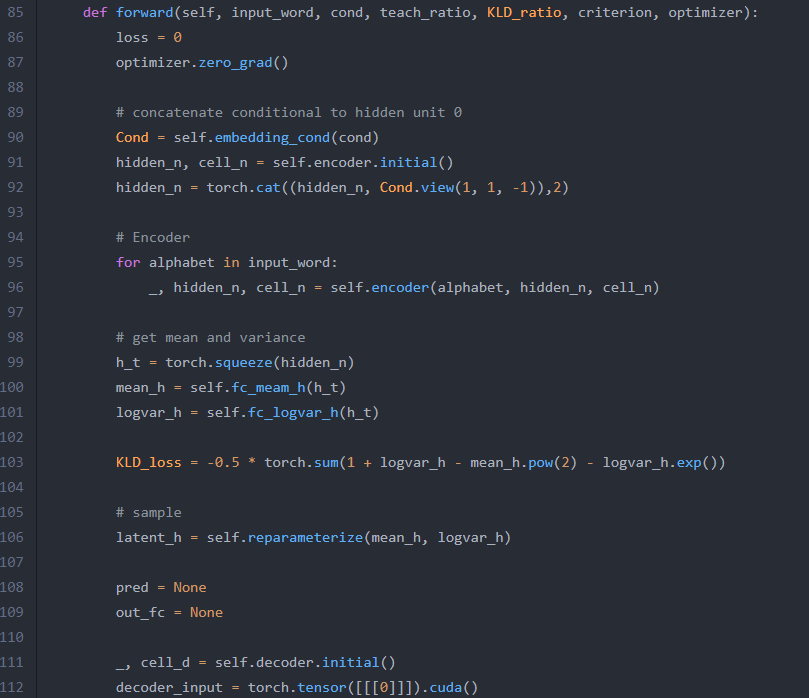
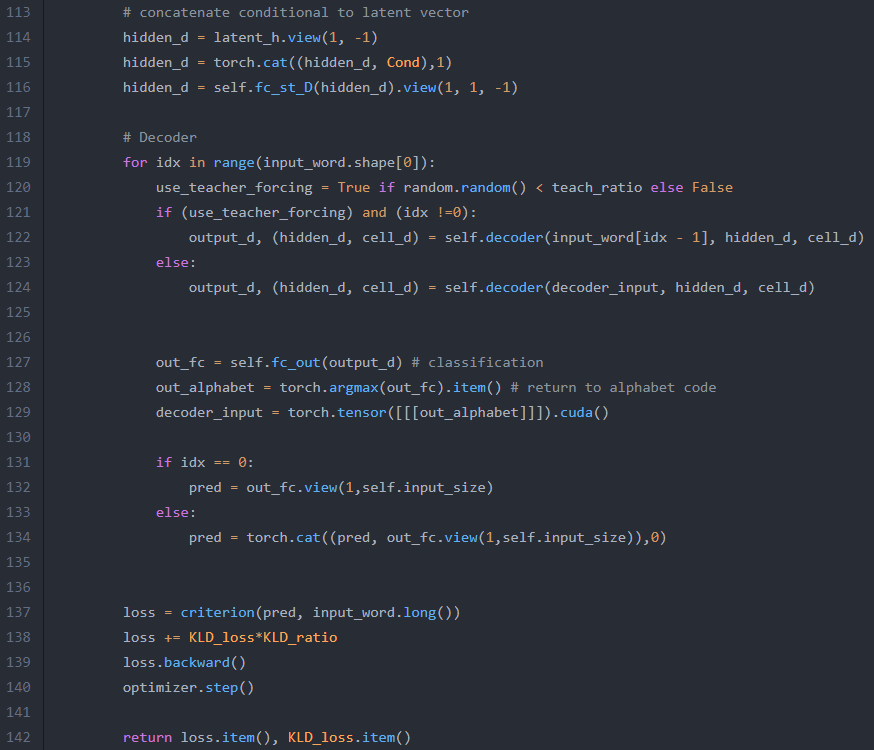
**self.fc\_st\_D** : A fully connection layer to change the size of hidden unit put into the decoder at initial state from 32+8 to 256 .

**self.fc\_out** : A fully connection layer to change the size of the output of decoder from 256 to 28(input size) .

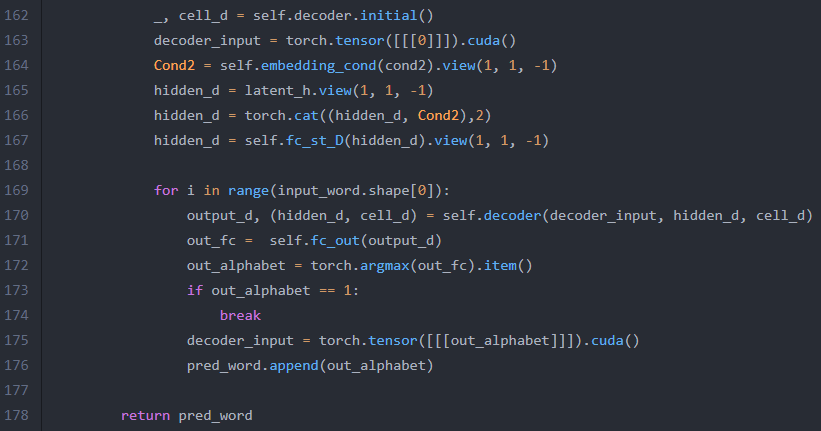
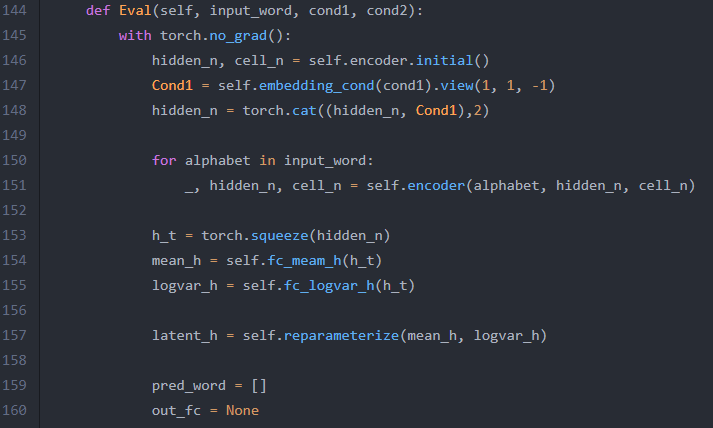
**Reparameterize** :



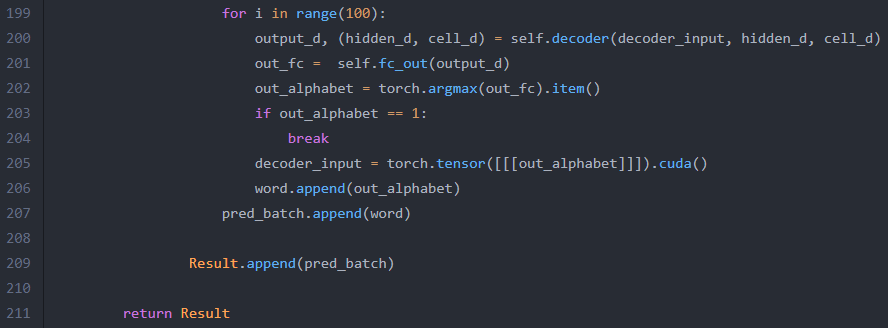
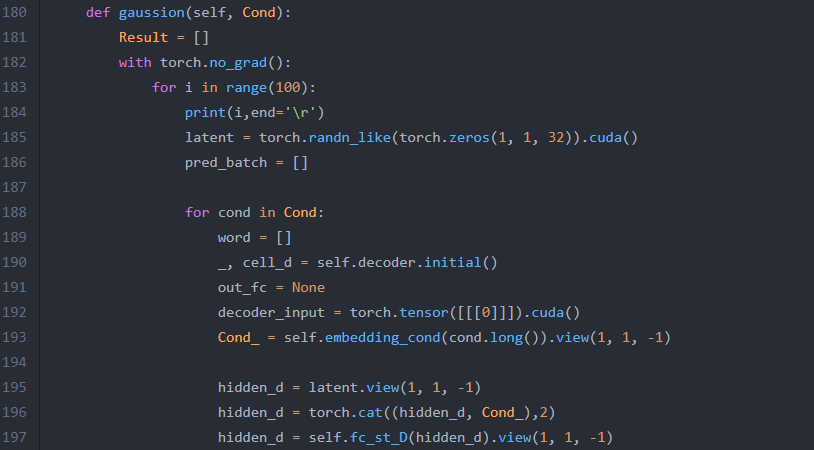
**Forward** :

**Evaluation** (for bleu testing ):



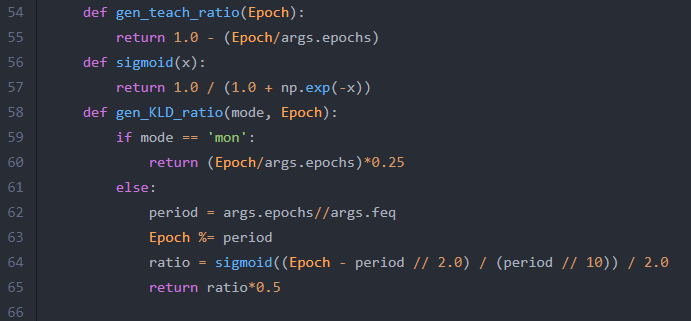
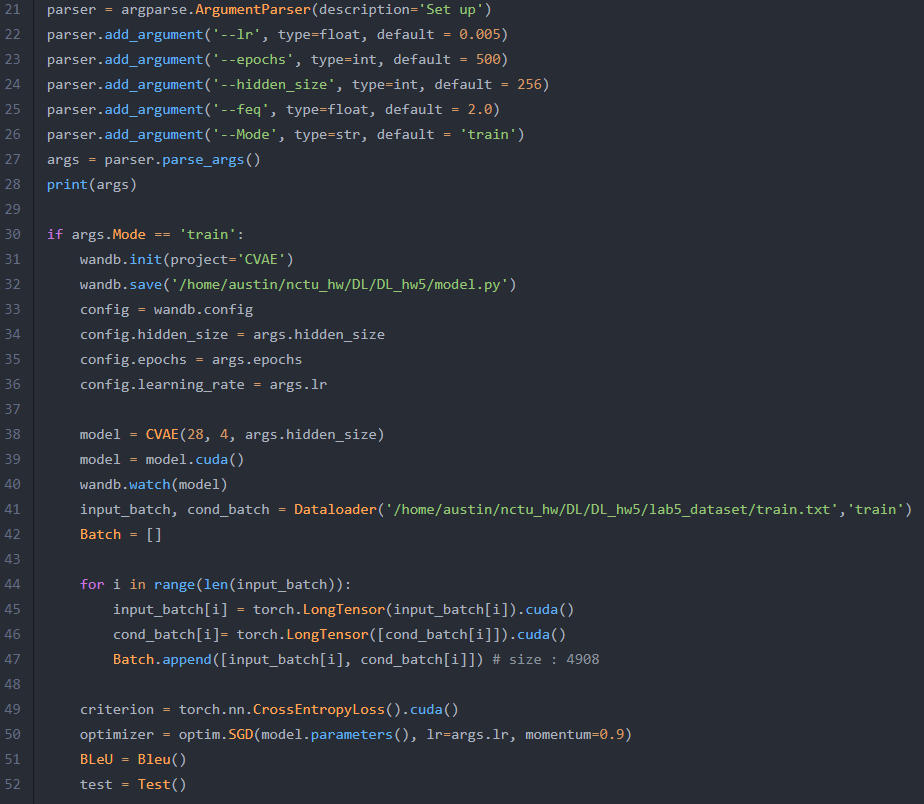
**Gaussian generation** (In line 185 : torch.randn\_like(torch.zeros(1, 1, 32)) is noise):

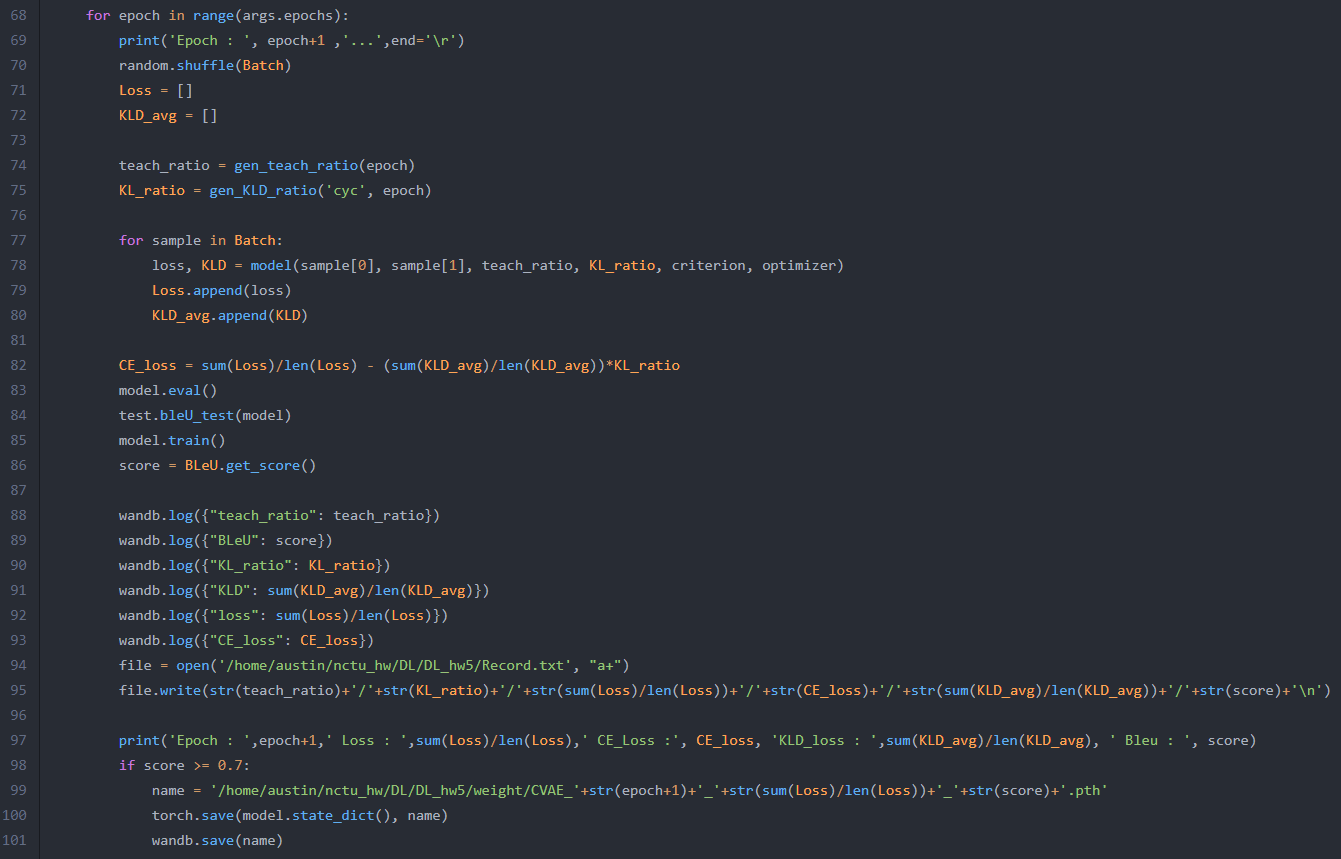


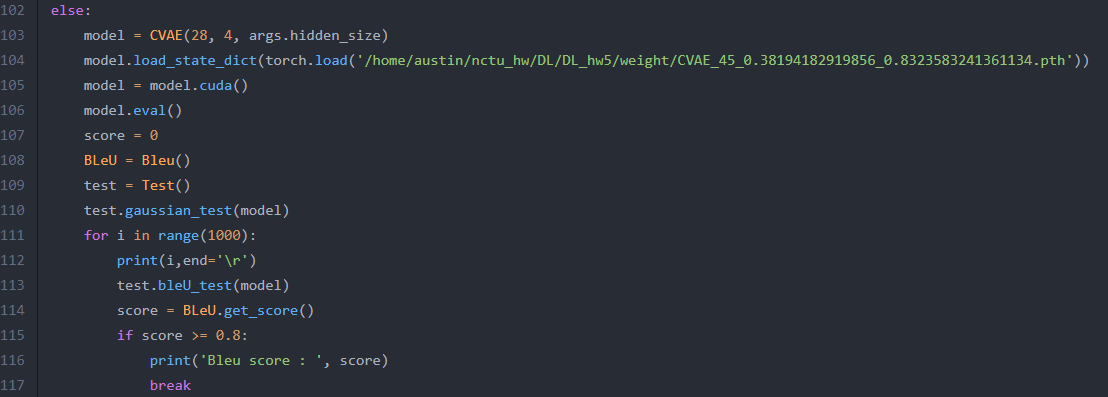
**Test** (test.py) : Test tense switching and gaussian score .

**blue.py** : compute bleu score .

**main.py ( for training and testing ):**

****

****

****

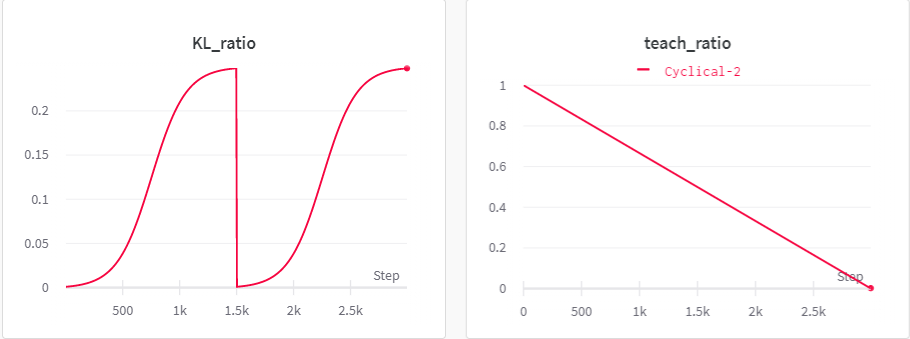
Hyperparameters :

Learning rate : 0.005 , Epochs : 500

I shuffle the training data with each epoch .

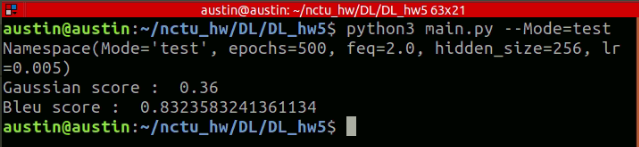
KL weight : sigmoid with two cycles (max : 0.25)

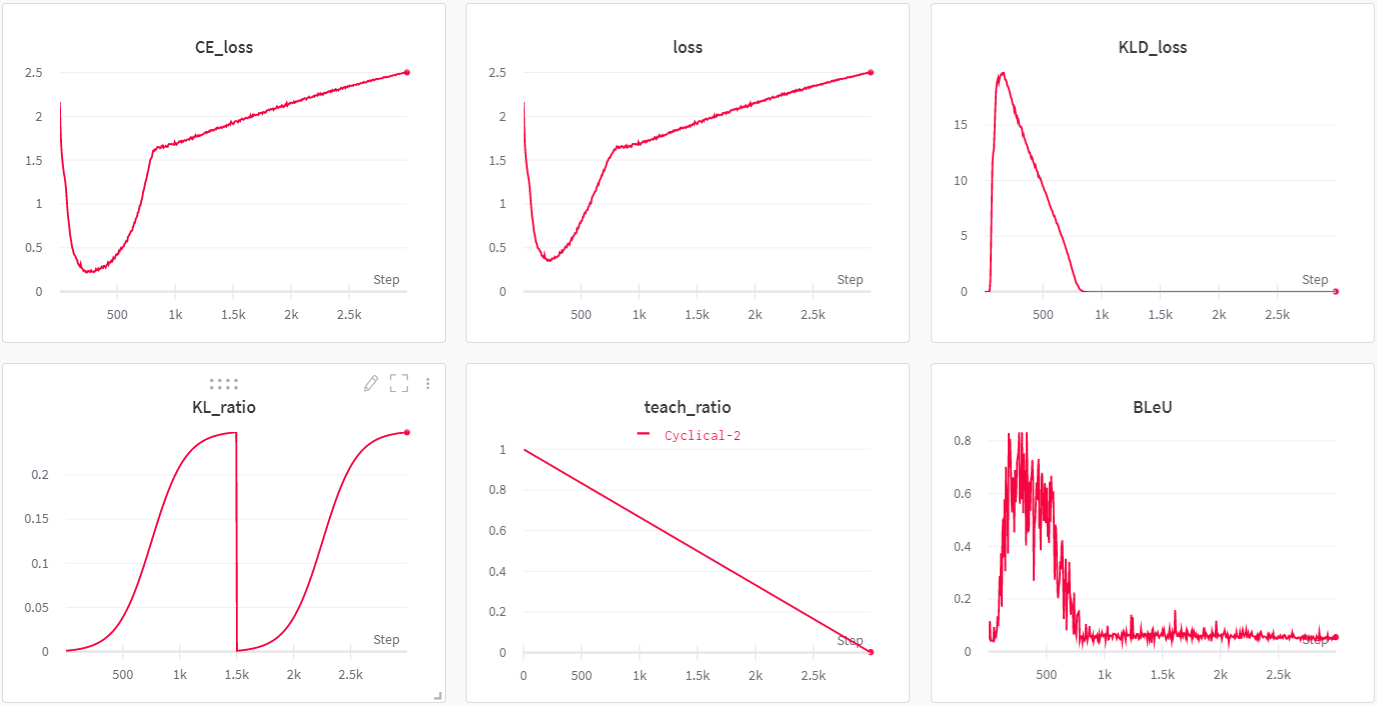
Teach : from 1 down to 0 .



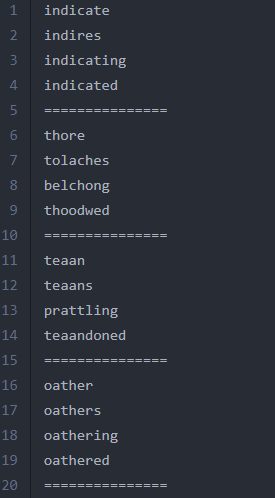
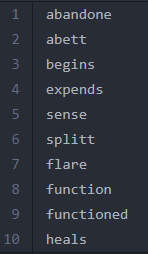
1. Result and discussion

( I use the model weight saved at 45 epochs . ):

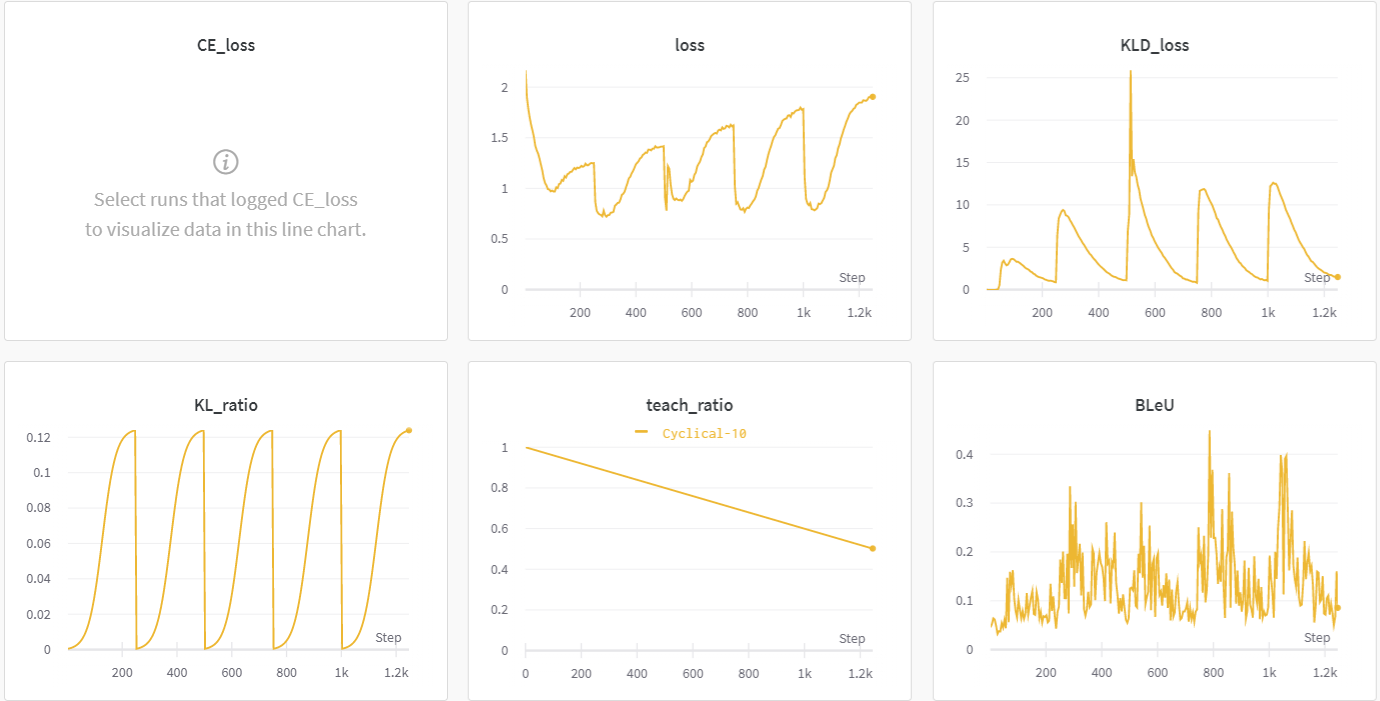




Bleu / Gaussian example :



I found that the bleu score is positively related to KLD loss , the two curves are very similar . In the early stages of training , KL weight was very low , and reconstruction loss was keep going down . When the KL weight up to 0.002 , reconstruction loss started to go up , and made the KLD loss and bleu down . I assume that when the KLD loss low enough , the reconstruction loss will keep down , so I set the max value of kl weight only 0.25 to ensure that the reconstruction loss drops enough to get high bleu score . For learning rate , because I update the model for every words in one epoch , It is better to use lower rate , and I find that 0.005 can make it almost drop vertically . For teach ratio , the higher ratio at the early stages can make loss drop steadily , so simply using monotonic mode . I found the issue that in the second cycle of kl weight , when it went down again , it doesn’t make reconstruction loss down , and KLD loss doesn’t go up again too . So I redo the experiment , this time I set 10 cycles of kl weight :



Although my computer was shutdown when epoch closed to 250 , only half of data , but it still can prove that my assumption is correct : When the KLD loss low enough , the reconstruction loss will keep down . And take a look at higher cycles , it makes the bleu score drops before it has time to rise enough , but the highest score in each cycle has an upward trend .

The curve of using monotonic KL weight (from 0 down to 0.25) :

