# Deep Learning Practice Lab 5

0856056 Yu-Wun Tseng

May 2020

## 1 Introduction

The goal of this lab is to implement a conditional seq2seq VAE for English tense conversion and generation. The training input are multiple a words with four tenses, the model need to generate the words of four tenses. The encoder and decoder are both LSTM as last Lab.

## 2 Derivation of CVAE

Conditional variational autoencoder(CVAE) is an extension of VAE with some condition. The derivation is on Figure 1.

## 3 Experiment Setups

This lab is done by using Pytorch. The procedure follows the official tutorial [1] [2] . However, the official tutorial doesn't consider the batch. Thus, I refer a seq2seq github repository[3] which using batch for training.

### 3.1 Encoder and Decoder

The encoder inherits the nn.Module class. Similar to Lab4, the encoder consists of an embedding layer and a recurrent neural network. In addition, I add two linear layers to generate the mean and variance. The encoder hidden state is initialized with zero tensors concatenated with embedded condition.

The architecture of the decoder is as same as Lab4, however, the hidden state initialization is different. Similar to the encoder, the hidden state is concatenated with embedded condition. However, not using zero tensors but using the result of reparameterization trick. The hidden sizes of both encoder and decoder are the original hidden size plus condition embedding size, which is set to 4.

### 3.2 Dataloader

In order to fetch the input and the target tensor, I implement a dataloader which uses a vocabulary to convert words into tensors. First, I use training data to build a vocabulary, which contains two methods. One is to convert words into vector, the other is to convert vector into words. Next,I implement the ___getitem___ function and convert the words into tensor. If the mode is train, the dataloader generate a 2d tensor, the first dimension is four, the second dimension is sequence length. If the mode is test, the dataloader generate a list which contains two tuple. The first tuple contains the input tense and the input tensor, while the second tuple contains the target's. Besides, I shuffle the training data.

### 3.3 Reparameterization Trick

The reparameterization trick is for calculating gradients, not using the the mean and the variance generated from the encoder directly, but using addition and multiplication to generate the latent vector. The equation is below:

$$z = \mu + \sigma \times \epsilon \text{ where } \epsilon \sim \mathrm{N}(0, 1) \tag{1}$$

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

$g(z|c)$ 为任意的概率分布

$$\Rightarrow \int g(z|c)\log P(x|c)\,dz = \int g(z|c)\log p(x,z|c)\,dz - \int g(z|c)\log p(z|x,c)\,dz$$

$$= \int g(z|c)\log P(x,z|c)\,dz - \int g(z|c)\log g(z|c)\,dz \Rightarrow \mathcal{L}$$

$$+ \int g(z|c)\log g(z|c)\,dz - \int g(z|c)\log P(z|x,c)\,dz \Rightarrow KL$$

$$= \mathcal{L}(x,g,\theta) + KL(g(z|c)\,\|\,P(z|x,c))$$

$$\Rightarrow \int g(z|c)\log P(x|c)\,dz = \log P(x|c)\int g(z|c)\,dz$$

$$= \log P(x|c) \geq \mathcal{L}(x,g,\theta)$$

$$\mathcal{L} = \int g(z|c)\log P(x,z|c)\,dz - \int g(z|c)\log g(z|c)\,dz$$

$$= \int g(z|c)\log P(x|z,c)\,dz + \int g(z|c)\log P(z|c)\,dz - \int g(z|c)\log g(z|c)\,dz$$

$g(z|c)$ 选 $g(z|x,c;\theta')$

$$\Rightarrow \mathcal{L} = E_{z\sim g(z|x,c;\theta')}\log P(x|z,c;\theta) + E_{z\sim g(z|x,c;\theta')}P(z|c;\theta) - E_{z\sim g(z|x,c;\theta')}g(z|x,c;\theta')$$

$$= E_{z\sim g(z|x,c;\theta')}\log P(x|z,c;\theta) - KL(g(z|x,c;\theta')\,\|\,P(z,c;\theta))$$

Figure 1: Derivation of CVAE

```python
def Gaussian_predict(encoder, decoder, vocab, batch_size=64, laten_size=32, condition_size=8, plot_pred=False):
    from train import condition_embedding
    outputs = []

    with torch.no_grad():
        batch_size = 100

        # sample 100 Gaussian
        laten_variable = torch.randn((batch_size, laten_size), device=device).view(1, batch_size, -1)

        # get 4 tense embedding tensor
        embedded_tenses = condition_embedding(condition_size , batch_size)

        # record outputs
        output_tensors = torch.zeros(4, vocab.max_length, batch_size)# (tense, seq_len, batch_size)

        # 4 tense iteration
        for index, embedded_tense in enumerate(embedded_tenses):

            decoder_input = torch.tensor([[SOS_token] for i in range(batch_size)], device=device)

            output = torch.zeros(vocab.max_length, batch_size)

            # init decoder hidden state and cat condition
            decoder_hidden = decoder.initHidden(laten_variable, embedded_tense, batch_size)

            #----------sequence to sequence part for decoder----------#
            for di in range(vocab.max_length):
                decoder_output, decoder_hidden = decoder(
                    decoder_input, decoder_hidden)
                topv, topi = decoder_output.topk(1)
                decoder_input = topi.squeeze().detach()  # detach from history as input
                output[di] = decoder_input

            # get predict tensors
            output_tensors[index] = output

        # transpose tensor from (tense, seq_len, batch_size) to (batch_size, tense, seq_len)
        output_tensors = output_tensors.permute(2, 0, 1)

        # convert input into string
        for idx in range(batch_size):
            outputs.append([vocab.indices2word(tense.data.numpy()) for tense in output_tensors[idx]])

    return outputs
```

Figure 2: Code of Gaussian Prediction

### 3.4 Hyperparameters

The hidden size is 256, the latent size is 32 and the condition embedding size is 4. The batch size is 64, the teacher forcing ratio is 1, the learning rate is 0.03 and the epochs is 1000. The KL weight has two annealing methods, one is monotonic, the other is cyclical. The epoch parameter is passed into the function. For monotonic, if the epoch is larger than 50, the KL weight is 1. If the epoch is no larger than 100, the KL weight is 0.01 times the epoch. For cyclical, if the epoch mod 100 is larger than 50, the KL weight is 1. If the epoch mod 100 is no larger than 50, the KL weight is 0.02 times the epoch mod 100.

### 3.5 Train and Evaluate

The train function and trainIter function are similar to Lab4. However, for training four tense, the training of a batch will repeats four times. In addition, the loss is cross entropy plus kl divergence.

The evaluation has two part, one is Gaussian noise with four tenses, the other is tense conversion with BLEU score. The Gaussian noise is generated as the Figure 2. The sampled noise is fed into decoder as the initial hidden state. The tense conversion is to use the input with input tense to generate the corresponding output with target tense.

## 4 Results

### 4.1 Loss curve

The crossentropy loss and the KL loss of monotonic weight annealing are shown on Figure 3 and 4. The crossentropy loss curve deceased stably. The KL loss curve was very high in the beginning, and kept very low after first epoch. It's due to the difference between the model distribution and the data distribution. The crossentropy loss and the KL loss of cyclical weight annealing are shown on Figure 5 and 6. Similar to monotonic, the crossentropy loss curve deceased stably and the KL loss began with very high value and dropped after first epoch.

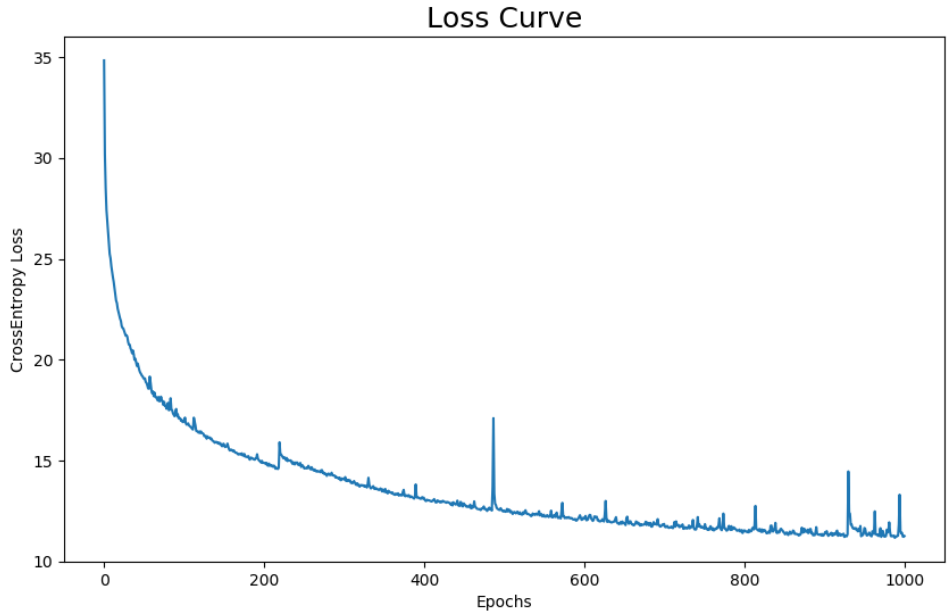



Figure 3: CrossEntropy Loss trend of Monotonic Wight Annealing

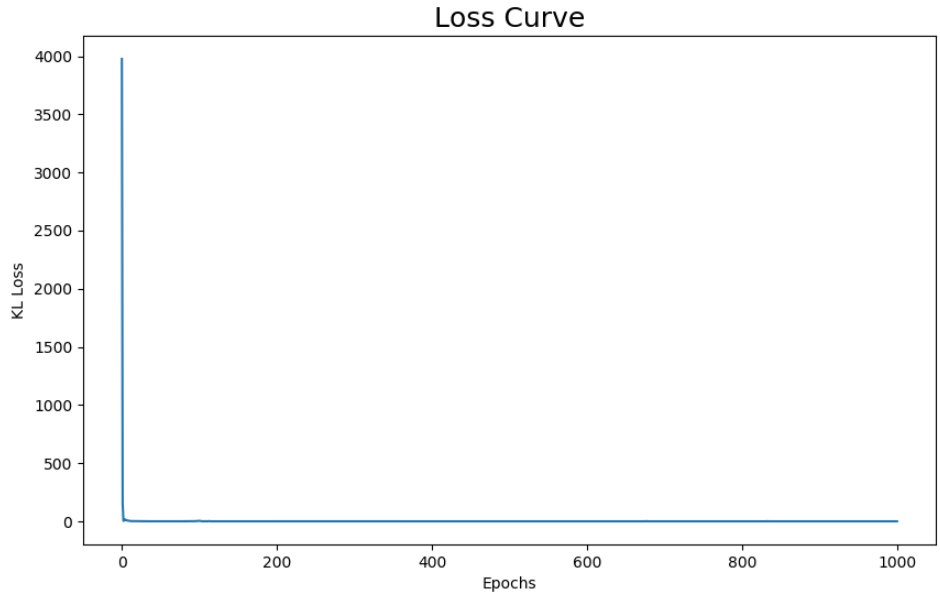



Figure 4: KL Loss trend of Monotonic Wight Annealing

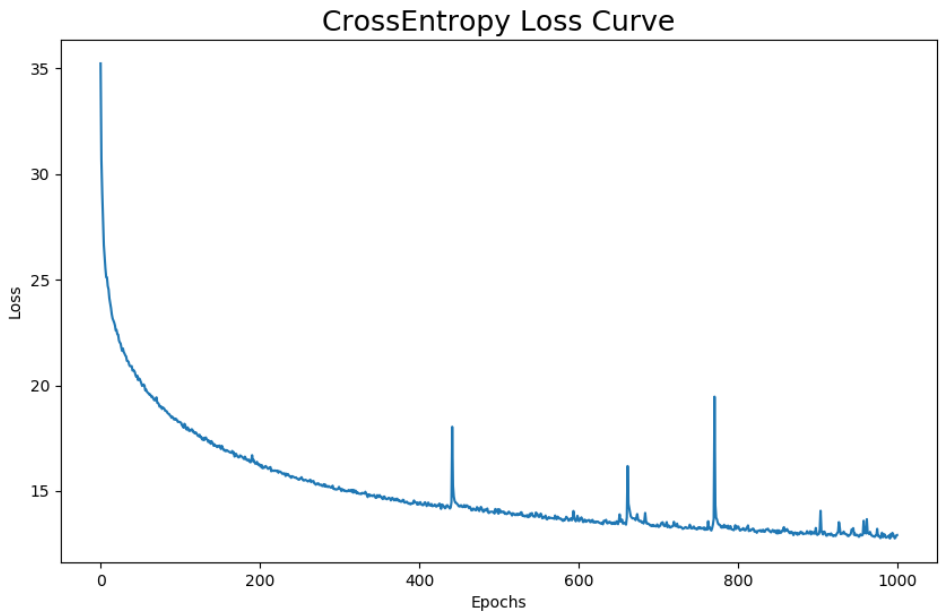



Figure 5: CrossEntropy Loss trend of Cyclical Wight Annealing

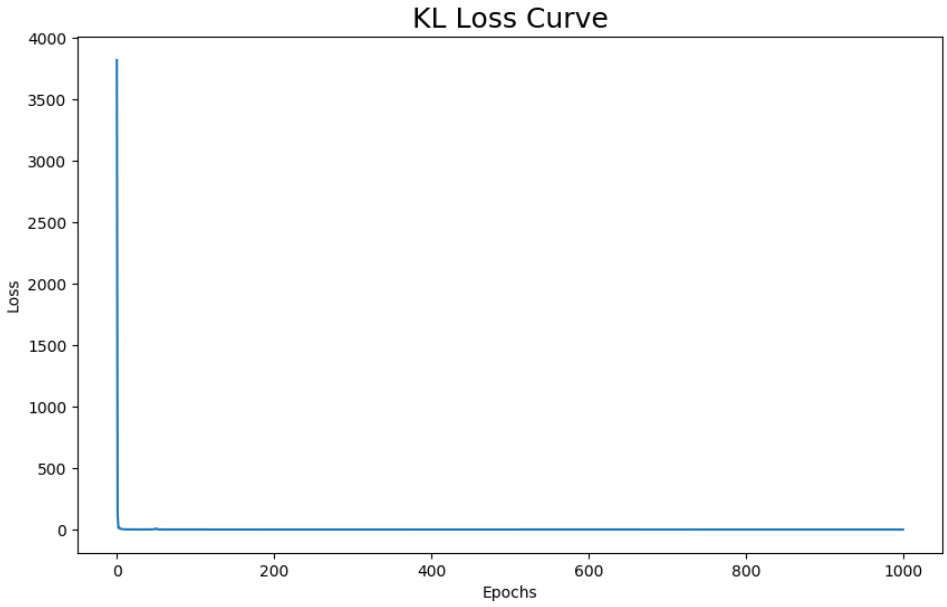



Figure 6: KL Loss trend of Cyclical Wight Annealing

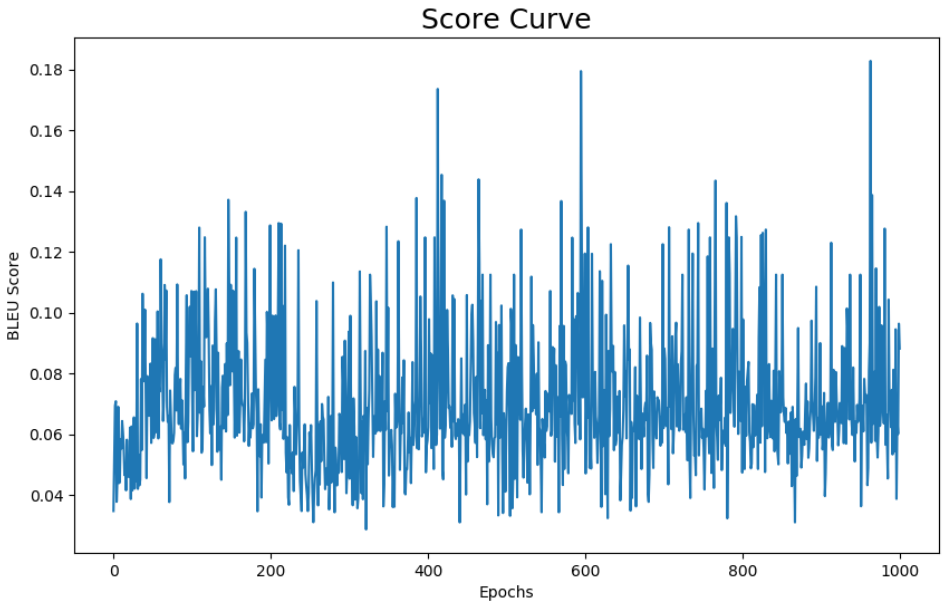



Figure 7: BLEU Score trend of Monotonic Wight Annealing

### 4.2 Score curve

The Gaussian score and the BLEU score of monotonic weight annealing are shown on Figure 7 and 8. The BLEU curve is very volatile and doesn't improve significantly. Most of the score are at range from 0.6 to 0.8. The Gaussian curve is weird, in the beginning, it's zero. And after 500 epochs, the most scores are close to 1. It's because the model only predict the same word with four tenses.

The Gaussian score and the BLEU score of cyclical weight annealing are shown on Figure 9 and 10. The BLEU score is much lower than monotonic, it may due to the smaller KL wight at the cycle beginning. In addition, the score drops after 200 epochs. Different from monotonic, the Gaussian score is zero, the model can't generate any correct words.

### 4.3 Predict result

The Gaussian prediction and the BLEU prediction of monotonic weight annealing are shown on Figure 11 and 12. The model couldn't predict the right tense conversion, and only predict the same word with four tenses.

The Gaussian prediction and the BLEU prediction of cyclical weight annealing are shown on Figure 13 and 14. The model couldn't generate any right word with four tenses, and predict the right tense conversion, either.

## 5 Discussion

First, I thought it's a reconstruct task, given the input with the tense, generate the same word with the same tense. But my classmate thought it's similar to Lab4, given a word with a tense, predict the target word with target tense, e.g, given sit and condition tp, predict sits. Thus, one data has 16 pairs, which equal to four tenses times four tenses. I feel confused, but I modified my code to fit it. However, the model couldn't learn well. After discussed with another classmate, we found the useful hint from the TA's ppt. Thar is the original way I did. Therefore, I modified my code again...

However, the model still couldn't learn. I found that the problem was from the condition embedding. I used the nn.Embedding provides by pytorch, but I used it as a function and generated a new embedding layer for each epoch. The model didn't backward the embedding layer. Thus, the model didn't learn the better way to embed condition. Consider the time is not enough, I decided to use one-hot. Though I still get very low score, the model model can generate one correct data.

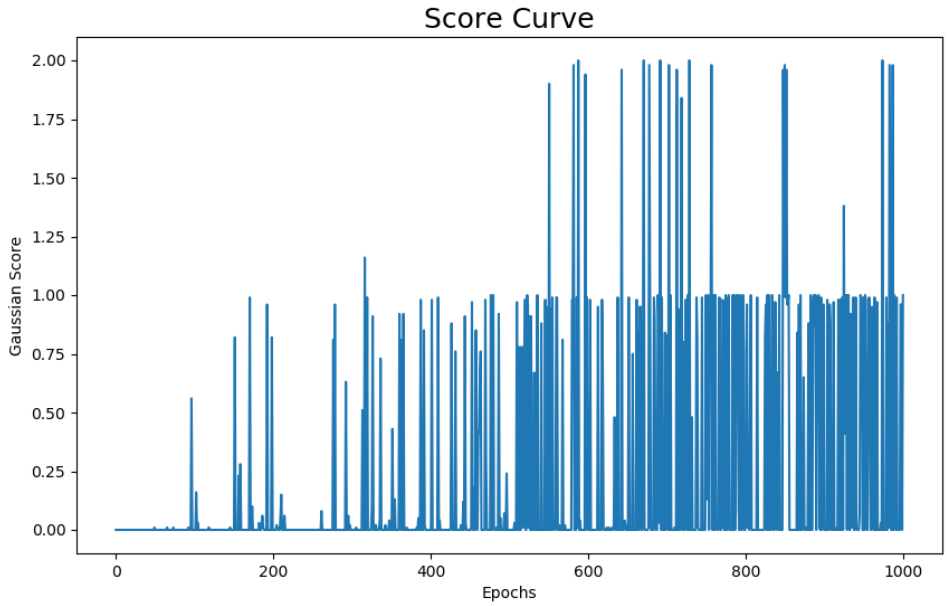



Figure 8: Gaussian Score trend of Monotonic Wight Annealing

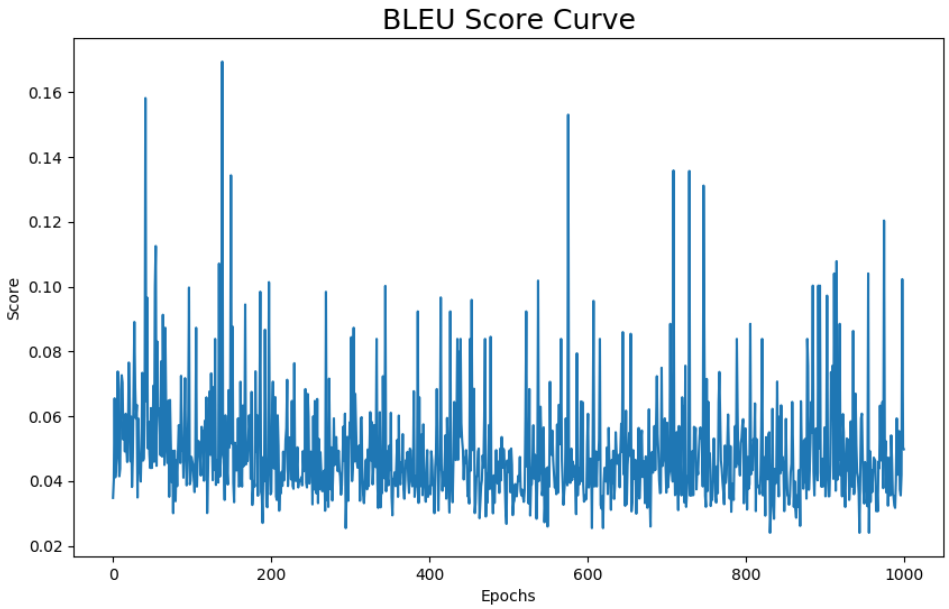



Figure 9: BLEU Score trend of Cyclical Wight Annealing

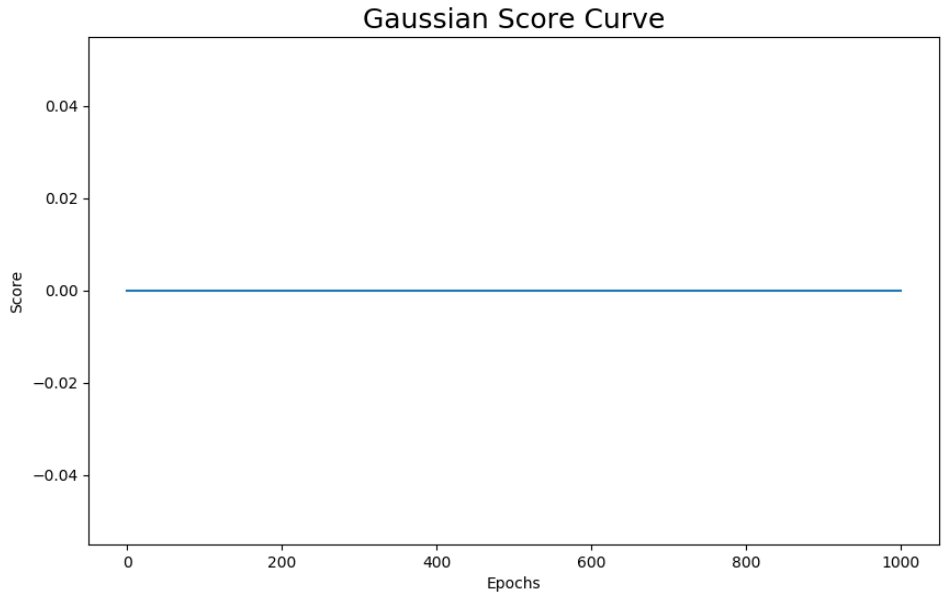



Figure 10: Gaussian Score trend of Cyclical Wight Annealing

# References

[1] Sean Robertson. Nlp from scratch: Translation with a sequence to sequence network and attention.

[2] Basic vae example.

[3] pengyuchen. Pytorch-batch-seq2seq.

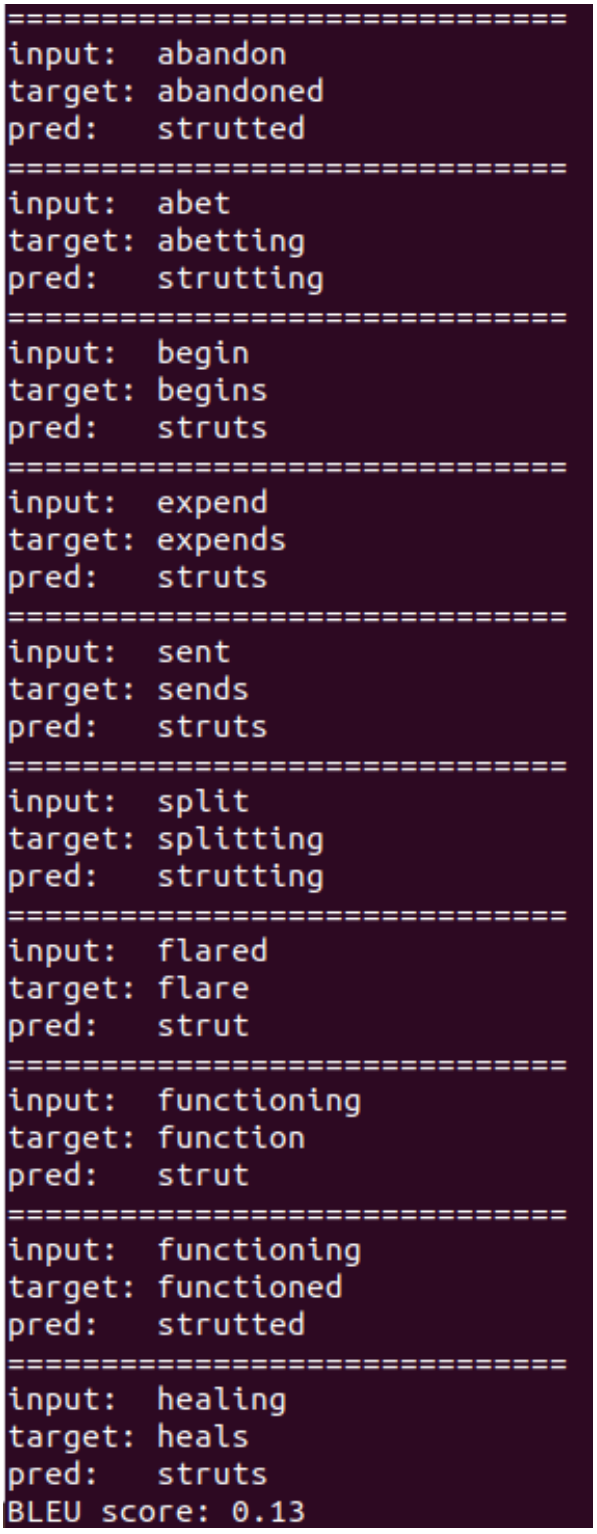
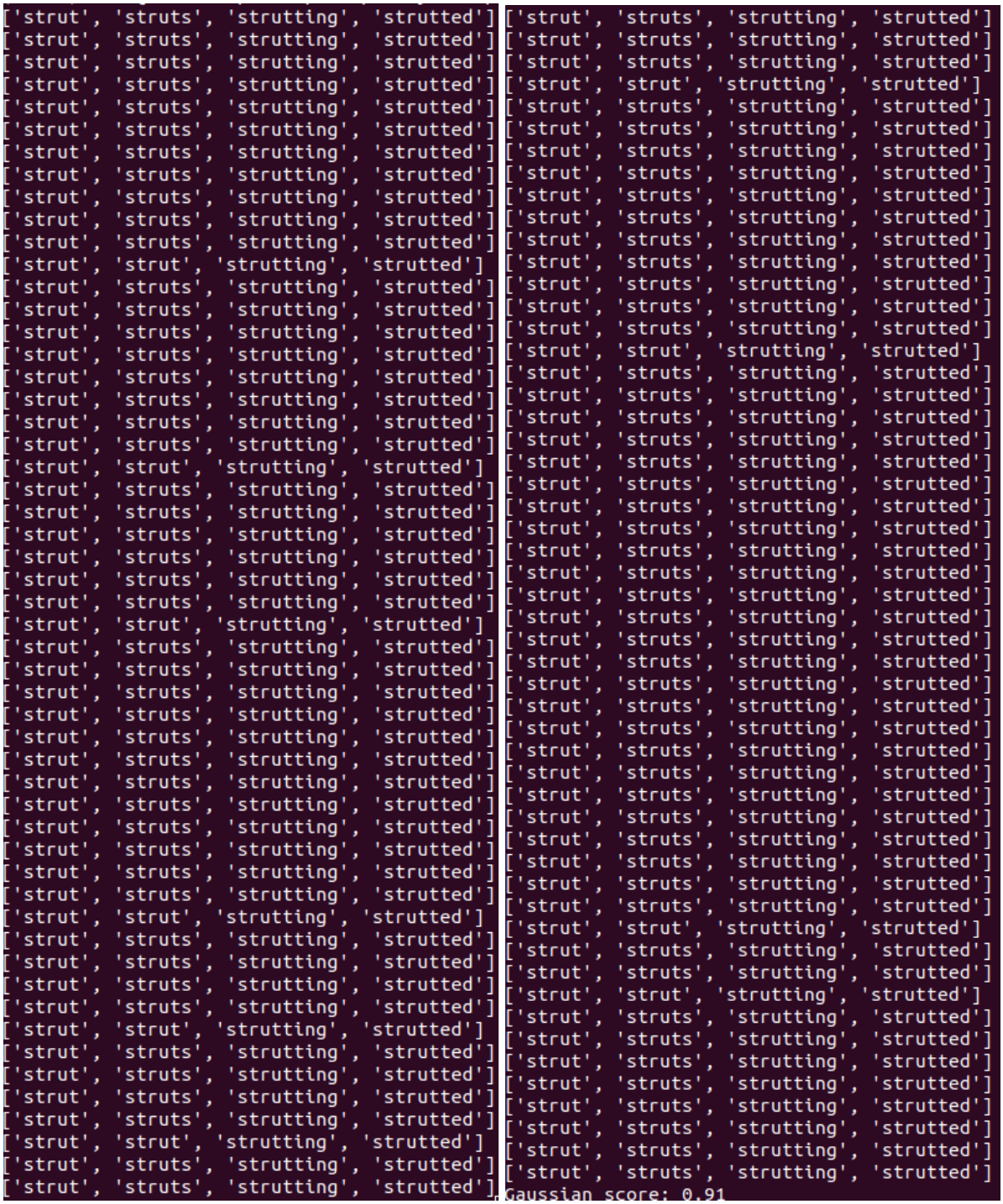



Figure 11: BLEU Prediction of Monotonic Wight Annealing




(a) (b)

Figure 12: Gaussian Prediction of Monotonic Wight Annealing

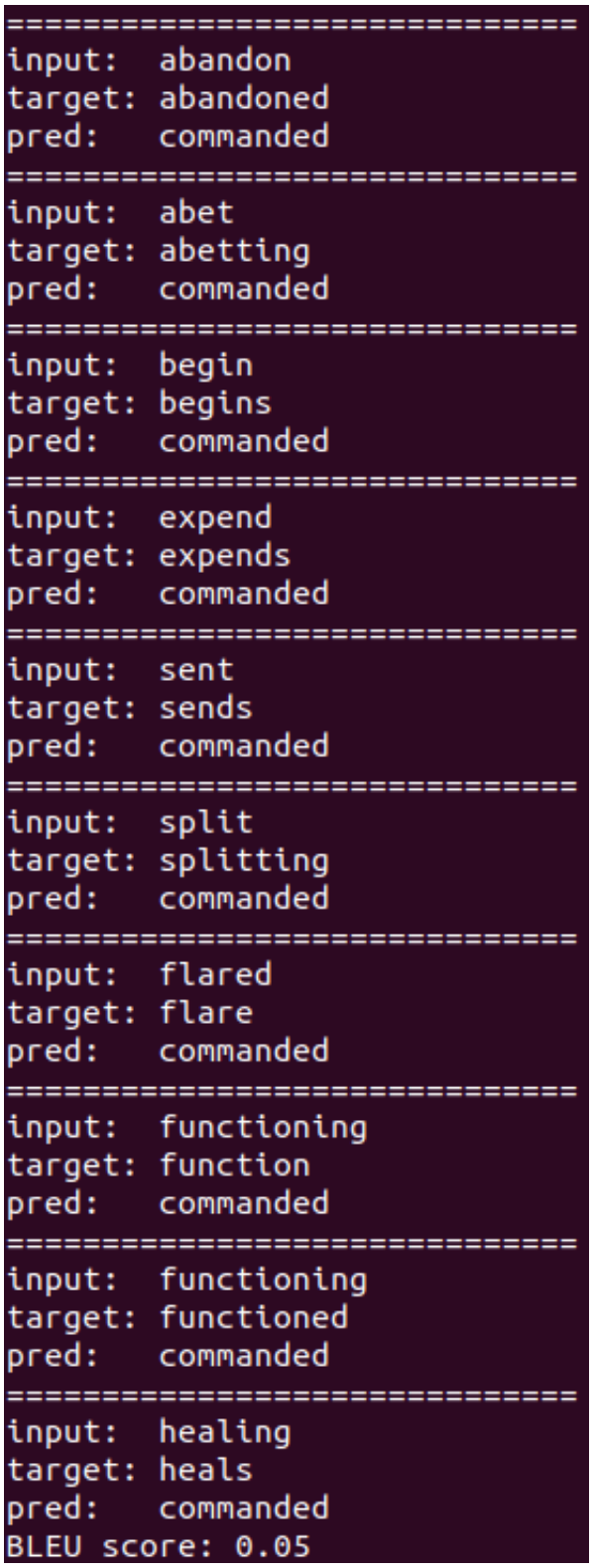
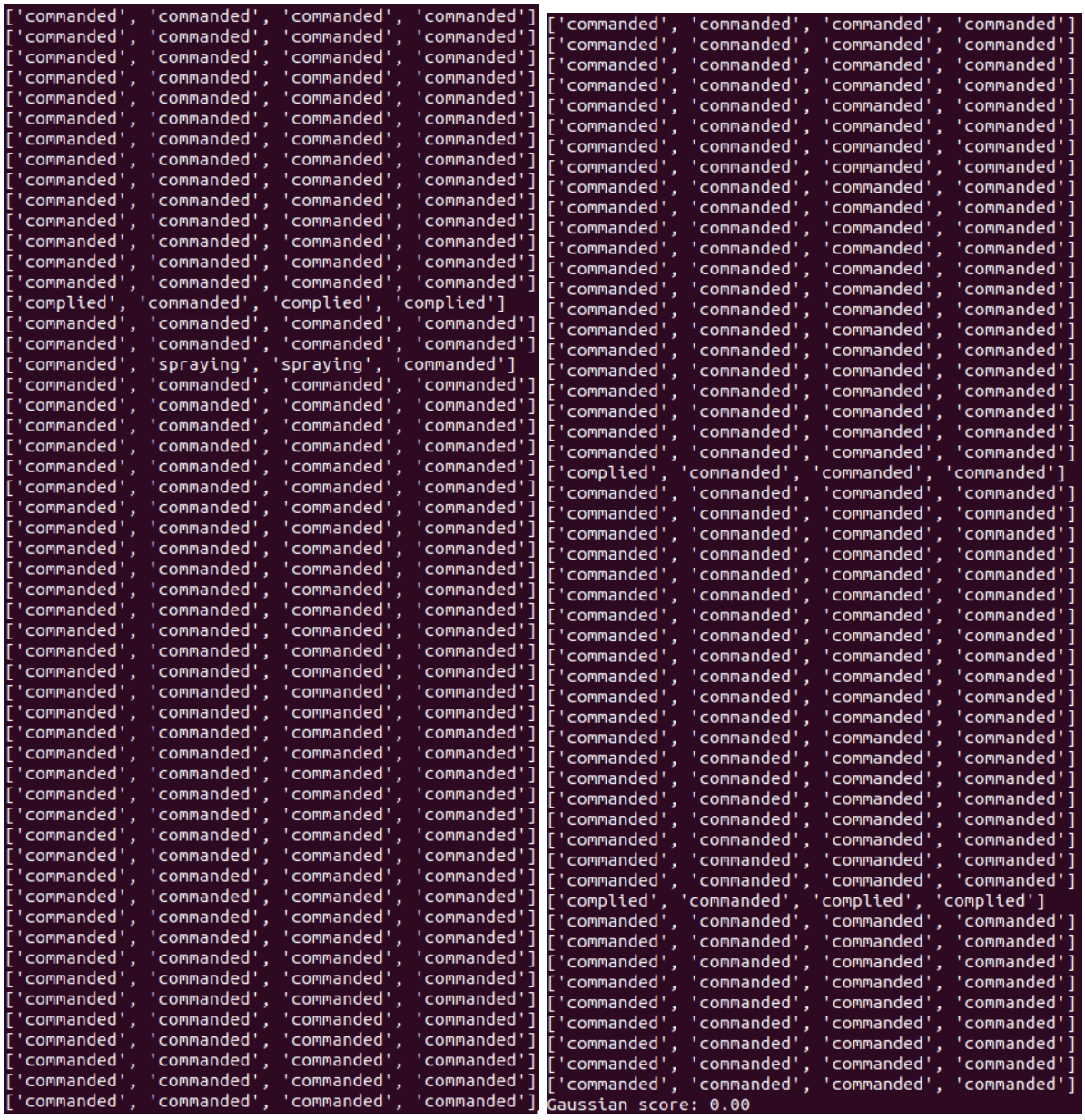



Figure 13: BLEU Prediction of Cyclical Wight Annealing




(a) (b)

Figure 14: Gaussian Prediction of Cyclical Wight Annealing