Lexicons and Minimum Risk Training for Neural Machine Translation: NAIST-CMU at WAT2016

Graham Neubig[†*]
[†]Nara Institute of Science and Technology, Japan
[*]Carnegie Mellon University, USA
gneubig@cs.cmu.edu

Abstract

This year, the Nara Institute of Science and Technology (NAIST)/Carnegie Mellon University (CMU) submission to the Japanese-English translation track of the 2016 Workshop on Asian Translation was based on attentional neural machine translation (NMT) models. In addition to the standard NMT model, we make a number of improvements, most notably the use of discrete translation lexicons to improve probability estimates, and the use of minimum risk training to optimize the MT system for BLEU score. As a result, our system achieved the highest translation evaluation scores for the task.

1 Introduction

Neural machine translation (NMT; (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014)), creation of translation models using neural networks, has quickly achieved state-of-the-art results on a number of translation tasks (Luong and Manning, 2015; Sennrich et al., 2016a). In this paper, we describe NMT systems for the Japanese-English scientific paper translation task of the Workshop on Asian Translation (WAT) 2016 (**?**).

The systems are built using attentional neural networks (Bahdanau et al., 2015; Luong et al., 2015), with a number of improvements (§2). In particular we focus on two. First, we follow the recent work of Arthur et al. (2016) in incorporating discrete translation lexicons to improve the probability estimates of the neural translation model (§3). Second, we incorporate minimum-risk training (Shen et al., 2016) to optimize the parameters of the model to improve translation accuracy (§4).

In experiments (§5), we examine the effect of each of these improvements, and find that they both contribute to overall translation accuracy, leading to state-of-the-art results on the Japanese-English translation task.

2 Baseline Neural Machine Translation Model

Our baseline translation model is the attentional model implemented in the lamtram toolkit (Neubig, 2015), which is a combination of the models of Bahdanau et al. (2015) and Luong et al. (2015) that we found to be effective. We describe the model briefly here for completeness, and refer readers to the previous papers for a more complete description.

2.1 Model Structure

Our model creates a model of target sentence $E = e_1^{|E|}$ given source sentence $F = f_1^{|F|}$. These words belong to the source vocabulary V_f, and the target vocabulary V_e respectively. NMT performs this translation by calculating the conditional probability $p_m(e_i|F, e_1^{i-1})$ of the ith target word e_i based on the source F and the preceding target words e_1^{i-1}. This is done by encoding the context $\langle F, e_1^{i-1} \rangle$ as a fixed-width vector $\boldsymbol{\eta}_i$, and calculating the probability as follows:

$$p_m(e_i|F, e_1^{i-1}) = \text{softmax}(W_s \boldsymbol{\eta}_i + \boldsymbol{b}_s), \quad (1)$$

where W_s and $\boldsymbol{b}_s$ are respectively weight matrix and bias vector parameters. The exact variety of the NMT model depends on how we calculate $\boldsymbol{\eta}_i$ used as input, and as mentioned above, in this case we use an attentional model.

First, an *encoder* converts the source sentence F into a matrix R where each column represents a single word in the input sentence as a continuous vector. This representation is generated using a bidirectional encoder

$$\overrightarrow{\boldsymbol{r}}_j = \text{enc}(\text{embed}(f_j), \overrightarrow{\boldsymbol{r}}_{j-1})$$
$$\overleftarrow{\boldsymbol{r}}_j = \text{enc}(\text{embed}(f_j), \overleftarrow{\boldsymbol{r}}_{j+1}).$$

Here the embed$(\cdot)$ function maps the words into a representation (Bengio et al., 2006), and enc$(\cdot)$ is long short term memory (LSTM) neural network (Hochreiter and Schmidhuber, 1997) with forget gates set to one minus the value of the input gate (Greff et al., 2015). For the final word in the sentence, we add a sentence-end symbol to the final state of both of these decoders

$$\overrightarrow{\boldsymbol{r}}_{|F|+1} = \text{enc}(\text{embed}(\langle \text{s} \rangle), \overrightarrow{\boldsymbol{r}}_{|F|})$$
$$\overleftarrow{\boldsymbol{r}}_{|F|+1} = \text{enc}(\text{embed}(\langle \text{s} \rangle), \overleftarrow{\boldsymbol{r}}_1).$$

Finally we concatenate the two vectors $\overrightarrow{\boldsymbol{r}}_j$ and $\overleftarrow{\boldsymbol{r}}_j$ into a bidirectional representation $\boldsymbol{r}_j$

$$\boldsymbol{r}_j = [\overleftarrow{\boldsymbol{r}}_j; \overrightarrow{\boldsymbol{r}}_j].$$

These vectors are further concatenated into the matrix R where the jth column corresponds to $\boldsymbol{r}_j$.

Next, we generate the output one word at a time while referencing this encoded input sentence and tracking progress with a *decoder* LSTM. The decoder's hidden state $\boldsymbol{h}_i$ is a fixed-length continuous vector representing the previous target words e_1^{i-1}, initialized as $\boldsymbol{h}_0 = \boldsymbol{r}_{|F|+1}$. This is used to calculate a context vector $\boldsymbol{c}_i$ that is used to summarize the source attentional context used in choosing target word e_i, and initialized as $\boldsymbol{c}_0 = \boldsymbol{0}$.

First, we update the hidden state to $\boldsymbol{h}_i$ based on the word representation and context vectors from the previous target time step

$$\boldsymbol{h}_i = \text{enc}([\text{embed}(e_{i-1}); \boldsymbol{c}_{i-1}], \boldsymbol{h}_{i-1}). \tag{2}$$

HERE

Based on this $\boldsymbol{h}_i$, we calculate a similarity vector $\boldsymbol{\alpha}_i$, with each element equal to

$$\alpha_{i,j} = \text{sim}(\boldsymbol{h}_i, \boldsymbol{r}_j). \tag{3}$$

sim$(\cdot)$ can be an arbitrary similarity function. In our systems, we test two similarity functions, the dot product (Luong et al., 2015)

$$\text{sim}(\boldsymbol{h}_i, \boldsymbol{r}_j) := \boldsymbol{h}_i^\intercal \boldsymbol{r}_j \tag{4}$$

and the multi-layered perceptron (Bahdanau et al., 2015)

$$\text{sim}(\boldsymbol{h}_i, \boldsymbol{r}_j) := \boldsymbol{w}_{a2}^\intercal \text{tanh}(W_{a1}[\boldsymbol{h}_i; \boldsymbol{r}_j]), \tag{5}$$

where W_{a1} and $\boldsymbol{w}_{a2}$ are the weight matrix and vector of the first and second layers of the MLP respectively.

We then normalize this into an *attention* vector, which weights the amount of focus that we put on each word in the source sentence

$$\boldsymbol{a}_i = \text{softmax}(\boldsymbol{\alpha}_i). \tag{6}$$

This attention vector is then used to weight the encoded representation R to create a context vector $\boldsymbol{c}_i$ for the current time step

$$\boldsymbol{c}_i = R\boldsymbol{a}_i.$$

Finally, we create $\boldsymbol{\eta}_i$ by concatenating the previous hidden state with the context vector, and performing an affine transform

$$\boldsymbol{\eta}_i = W_\eta[\boldsymbol{h}_i; \boldsymbol{c}_i] + b_\eta,$$

Once we have this representation of the current state, we can calculate $p_m(e_i|F, e_1^{i-1})$ according to Equation (1). The next word e_i is chosen according to this probability.

2.2 Parameter Optimization

If we define all the parameters in this model as θ, we can then train the model by minimizing the negative log-likelihood of the training data

$$\hat{\theta} = \operatorname*{argmin}_{\theta} \sum_{\langle F,\, E\rangle} \sum_{i} -\log(p_m(e_i|F, e_1^{i-1}; \theta)).$$

Specifically, we use the ADAM optimizer (Kingma and Ba, 2014), with an initial learning rate of 0.001. Minibatches of 2048 words are created by sorting sentences in descending order of length and grouping sentences sequentially, adding sentences to the minibatch until the next sentence would cause the minibatch size to exceed 2048 words.[1] Gradients are clipped so their norm does not exceed 5.

Training is allowed to run, checking the likelihood of the development set periodically (every 250k sentences processed), and the model that achieves the best likelihood on the development set is saved. Once no improvements on the development set have been observed for 2M training sentences, training is stopped, and re-started using the previously saved model with a halved learning rate 0.0005. Once training converges for a learning rate of 0.0005, the same procedure is performed with a learning rate of 0.00025, resulting in the final model.

2.3 Search

At test time, to find the best-scoring translation, we perform beam search with a beam size of 5. At each step of beam search, the best-scoring hypothesis remaining in the beam that ends with the sentence-end symbol is saved. At the point where the highest-scoring hypothesis in the beam has a probability less than or equal to the best sentence-ending hypothesis, search is terminated and the best sentence-ending hypothesis is output as the translation.

In addition, because NMT models often tend to be biased towards shorter sentences, we add an optional "word penalty" λ, where each hypothesis's probability is multiplied by $e^{\lambda|E'|}$ for comparison with other hypotheses of different lengths. This is equivalent to adding an exponential prior probability on the length of output sentences, and if $\lambda > 0$, then this will encourage the decoder to find longer hypotheses.

3 Incorporating Discrete Lexicons

The first modification that we make to the base model is incorporating discrete lexicons to improve translation probabilities, according to the method of Arthur et al. (2016). The motivation behind this method is twofold:

Handling low-frequency words: Neural machine translation systems tend to have trouble translating low-frequency words (Sutskever et al., 2014), so incorporating translation lexicons with good coverage of content words could improve translation accuracy of these words.

Training speed: Training the alignments needed for discrete lexicons can be done efficiently (Dyer et al., 2013), and by seeding the neural MT system with these efficiently trained alignments it is easier to learn models that achieve good results more quickly.

The model starts with lexical translation probabilities $p_l(e|f)$ for individual words, which have been obtained through traditional word alignment methods. These probabilities must first be converted to a form that can be used together with $p_m(e_i|e_1^{i-1}, F)$. Given input sentence F, we can construct a matrix in which each column corresponds to a word in the input sentence, each row corresponds to a word in the V_E, and the entry corresponds to the appropriate lexical probability:

$$L_F = \begin{bmatrix} p_l(e=1|f_1) & \cdots & p_l(e=1|f_{|F|}) \\ \vdots & \ddots & \vdots \\ p_l(e=|V_e||f_1) & \cdots & p_l(e=|V_e||f_{|F|}) \end{bmatrix}.$$

[1] It should be noted that it is more common to create minibatches with a fixed number of sentences. We use words here because the amount of memory used in processing a minibatch is more closely related to the number of words in the minibatch than the number of sentences, and thus fixing the size of the minibatch based on the number of words leads to more stable memory usage between minibatches.

This matrix can be precomputed during the encoding stage because it only requires information about the source sentence F.

Next we convert this matrix into a predictive probability over the next word: $p_l(e_i|F, e_1^{i-1})$. To do so we use the alignment probability $\boldsymbol{a}$ from Equation (6) to weight each column of the L_F matrix:

$$p_l(e_i|F, e_1^{i-1}) = L_F \boldsymbol{a}_i = \begin{bmatrix} p_l(e=1|f_1) & \cdots & p_{lex}(e=1|f_{|F|}) \\ \vdots & \ddots & \vdots \\ p_l(e=V_e|f_1) & \cdots & p_{lex}(e=V_e|f_{|F|}) \end{bmatrix} \begin{bmatrix} a_{i,1} \\ \vdots \\ a_{i,|F|} \end{bmatrix}.$$

This calculation is similar to the way how attentional models calculate the context vector $\boldsymbol{c}_i$, but over a vector representing the probabilities of the target vocabulary, instead of the distributed representations of the source words.

After calculating the lexicon predictive probability $p_l(e_i|e_1^{i-1}, F)$, next we need to integrate this probability with the NMT model probability $p_m(e_i|e_1^{i-1}, F)$. Specifically, we use $p_l(\cdot)$ to bias the probability distribution calculated by the vanilla NMT model by adding a small constant ϵ to $p_l(\cdot)$, taking the logarithm, and adding this adjusted log probability to the input of the softmax as follows:

$$p_b(e_i|F, e_1^{i-1}) = \text{softmax}(W_s \boldsymbol{\eta}_i + b_s + \log(p_l(e_i|F, e_1^{i-1}) + \epsilon)).$$

We take the logarithm of $p_l(\cdot)$ so that the values will still be in the probability domain after the softmax is calculated, and add the hyper-parameter ϵ to prevent zero probabilities from becoming $-\infty$ after taking the log. We test various values including $\epsilon = \{10^{-4}, 10^{-5}, 10^{-6}\}$ in experiments.

4 Minimum Risk Training

The second improvement that we make to our model is the use of minimum risk training. As mentioned in Section 2.2 our baseline model optimizes the model parameters according to maximize the likelihood of the training data. However, there is a disconnect between the evaluation of our systems using translation accuracy (such as BLEU (Papineni et al., 2002)) and this maximum likelihood objective.

To remove this disconnect, we use the method of Shen et al. (2016) to optimize our systems directly using BLEU score. Specifically, we define the following loss function over the model parameters θ for a single training sentence pair $\langle F, E \rangle$

$$\mathcal{L}_{F,E}(\theta) = \sum_{E'} \text{err}(E, E') P(E'|F; \theta),$$

which is summed over all potential translations E' in the target language. Here err$(\cdot)$ can be an arbitrary error function, which we define as $1 - \text{SBLEU}(E, E')$, where SBLEU$(\cdot)$ is the smoothed BLEU score (BLEU+1) proposed by Lin and Och (2004). As the number of target-language translations E' is infinite, the sum above is intractable, so we approximate the sum by randomly sampling a subset of translations $\mathcal{S}$ according to $P(E|F; \theta)$, then enumerating over this sample:[2]

$$\mathcal{L}_{F,E}(\theta) = \sum_{E' \in \mathcal{S}} \text{err}(E, E') \frac{P(E'|F; \theta)}{\sum_{E'' \in \mathcal{S}} P(E''|F; \theta)}.$$

This objective function is then modified by introducing a scaling factor α, which makes it possible to adjust the smoothness of the distribution being optimized, which in turn results in adjusting the strength with which the model will try to push good translations to have high probabilities.

$$\mathcal{L}_{F,E}(\theta) = \sum_{E' \in \mathcal{S}} \text{err}(E, E') \frac{P(E'|F; \theta)^\alpha}{\sum_{E'' \in \mathcal{S}} P(E''|F; \theta)^\alpha}.$$

In this work, we set $\alpha = 0.005$ following the original paper, and set the number of samples to be 20.

[2]The actual procedure for obtaining a sample consists of calculating the probability of the first word $P(e_1|F)$, sampling the first word from this multinomial, and then repeating for each following word until the end of sentence symbol is sampled.

	Attent	Lex (ϵ)	ML (λ=0.0)			ML (λ=0.8)			MR (λ=0.0)		
			B	R	Rat.	B	R	Rat.	B	R	Rat.
(1)	dot	No	22.9	74.4	89.9	24.7	74.3	100.9	25.7	75.4	97.3
(2)	dot	Yes (10^{-4})	23.0	74.6	91.0	24.5	74.2	100.4	25.3	75.3	99.2
(3)	dot	Yes (10^{-5})	23.8	74.6	91.4	25.1	74.2	100.4	25.9	75.5	98.0
(4)	dot	Yes (10^{-6})	23.7	74.4	92.1	25.3	74.3	99.6	26.2	76.0	98.6
(5)	MLP	Yes (10^{-4})	23.7	75.3	88.5	25.5	75.2	97.9	26.9	76.3	98.8
(6)	MLP	Yes (10^{-5})	23.7	75.1	90.5	25.3	74.8	98.6	26.4	75.9	97.7
(7)	MLP	Yes (10^{-6})	23.9	74.6	89.4	25.8	74.6	99.3	26.3	75.7	97.3
(8)	(2)-(7) Ensemble			-		27.3	75.8	99.8	29.3	77.3	97.9

Table 1: Overall BLEU, RIBES, and length ratio for systems with various types of attention (dot product or multi-layer perceptron), lexicon (yes/no and which value of λ), training algorithm (maximum likelihood or minimum risk), and word penalty value.

5 Experiments

5.1 Experimental Setup

To create data to train the model, we use the top 2M sentences of the ASPEC Japanese-English training corpus (**?**) provided by the task. The Japanese size of the corpus is tokenized using KyTea (Neubig et al., 2011), and the English side is tokenized with the tokenizer provided with the Travatar toolkit (Neubig, 2013). Japanese is further normalized so all full-width roman characters and digits are normalized to half-width. The words are further broken into subword units using joint byte pair encoding (Sennrich et al., 2016b) with 100,000 merge operations.

5.2 Experimental Results

In Figure 1 we show results for various settings regarding attention, the use of lexicons, training criterion, and word penalty. In addition, we calculate the ensemble of 6 models, where the average probability assigned by each of the models is used to determine the probability of the next word at test time.

From the results in the table, we can glean a number of observations.

Use of Lexicons: Comparing (1) with (2-4), we can see that in general, using lexicons tends to provide a benefit, particularly when the ϵ parameter is set to a small value.

Type of Attention: Comparing (2-4) with (5-7) we can see that on average, multi-layer perceptron attention was more effective than using the dot product.

Use of Word Penalties: Comparing the first and second columns of results, there is a large increase in accuracy across the board when using a word penalty, demonstrating that this is an easy way to remedy the length of NMT results.

Minimum Risk Training: Looking at the third column, we can see that there is an additional increase in accuracy from minimum risk training. In addition, we can see that after minimum risk, the model produces hypotheses that are more-or-less appropriate length without using a word penalty, an additional benefit.

Ensemble: As widely reported in previous work, ensembling together multiple models greatly improved performance.

5.3 Manual Evaluation Results

The maximum-likelihood trained ensemble system with a word penalty of 0.8 (the bottom middle system in Table 1) was submitted for manual evaluation. The system was evaluated according to the official WAT "HUMAN" metric (**?**), which consists of pairwise comparisons with a baseline phrase-based system,

where the evaluated system receives +1 for every win, -1 for every tie, 0 for every loss, these values are averaged over all evaluated sentences, then the value is multiplied by 100. This system achieved a manual evaluation score of 47.50, which was slightly higher than other systems participating in the task. In addition, while the full results of the minimum-risk-based ensemble were not ready in time for the manual evaluation stage, a preliminary system ensembling the minimum-risk-trained versions of the first four systems (1)-(4) in Table 1 was also evaluated (its BLEU/RIBES scores were comparable to the fully ensembled ML-trained system), and received a score of 48.25, the best in the task, albeit by a small margin.

6 Conclusion

In this paper, we described the NAIST-CMU system for the Japanese-English task at WAT, which achieved the most accurate results on this language pair. In particular, incorporating discrete translation lexicons and minimum risk training were found to be useful in achieving these results.

Acknowledgments:

This work was supported by JSPS KAKENHI Grants Number 25730136 and 16H05873.

References

Philip Arthur, Graham Neubig, and Satoshi Nakamura. 2016. Incorporating discrete translation lexicons into neural machine translation. In *Proc. EMNLP*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proc. ICLR*.

Yoshua Bengio, Holger Schwenk, Jean-Sébastien Senécal, Fréderic Morin, and Jean-Luc Gauvain. 2006. Neural probabilistic language models. In *Innovations in Machine Learning*, volume 194, pages 137–186.

Chris Dyer, Victor Chahuneau, and Noah A. Smith. 2013. A simple, fast, and effective reparameterization of ibm model 2. In *Proc. NAACL*, pages 644–648.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. 2015. LSTM: A search space odyssey. *CoRR*, abs/1503.04069.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *Proc. EMNLP*, pages 1700–1709.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Chin-Yew Lin and Franz Josef Och. 2004. Orange: a method for evaluating automatic evaluation metrics for machine translation. In *Proc. COLING*, pages 501–507.

Minh-Thang Luong and Christopher D Manning. 2015. Stanford neural machine translation systems for spoken language domains. In *Proc. IWSLT*.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proc. EMNLP*, pages 1412–1421.

Graham Neubig, Yosuke Nakata, and Shinsuke Mori. 2011. Pointwise prediction for robust, adaptable Japanese morphological analysis. In *Proc. ACL*, pages 529–533.

Graham Neubig. 2013. Travatar: A forest-to-string machine translation engine based on tree transducers. In *Proc. ACL Demo Track*, pages 91–96.

Graham Neubig. 2015. lamtram: A toolkit for language and translation modeling using neural networks. http://www.github.com/neubig/lamtram.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proc. ACL*, pages 311–318.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. Edinburgh neural machine translation systems for WMT16. In *Proc. WMT*, pages 371–376.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. Neural machine translation of rare words with subword units. In *Proc. ACL*, pages 1715–1725.

Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. 2016. Minimum risk training for neural machine translation. In *Proc. ACL*, pages 1683–1692.

Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. 2014. Sequence to sequence learning with neural networks. In *Proc. NIPS*, pages 3104–3112.