Progress of Neural Machine Translation with Memory Network after IJCAI2017

Yang Feng

Feb 21, 2016

1 Baseline

1.1 initialization

- initialize the baseline according to the paper Bahdana et al. 2014 (2.21)
- initialize as old expect V_a and $proj\_b$ are zeros (2.28)
- added masks to baseline. (3.6).
  Analysis: To use the build-in mask, we have to encode the source sentence in positive sequence which I have proved that produced a lower BLEU score than the reversed order by several point.
  To do: add the mask vector not to use the seuqnce_len directly in the bidrectional-rnn.

system	test
500-310-old	43.2
500-310-new	37.6
500-310-v_a0-b0	44.5
1000-620-old	44.7
1000-620-new	40.6
500-310-v_a0-b0-mask	44.4

Table 1: The results for different configuration. *old* means all the configuration is set default except the decoder embedding. *new* means all the initializer is set according to the above paper.

1.2 CS-EN

- initialized according to the paper Bahdana et al. 2014 (2.28)
- test on the random selected 1000 train-heldout data set and the dev set (results shown in Table 1.2)

test set	BLEU
train-held	30.4
dev	6.8
dev-processed	7.5

Table 2: The results of CS-EN.

- analyzed the results above and found:1. the system substitute the figures with '0'; 2. too many UNKs (3200) according to the reference (2300)(from 3.6)
- re-evaluated the BLEU score by substituting figures with '0' in the reference set and dropped UNKs in the results. (results shown in Table 1.2 as dev-processed)

2 $\alpha + \gamma +$ multi-task-training

α : the probabilities that each source word is translated into all the target words in the vocabulary according to the attention. Calculated by $p^{\alpha}(e_i|f_j) = \sum_j \alpha_{ij} p(e_i|f_j)$

γ: the probability that each source word is translated into the memory words, but project to the whole target vocabulary

$multi-task$: 1. translation-probability=$p(e_i|f_j) + p^{\alpha}(e_i|f_j) + p^{\gamma}(e_i|f_j)$ 2. align$^{\alpha}$ 3. align$^{\gamma}$

- modified the calculation of translation-probability by normalize(softmax(model-logits) + 0.5 * normalized(p$^{\alpha}$) + 0.5 * normalized(p$^{\gamma}$)), but the loss during training didn't decline. (2.28)
- added encoder-masks, memory-masks and fixed the bug (results shown in Table 2) (3.6)
- found the reason of slow training as I used softmax two times in order to use the lib function softmax_cross_entropy_with_logits(). The solution is to write this function myself so as to do softmax onlly once.

system	BLEU
baseline-mask	44.4
α-γ-mask	44.9