Lab5

鄭余玄

1 Introduction

In this lab, we are going to implement a conditional variational auto-encoder (CVAE) using sequence-to-sequence as the encoder and decoder. I will explain the details if my implementation in Section 2, and the training details and results are shown in Section 3.

2 Method

In this section, I will describe the method of prepossessing data in Subsection 2.1, CVAE model in Subsection 2.2, training in Subsection 2.3, testing in Subsection 2.4, and finally the normal latent code test in Subsection 2.5.

2.1 Data

We construct a dictionary to map all possible characters, including all alphabets and special tokens: unknown (`UNK`), start (`SOS`), and end (`EOS`), to numbers. The training data is consists of several lines of the four tenses of a word separated by a whitespace. Each word in a line is appended with an `EOS` token, and is converted to a tensor by the dictionary. The testing data is consists of lines of two tenses of a word. Both words in a line are converted to tensors with `EOS` tokens, and additionally being labeled with a specific tense.

2.2 Conditional Variational Auto-Encoder

In this subsection, I am going to introduce the main architecture of the CVAE. Equation (1) shows the variational lower bound of the CVAE architecture:

$$L = \mathbb{E}_{\mathbf{Z}\sim q(\mathbf{Z}|\mathbf{X},\mathbf{c};\theta')} \log p(\mathbf{X}|\mathbf{Z},\mathbf{c};\theta) \tag{1a}$$

$$- w_{\mathcal{KL}} \times \mathcal{KL}[q(\mathbf{Z}|\mathbf{X},\mathbf{c};\theta')||p(\mathbf{Z}|\mathbf{c})] \tag{1b}$$

where the first term is calculated by the loss of the neural network, the second term is described in Subsection 2.2.3, and $w_{\mathcal{KL}}$ is the KL-weight.

The following section comprise with the order of the training flow. Note that the embedding weights in Subsection 2.2.1 and 2.2.4 are shared.

2.2.1 Encoder Initial Hidden

The initial hidden vector of the encoder is a concatenation of a zero (latent) vector of dimension d_l $(= d_h - d_c)$ and an embedded tense vector, which plays a role of the condition in the CVAE architecture, of dimension d_c, which results in a hidden vector of dimension d_h. Note that the weights of the condition embedding is shared with the decoder, which will discuss more in Subsection 2.2.4.

2.2.2 Encoder

The encoder receives a character of the input word and a previous hidden vector, and encodes into the current hidden vector using an RNN unit. This process repeats until all characters in the input word are encoded into hidden vectors. The initial hidden vector is described in Subsection 2.2.1, and all outputs from the encoder are discarded.

2.2.3 Reparametrization

The last time-step hidden vector of the encoder is of dimension d_h and directly connects to two different linear layers representing the mean $(\mu(\mathbf{X}))$ and the log-variance $(\log \Sigma(\mathbf{X}))$ of the data $(\mathbf{X})$, and therefore the reparametrization of the latent vector is

$$\begin{aligned}\mathbf{z} &= \mu + \mathbf{\Sigma}^{\frac{1}{2}}\varepsilon \\ &= \mu + e^{\frac{1}{2}\log \mathbf{\Sigma}}\varepsilon\end{aligned}$$

where $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\mathbf{z}$ is of dimension d_l.

In addition, the KL-term in Equation (1b) can be calculated by

$$\mathcal{KL}[\mathcal{N}(\mu(\mathbf{X}), \Sigma(\mathbf{X}))||\mathcal{N}(\mathbf{0}, \mathbf{I})]$$

2.2.4 Decoder Initial Hidden

The initial hidden vector of the decoder is a concatenation of the latent vector $\mathbf{z}$ of dimension d_l and an embedded tense vector of dimension d_c and connects to a linear layer to increase the output dimension to d_h,

which is slightly different from the encoder initial hidden described in Subsection 2.2.1. However, they both share the same weights over the condition embedding.

2.2.5 Decoder

The decoder receives an input vector (typically from the previous output) and a previous hidden vector, and decodes into the current hidden vector and output vector using an RNN unit. This process repeats until the output character is the end token (`EOS`). The initial input vector is the start token (`SOS`) and the initial hidden vector is described in Subsection 2.2.4.

Moreover, there is some probability (teacher forcing ratio) that the input vector directly comes from the training data not the previous decoder output, which helps the decoder learns faster. On the other hand, the input vector has a probability (word dropout ratio) to be replaced as the unknown token (`UNK`) which enhances the robustness of the decoder.

2.3 Train

The main part of training follows the process in Section 2.2. The input data is randomly shuffled to increase the robustness, and the loss is average over the length of the input data. Besides, the output word and tense is the same as ones of the input.

On the other hand, all states of models and the optimizer are saved to a checkpoint file every epoch. Extra information such as the epoch, loss, KL-loss, KL-weights, and BLEU-4 score is also stored.

2.4 Test

The main part of testing also follow the process in Section 2.2. However, the input data need not to be shuffled, and the output word and tense is specified by the data described in Section 2.1. The output tensor is converted to the result decoded word by inverting the dictionary index. Then, the result word calculates the BLEU-4 score with the labelled word.

2.5 Gaussian Test

This part of testing generates a normal-distributed latent vector instead of the original one from the encoder. This latent vector concatenates with all of the four tenses as the initial decoder hidden similar to method in Subsection 2.2.4. The four decoded words should be four tenses of a word.

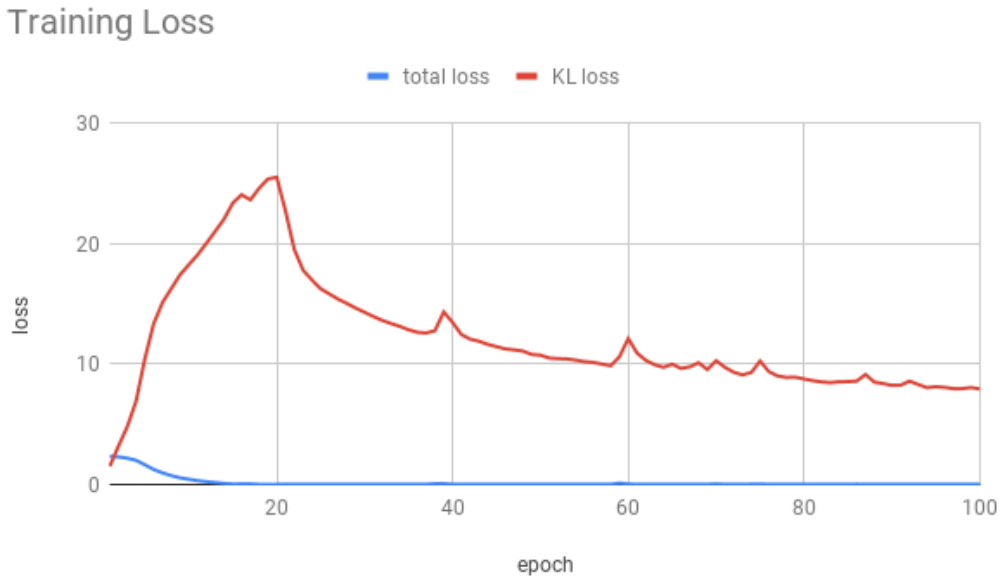

Figure 1: Training Loss

3 Result

I use GRU as the RNN unit, `SGD` as the optimizer, and `CrossEntropyLoss` as the loss function of the neural network whose outputs had not pass the `Softmax` layer. I choose the condition embedding size as $d_c = 8$, the latent size $d_l = 40$, the hidden size $d_h = 512$ and the vocabulary size $d_v = 29$. The learning rate is 0.001, teacher forcing ratio is constant 0.5, and the word drop-out rate is constant 0.1.

Figure 1 shows the total loss and KL loss during training in 100 epochs, and Figure 2 shows the BLEU-4 score during training. The KL weight is set to zero in order to train the sequence-to-sequence first. The KL weight annealing is shown in Figure 3. If the KL weight is too large in the beginning of the training, the KL loss would explode. Therefore, I choose the KL weight to slightly increase over time after the 20-th epoch. As a result, the KL loss, shown in Figure 1, is decreased after the 20-th epoch.

The highest BLEU-4 score is 0.74 and the following are examples of words generated by normal-distributed latent vector:

```
spend, spends, spending, spend
tear, tears, tearing, teare
belt, belts, belting, boles
```

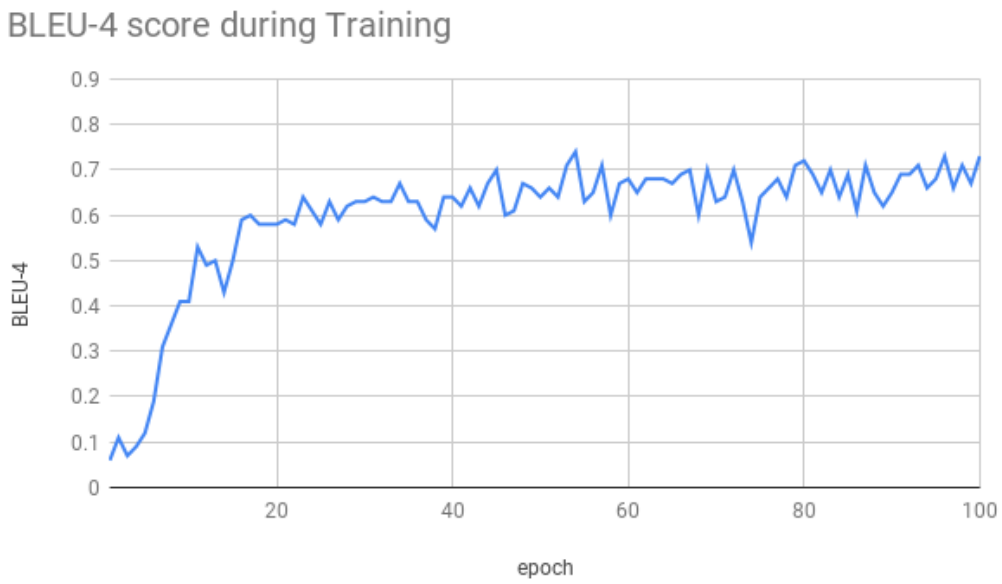

Figure 2: BLEU-4 score during Training

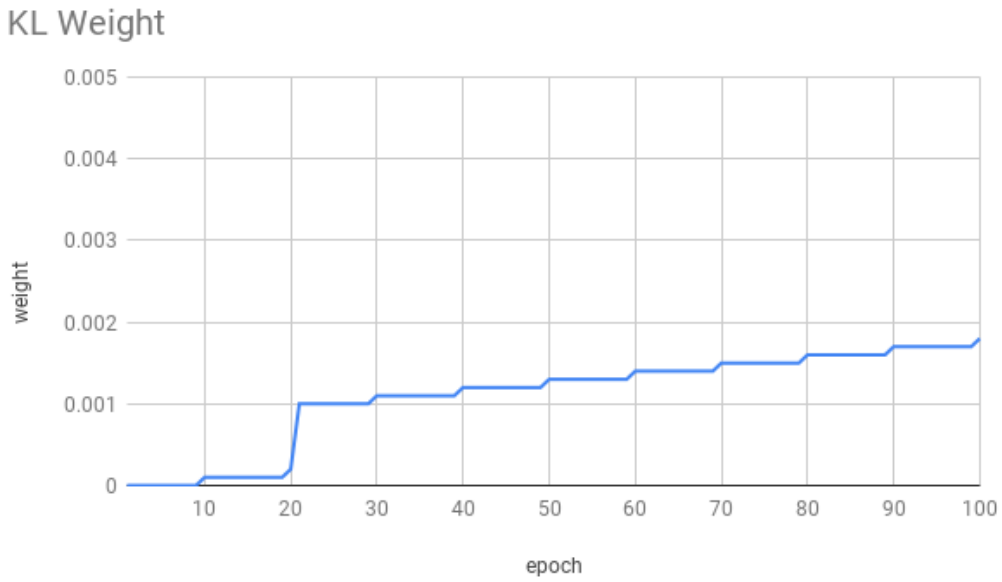

Figure 3: KL Weight

```
hear, hears, hearing, hears
hire, hires, hiring, hired
```

4 Discussion

The KL weight is hard to tune during training, and thus I have saved all states of the model at each epoch which can be restarted with a different KL weight at certain epoch.