Interactive Visualization and Manipulation of Attention-based Neural Machine Translation

Jaesong Lee and **Joong-Hwi Shin** and **Jun-Seok Kim**
NAVER Corp.
{jaesong.lee,joonghwi.shin,jun.seok}@navercorp.com

Abstract

While neural machine translation (NMT) provides high-quality translation, it is still hard to interpret and analyze its behavior. We present an interactive interface for visualizing and intervening behavior of NMT, specifically concentrating on the behavior of beam search mechanism and attention component. The tool (1) visualizes search tree and attention and (2) provides interface to adjust search tree and attention weight (manually or automatically) at real-time. We show the tool help users understand NMT in various ways.

1 Introduction

Recent advances in neural machine translation (NMT) (Sutskever et al., 2014) have changed the direction of machine translation community. Compared to traditional phrase-based statistical machine translation (SMT) (Koehn, 2010), NMT provides more accurate and fluent translation results. Companies also have started to adopt NMT for their machine translation service.

However, it is still challenging to analyze translation behavior of NMT. While SMT provides interpretable features (like phrase table), NMT directly learns complex features which are obscure to human. This is especially problematic in the case of wrong translation, since it is even hard to understand why the system generated such sentences.

To help the analysis, we propose a tool for visualizing and intervening NMT behavior, concentrated on beam search decoder and attention. The features can be grouped by two categories:

- Visualizing decoder result, including how decoder assigns probability to each token (word, sub-word, etc.), how beam search maintains and discards intermediate hypotheses, and how attention layer assigns attention weight. This enables detailed observation of decoder behavior.
- Intervening in decoder behavior, including manually expanding hypothesis discarded during search and adjusting attention weight. This helps understanding how the components affect translation quality.

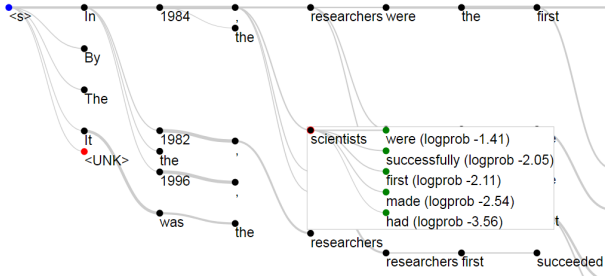

Figure 1: Beam search tree interface. Beam search result is shown as a tree (section 3.1). By hovering mouse over node, its corresponding output candidates can be seen. User may click the candidate to expand node which are discarded during search (section 3.3).

We show the mechanism of visualization (Section 3.1 and 3.2) and manipulation (Section 3.3 and 3.4) and its usefulness with examples.

2 Related Work

There have been various methods proposed for visualizing and intervening neural models for NLP. (Li et al., 2015) provides a concise literature review.

Visualization and manipulation of NMT could be grouped into three parts: RNN (of encoder and decoder), attention (of decoder), and beam search (of decoder).

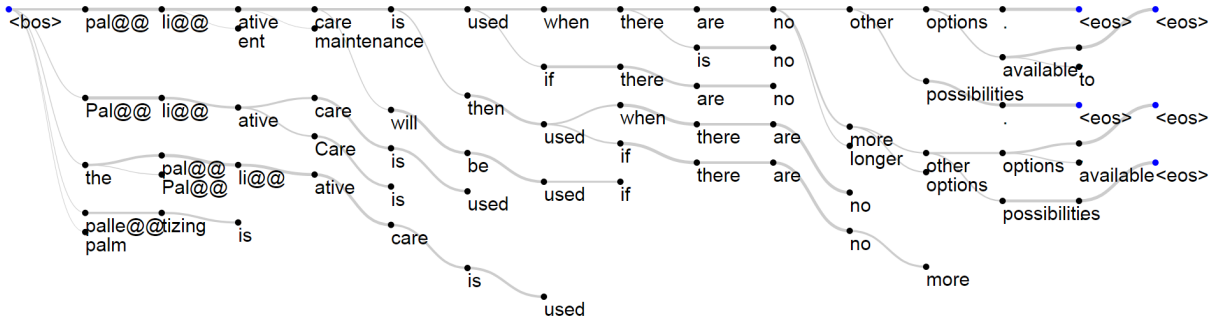

Figure 2: Beam search tree of de-en NMT.

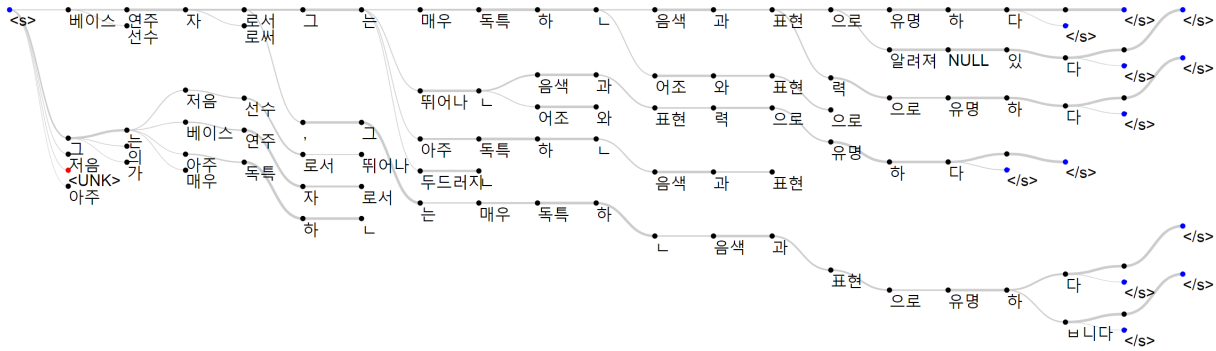

Figure 3: Beam search tree of en-ko NMT. Input sentence is: "As a bass player, he is known for his highly distinctive tone and phrasing."

RNN plays a central role in recognizing source sentences and generating target sentences. Although we here treat RNN as a black-box, there exists various methods to understand RNNs, e.g. by observing intermediate values (Strobelt et al., 2016; Karpathy et al., 2015; Li et al., 2015) or by removing some parts of them (Goh, 2016; Li et al., 2016).

Attention (Bahdanau et al., 2014; Luong et al., 2015) is an important component for improving NMT quality. Since the component behaves like alignment in traditional SMT, it has been proposed to utilize attention during training (Cheng et al., 2015; Tu et al., 2016b) or during decoding (Wu et al., 2016). In this work, we propose a way to manipulate attention and to understand the behavior.

Beam search is known to improve quality of NMT translation output. However, it is also known that larger beam size does not always helps but rather hurts the quality (Tu et al., 2016a). Therefore it is important to understand how beam search affects quality. (Wu et al., 2016; Freitag and Al-Onaizan, 2017) proposed several penalty functions and pruning methods for beam search. We directly visualize beam search result as a tree and manually explore hypotheses discarded by decoder.

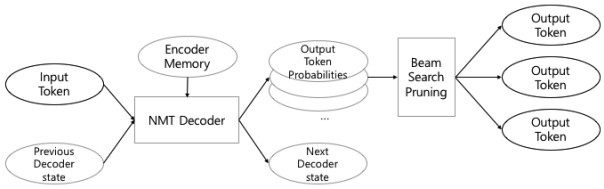

Figure 4: Diagram of NMT decoder step. Search tree visualization (Figure 2) shows input and output tokens as a tree.

3 Interactive Beam Search

We propose an interactive tool for visualizing and manipulating NMT decoder behavior. The system consists of two parts: back-end NMT server and front-end web interface. NMT server is responsible for NMT computation. Web interface is responsible for requesting computation to NMT server and showing results at real time.

For back-end implementation, we use two NMT models. For English-Korean (en-ko), we use a model used in Naver Papago (Lee et al., 2016) service[1] ported to TensorFlow. For German-English

[1] https://papago.naver.com/

	zur	Verhinderung	einer	Inflation	wird	diese	Belohnung	regelmäßig	halbiert	,	wahrscheinlich	vor	Ende	2012	.	<eos>
<bos> → to	0.046	0.500	0.167	0.008	0.019	0.024	0.157	0.011	0.029	0.004	0.002	0.012	0.004	0.001	0.000	0.018
to → prevent	0.010	0.074	0.850	0.014	0.013	0.001	0.005	0.001	0.009	0.002	0.000	0.000	0.000	0.001	0.000	0.019
prevent → inflation	0.002	0.005	0.011	0.709	0.258	0.001	0.005	0.004	0.001	0.000	0.000	0.000	0.000	0.001	0.000	0.003
inflation → ,	0.072	0.006	0.029	0.065	0.103	0.317	0.159	0.005	0.074	0.023	0.003	0.006	0.003	0.003	0.001	0.131
, → that	0.015	0.008	0.003	0.006	0.064	0.029	0.737	0.047	0.049	0.010	0.002	0.006	0.002	0.001	0.000	0.021
that → reward	0.000	0.000	0.000	0.000	0.001	0.002	0.003	0.984	0.002	0.002	0.000	0.000	0.000	0.000	0.000	0.004
reward → is	0.002	0.007	0.004	0.001	0.001	0.468	0.024	0.031	0.357	0.076	0.003	0.005	0.002	0.000	0.000	0.019
is → regularly	0.003	0.003	0.003	0.000	0.000	0.084	0.007	0.002	0.536	0.332	0.004	0.011	0.001	0.000	0.000	0.014
regularly → halved	0.002	0.002	0.002	0.000	0.001	0.028	0.003	0.002	0.022	0.923	0.006	0.001	0.002	0.000	0.000	0.007
halved → ,	0.026	0.033	0.001	0.004	0.003	0.022	0.040	0.002	0.074	0.084	0.549	0.022	0.048	0.009	0.004	0.079
, → probably	0.002	0.001	0.000	0.000	0.000	0.004	0.005	0.001	0.008	0.002	0.026	0.892	0.020	0.002	0.001	0.034
probably → before	0.002	0.002	0.001	0.001	0.001	0.021	0.002	0.000	0.002	0.011	0.004	0.014	0.741	0.089	0.002	0.108
before → the	0.001	0.001	0.001	0.002	0.002	0.001	0.000	0.000	0.000	0.001	0.000	0.006	0.013	0.748	0.155	0.067
the → end	0.002	0.001	0.004	0.002	0.004	0.001	0.000	0.000	0.001	0.001	0.000	0.005	0.021	0.800	0.082	0.076
end → of	0.004	0.001	0.001	0.004	0.001	0.001	0.000	0.000	0.001	0.000	0.000	0.001	0.009	0.071	0.873	0.033
of → 2012	0.001	0.000	0.000	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.055	0.903	0.034
2012 → .	0.139	0.028	0.009	0.028	0.020	0.048	0.013	0.016	0.029	0.014	0.021	0.011	0.008	0.035	0.032	0.550
. → <eos>	0.513	0.081	0.002	0.011	0.011	0.010	0.016	0.003	0.004	0.000	0.004	0.013	0.002	0.003	0.001	0.326

Figure 5: Attention Table. Weight is represented as number and green color.

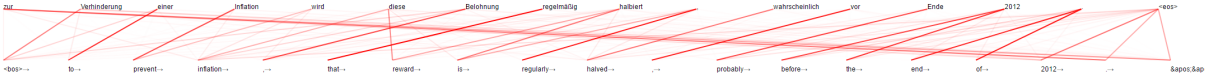

Figure 6: Attention Graph Dialog. Weight is represented as red color.

(de-en), we adopted Nematus[2] and pretrained models provided by (Sennrich et al., 2016). For front-end we implemented JavaScript-based web page with d3.js[3].

3.1 Search Tree Visualization

To understand how beam search decoder selects and discards intermediate hypothesis, we first plot all hypotheses as a tree (Figure 2, 3). For each input token (word or sub-word) and decoder (RNN) state vector, the decoder computes output probability of all possible output token, then beam search routine selects token based on its probability value (Figure 4). We plot each input and output token as tree node, and input-output relation as edge. If a node is mouse-hovered, it shows its next possible tokens with highest probability, including pruned ones (Figure 1). We also visualize output probability of node using edge thickness; thicker edge means higher probability.

3.2 Attention Visualization

We show the attention weight of (partially) generated sentence as table (Figure 5) and as graph (Figure 6). Table interface provides detailed information, and graph interface provides more concise view therefore better for long sentences.

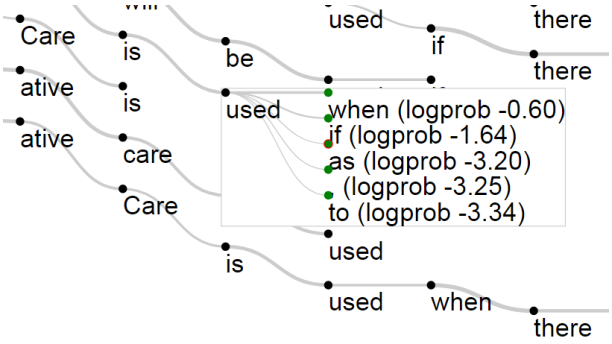

Figure 7: Manual expansion of nodes not explored at Figure 2. For the partial sentence “the Pal@@ li@@ ative Care is”, new subtree “. . . used when there . . . ” are created.

3.3 Search Tree Manipulation

We implemented an interface to manually expand nodes which are discarded during beam search. In search tree visualization (Figure 7) or attention manipulation dialog (Figure 9), a user can click one of output candidate (green node) then the system computes its next outputs and extends the tree. This enables exploration of hypotheses not covered by decoder but worth to analyze.

3.4 Attention Manipulation

We are interested in understanding attention layer of (Bahdanau et al., 2014; Luong et al., 2015), es-

[2] https://github.com/rsennrich/nematus
[3] https://d3js.org/

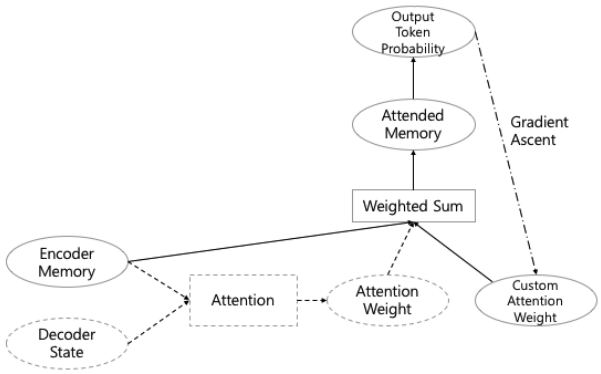

Figure 8: Diagram of attention manipulation mechanism. The dashed components are original component of NMT decoder. Here "attention weight" is replaced by "custom attention weight" which is given by user or computed to maximize probability of output token.

pecially the role and effect of attention weights. To achieve it, we modified NMT decoder to accept arbitrary attention weight instead of what the decoder computes (Figure 8).

3.4.1 Manual Adjustment of Attention Weight

For given memory cells (encoder outputs) $(m_1, \cdots, m_n)$ and decoder internal state h, the attention layer first computes relevance score of memory cell $s_i = f(m_i, h)$ and attention weight $w_i = \text{softmax}(s_1, \cdots, s_n)_i$. Then memory cells are summarized into one fixed vector $(\tilde{m})$ via weighted sum: $\tilde{m} = \sum_i w_i m_i$. The summarized vector is fed to next layer to compute output token probabilities: $p(y_j) = g(\tilde{m}, h)_j$.

We modified the decoder to accept custom weight $w' = (w'_1, \cdots w'_n)$ instead of original ones w, when w' is provided by user. We also implemented front-end interface to adjust custom weight (Figure 9). If user drags circle on the bar, the weights are adjusted and the system computes new output probabilities using the weight. It helps to understand what is encoded in memory cell and how decoder utilizes the attended memory $\tilde{m}$. For example, user may increase or decrease weight of specific memory cell and observe its effect.

Figure 9 shows an illustrative example that how adjusting attention weight could change output probability distribution. When weight of "highly" and "tone" are high, NMT puts high probability to "어조" ("tone of voice"). When weight of "distinctive" is high, NMT recognizes "tone" in current context (musical instrument) and puts high probability to "음색" ("timbre").

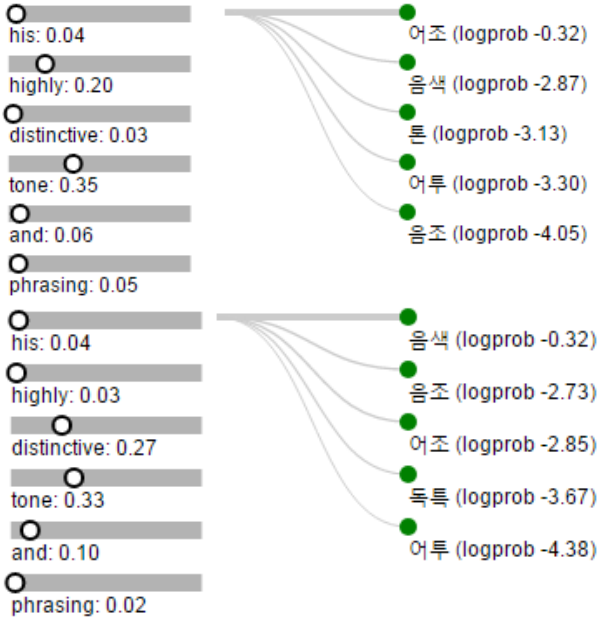

Figure 9: Result of attention manipulation for two output tokens "어조" and "음색".

3.4.2 Automatic Adjustment of Attention Weight

We also implemented a method to find attention weight maximizing output probability of a specific token. For attention weight w and token y, we see this problem as a constrained optimization: maximize $\log p(y|w, \cdots)$ s.t. $w_i \geq 0, \sum_i w_i = 1$. Since the toolkits we use (TensorFlow[4] and Theano[5]) provide unconstrained gradient descent optimizer, we cast the original problem to unconstrained optimization: instead of weight w, we optimize unnormalized score s before softmax, initialized as $s_i = \log w_i$. The method can be used to optimize weight for specific time step (Figure 9) or for whole sentence (Figure 10).

For English-Korean, this technique is particularly useful because the original attention weight is sometimes hard to interpret. Due to ordering differences between two languages, en-ko NMTs tend to generate diverse sentences and they have very different orderings among each other. In Figure 3, input sentence is "As a *bass* player, *he* is ...". NMT puts high probability to output sentences starting with either "베이스" ("bass")

[4] https://www.tensorflow.org/
[5] http://deeplearning.net/software/theano/

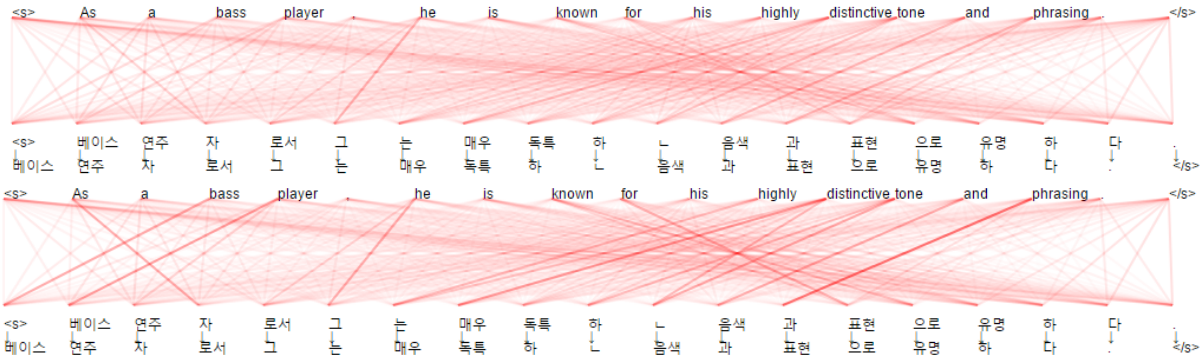

Figure 10: Two attention graphs of en-ko NMT. The first one shows attention weights from NMT. The second one shows attention weights adjusted to maximize target sentence. It reveals clearer and more interpretable relation than original attention.

or "그" ("he"), since both are valid. Therefore, corresponding source words have high attention weights (0.14 for "bass" and 0.07 for "he"). Since output token is chosen after attention, the attention weights do not necessarily look like alignment between source and output sentences, but rather look like a *mixture* of alignments of possible output sentences.

Once output token is chosen, we can find new attention weight which increases probability of output token, which would be more interpretable than the original weight. An example of such adjustment is shown at Figure 10.

4 Conclusion

We propose a web-based interface for visualizing, investigating and understanding neural machine translation (NMT). The tool provides several methods to understand beam search and attention mechanism in an interactive way, by visualizing search tree and attention, expanding search tree manually, and changing attention weight either manually or automatically. We show the visualization and manipulation helps understanding NMT behavior.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations*.

Yong Cheng, Shiqi Shen, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. 2015. Agreement-based joint training for bidirectional attention-based neural machine translation. *CoRR*, abs/1512.04650.

Markus Freitag and Yaser Al-Onaizan. 2017. Beam search strategies for neural machine translation. *CoRR*, abs/1702.01806.

Gabriel Goh. 2016. Decoding the thought vector.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2015. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*.

Philipp Koehn. 2010. *Statistical Machine Translation*, 1st edition. Cambridge University Press, New York, NY, USA.

Hyoung-Gyu Lee, Jun-Seok Kim, Joong-Hwi Shin, Jaesong Lee, Ying-Xiu Quan, and Young-Seob Jeong. 2016. papago: A machine translation service with word sense disambiguation and currency conversion. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations*, pages 185–188, Osaka, Japan. The COLING 2016 Organizing Committee.

Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. 2015. Visualizing and understanding neural models in nlp. *arXiv preprint arXiv:1506.01066*.

Jiwei Li, Will Monroe, and Dan Jurafsky. 2016. Understanding neural networks through representation erasure. *CoRR*, abs/1612.08220.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Edinburgh neural machine translation systems for wmt 16. In *Proceedings of the First*

Conference on Machine Translation, pages 371–376, Berlin, Germany. Association for Computational Linguistics.

Hendrik Strobelt, Sebastian Gehrmann, Bernd Huber, Hanspeter Pfister, and Alexander M. Rush. 2016. Visual analysis of hidden state dynamics in recurrent neural networks. *CoRR*, abs/1606.07461.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Zhaopeng Tu, Yang Liu, Lifeng Shang, Xiaohua Liu, and Hang Li. 2016a. Neural machine translation with reconstruction. *CoRR*, abs/1611.01874.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016b. Coverage-based neural machine translation. *CoRR*, abs/1601.04811.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.