Unsupervised Neural Machine Translation with Weight Sharing

Zhen Yang[1,2], **Wei Chen**[1], **Feng Wang**[1,2]*, **Bo Xu**[1]
[1]Institute of Automation, Chinese Academy of Sciences
[2]University of Chinese Academy of Sciences
{yangzhen2014, wei.chen.media, feng.wang, xubo}@ia.ac.cn

A Experiments on the layer number for encoders and decoders

To determine the number of layers for encoders and decoders in our system beforehand, we conduct experiments on English-German translation tasks to test how the amount of layers in encoders and decoders affects the translation performance. We vary the number of layers from 2 to 6 and the results are reported in table 1. We can find that the translation performance achieves substantial improvement with the layer number increasing from 2 to 4. However, with layer number set larger than 4, we get little improvement. To make a trade-off between the translation performance and the computation complexity, we set the layer number as 4 for our encoders and decoders.

layer num	en-de	de-en
2	11.57	14.01
3	12.43	14.99
4	12.86	15.62
5	12.91	15.83
6	12.95	15.79

Table 1: The experiments on the number of layers for encoders and decoders.

B The architecture of the global discriminator

The global discriminator is applied to classify the generated sentences as source language, target language or generated sentences. Following (Yang et al., 2017), we implement the global discriminator based on CNN. Since sentences generated by the generator (the composition of the encoder and decoder) have variable lengths, the CNN padding is used to transform the sentences to sequences with fixed length T, which is the maximum length set for the output of the generator. Given the generated sequences $x_1, \dots, x_T$, we build the matrix $X_{1:T}$ as:

$$X_{1:T} = x_1; x_2; \dots ; x_T \tag{1}$$

where $x_t \in R^k$ is the k-dimensional word embedding and the semicolon is the concatenation operator. For the matrix $X_{1:T}$, a kernel $w_j \in R^{l \times k}$ applies a convolutional operation to a window size of l words to produce a series of feature maps:

$$c_{ji} = \rho(BN(w_j \otimes X_{i:i+l-1} + b)) \tag{2}$$

where $\otimes$ operator is the summation of element-wise production and b is a bias term. ρ is a non-linear activation function which is implemented as ReLu in this paper. To get the final feature with respect to kernel w_j, a max-over-time pooling operation is leveraged over the feature maps:

$$\widetilde{c}_j = max\{c_{j1}, \dots, c_{jT-l+1}\} \tag{3}$$

We use various numbers of kernels with different window sizes to extract different features, which are then concatenated to form the final sentence representation x_c. Finally, we pass x_c through a fully connected layer and a softmax layer to generate the probability $p(f_g|x_1, \dots, x_T)$ as:

$$p(f_g|x_1, \dots, x_T) = softmax(V * x_c) \tag{4}$$

where V is the transformation matrix and $f_g \in \{true, generated\}$.

C The training procedure of the global GAN

We apply the global GANs to finetune the whole model. Here, we provide detailed strategies for

[1]Feng Wang is the corresponding author of this paper

training the global GANs. Firstly, we generate the machine-generated source language sentences by using Enc_t and Enc_s to decode the monolingual data in target language. Similarly, we get the generated sentences in target language with Enc_s and Dec_t by decoding source language monolingual data. We simply use the greedy sampling method instead of the beam search method for decoding. Next, we pre-train D_{g1} on the combination of true monolingual data and the generated data in the source language. Similarly, we also pre-train D_{g2} on the combination of true monolingual data and the generated data in the target language. Finally, we jointly train the generators and discriminators. The generators are trained with policy gradient training methods. For the details about the policy gradient training, we refer the reader to (Yang et al., 2017).

D The configurations for the open-source toolkits

We train the word embedding use the following script:

./word2vec -train text -output embedding.txt -cbow 0 -size 512 -window 10 -negative 10 -hs 0 -sample 1e- -threads 50 -binary 0 -min-count 5 -iter 10

After we get the embeddings for both the source and target languages, we use the open-source VecMap [1] to map these embeddings to a shared-latent space with the following scripts:

python3 normalize_embeddings.py unit center -i s_embedding.txt -o s_embedding.normalized.txt

python3 normalize_embeddings.py unit center -i t_embedding.txt -o t_embedding.normalized.txt

python3 map_embeddings.py –orthogonal s_embedding.normalized.txt t_embedding.normalized.txt s_embedding.mapped.txt t_embedding.mapped.txt –numerals –self_learning -v

References

Zhen Yang, Wei Chen, Feng Wang, and Bo Xu. 2017. Improving neural machine translation with conditional sequence generative adversarial nets .

[1] https://github.com/artetxem/vecmap