Why Neural Translations are the Right Length

Xing Shi[1], **Kevin Knight**[1], **and Deniz Yuret**[2]
[1]Information Sciences Institute & Computer Science Department
University of Southern California
{xingshi,knight}@isi.edu
[2]Computer Engineering, Koç University
dyuret@ku.edu.tr

Abstract

We investigate how neural, encoder-decoder translation systems output target strings of appropriate lengths, finding that a collection of hidden units learns to explicitly implement this functionality.

1 Introduction

The neural encoder-decoder framework for machine translation (Neco and Forcada, 1997; Castaño and Casacuberta, 1997; Sutskever et al., 2014; Bahdanau et al., 2014; Luong et al., 2015) provides new tools for addressing the field's difficult challenges. In this framework (Figure 1), we use a recurrent neural network (*encoder*) to convert a source sentence into a dense, fixed-length vector. We then use another recurrent network (*decoder*) to convert that vector into a target sentence. In this paper, we train long short-term memory (LSTM) neural units (Hochreiter and Schmidhuber, 1997) trained with back-propagation through time (Werbos, 1990).

A remarkable feature of this simple neural MT (NMT) model is that it produces translations of the right length. When we evaluate the system on previously unseen test data, using BLEU (Papineni et al., 2002), we consistently find the length ratio between MT outputs and human references translations to be very close to 1.0. Thus, no brevity penalty is incurred. This behavior seems to come for free, without special design.

By contrast, builders of standard statistical MT (SMT) systems must work hard to ensure correct length. The original mechanism comes from the IBM SMT group, whose famous Models 1-5 included a learned table $\epsilon(y|x)$, with x and y being the lengths of source and target sentences (Brown et al., 1993). But they did not deploy this table when decoding a foreign sentence f into an English sentence e; it did not participate in incremental scoring and pruning of candidate translations. As a result (Brown et al., 1995):

"However, for a given f, if the goal is to discover the most probable e, then the product P(e) P(f|e) is too small for long English strings as compared with short ones. As a result, short English strings are improperly favored over longer English strings. This tendency is counteracted in part by the following modification: Replace P(f|e) with $c^{length(e)} \cdot P(f|e)$ *for some empirically chosen constant c. This modification is treatment of the symptom rather than treatment of the disease itself, but it offers some temporary relief. The cure lies in better modeling."*

More temporary relief came from Minimum Error-Rate Training (MERT) (Och, 2003), which automatically sets c to maximize BLEU score. MERT also sets weights for the language model P(e), translation model P(f|e), and other features. The length feature combines so sensitively with other features that MERT frequently returns to it as it revises one weight at a time.

NMT's ability to correctly model length is remarkable for these reasons:

- SMT relies on maximum BLEU training to obtain a length ratio that is prized by BLEU, while NMT obtains the same result through generic maximum likelihood training.
- Standard SMT models explicitly "cross off"

Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 2278–2282,
Austin, Texas, November 1-5, 2016.

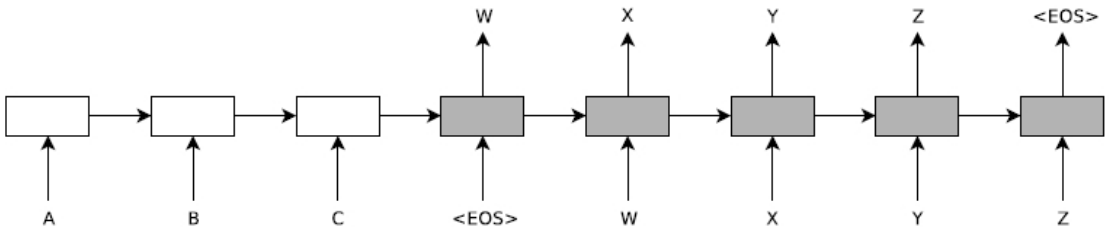

Figure 1: The encoder-decoder framework for neural machine translation (NMT) (Sutskever et al., 2014). Here, a source sentence C B A (fed in reverse as A B C) is translated into a target sentence W X Y Z. At each step, an evolving real-valued vector summarizes the state of the encoder (left half) and decoder (right half).

source words and phrases as they are translated, so it is clear when an SMT decoder has finished translating a sentence. NMT systems lack this explicit mechanism.

- SMT decoding involves heavy search, so if one MT output path delivers an infelicitous ending, another path can be used. NMT decoding explores far fewer hypotheses, using a tight beam without recombination.

In this paper, we investigate how length regulation works in NMT.

2 A Toy Problem for Neural MT

We start with a simple problem in which source strings are composed of symbols *a* and *b*. The goal of the translator is simply to copy those strings. Training cases look like this:

a a a b b a <EOS>	→ *a a a b b a* <EOS>
b b a <EOS>	→ *b b a* <EOS>
a b a b a b a a <EOS>	→ *a b a b a b a a* <EOS>
b b a b b a b b a <EOS>	→ *b b a b b a b b a* <EOS>

The encoder must summarize the content of any source string into a fixed-length vector, so that the decoder can then reconstruct it.[1] With 4 hidden LSTM units, our NMT system can learn to solve this problem after being trained on 2500 randomly chosen strings of lengths up to 9.[2] [3]

To understand how the learned system works, we encode different strings and record the resulting LSTM cell values. Because our LSTM has four hidden units, each string winds up at some point in four-dimensional space. We plot the first two dimensions (unit_1 and unit_2) in the left part of Figure 2, and we plot the other two dimensions (unit_3 and unit_4) in the right part. There is no dimension reduction in these plots. Here is what we learn:

- unit_1 records the approximate length of the string. Encoding a string of length 7 may generate a value of -6.99 for unit_1.
- unit_2 records the number of *b*'s minus the number of *a*'s, thus assigning a more positive value to *b*-heavy strings. It also includes a +1 bonus if the string ends with *a*.
- unit_3 records a prefix of the string. If its value is less than 1.0, the string starts with *b*. Otherwise, it records the number of leading *a*'s.
- unit_4 has a more diffuse function. If its value is positive, then the string consists of all *b*'s (with a possible final *a*). Otherwise, its value correlates with both negative length and the preponderance of *b*'s.

For our purposes, unit_1 is the interesting one. Figure 3 shows the progression of "*a b a b b b*" as it gets encoded (top figure), then decoded (bottom two figures). During encoding, the value of unit_1 decreases by approximately 1.0 each time a letter is read. During decoding, its value increases each time a letter is written. When it reaches zero, it signals the decoder to output <EOS>.

The behavior of unit_1 shows that the translator incorporates explicit length regulation. It also explains two interesting phenomena:

- When asked to transduce previously-unseen strings up to length 14, the system occasionally makes a mistake, mixing up an *a* or *b*. However, the output length is never wrong.[4]

[1] We follow Sutskever et al. (2014) in feeding the input string backwards to the encoder.

[2] Additional training details: 100 epochs, 100 minibatch size, 0.7 learning rate, 1.0 gradient clipping threshold.

[3] We use the toolkit: https://github.com/isi-nlp/Zoph_RNN

[4] Machine translation researchers have also noticed that

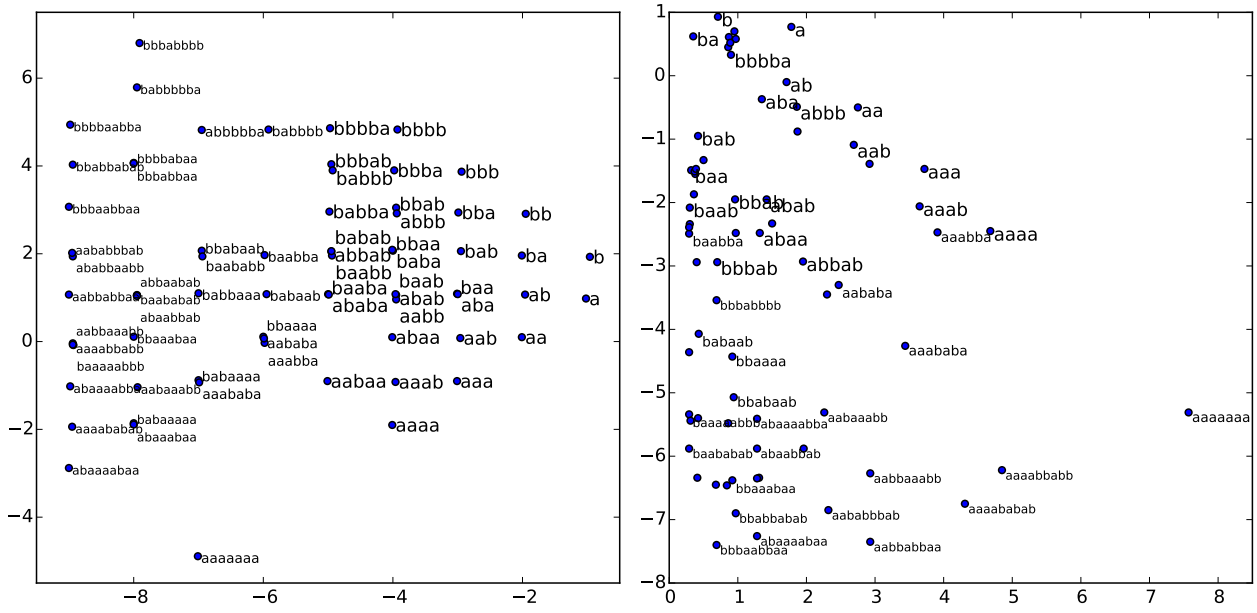

Figure 2: After learning, the recurrent network can convert any string of *a*'s and *b*'s into a 4-dimensional vector. The left plot shows the encoded strings in dimensions described by the cell states of LSTM unit_1 (x-axis) and unit_2 (y-axis). unit_1 learns to record the length of the string, while unit_2 records whether there are more *b*'s than *a*'s, with a +1 bonus for strings that end in *a*. The right plot shows the cell states of LSTM unit_3 (x-axis) and unit_4 (y-axis). unit_3 records how many *a*'s the string begins with, while unit_4 correlates with both length and the preponderance of *b*'s. Some text labels are omitted for clarity.

- When we ask the system to transduce very long strings, beyond what it has been trained on, its output length may be slightly off. For example, it transduces a string of 28 *b*'s into a string of 27 *b*'s. This is because unit_1 is not incremented and decremented by exactly 1.0.

3 Full-Scale Neural Machine Translation

Next we turn to full-scale NMT. We train on data from the WMT 2014 English-to-French task, consisting of 12,075,604 sentence pairs, with 303,873,236 tokens on the English side, and 348,196,030 on the French side. We use 1000 hidden LSTM units. We also use *two layers* of LSTM units between source and target.[5]

After the LSTM encoder-decoder is trained, we send test-set English strings through the encoder portion. Every time a word token is consumed, we record the LSTM cell values and the length of the string so far. Over 143,379 token observations, we investigate how the LSTM encoder tracks length.

Top 10 units by ...	1st layer	2nd layer
Individual R^2	0.868	0.947
Greedy addition	0.968	0.957
Beam search	0.969	0.958

Table 1: R^2 values showing how differently-chosen sets of 10 LSTM hidden units correlate with length in the NMT encoder.

With 1000 hidden units, it is difficult to build and inspect a heat map analogous to Figure 3. Instead, we seek to predict string length from the cell values, using a weighted, linear combination of the 1000 LSTM cell values. We use the least-squares method to find the best predictive weights, with resulting R^2 values of 0.990 (for the first layer, closer to source text) and 0.981 (second layer). So the entire network records length very accurately.

However, unlike in the toy problem, no single unit tracks length perfectly. The best unit in the second layer is unit_{109}, which correlates with R^2=0.894.

We therefore employ three mechanisms to locate

when the translation is completely wrong, the length is still correct (anonymous).

[5]Additional training details: 8 epochs, 128 minibatch size, 0.35 learning rate, 5.0 gradient clipping threshold.

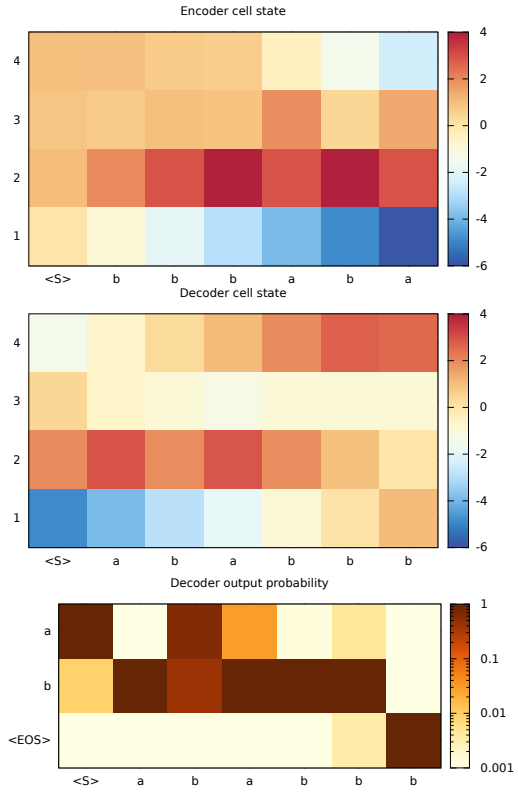

Figure 3: The progression of LSTM state as the recurrent network encodes the string "*a b a b b b*". Columns show the inputs over time and rows show the outputs. Red color indicates positive values, and blue color indicates negative. The value of unit$_1$ decreases during the encoding phase (top figure) and increases during the decoding phase (middle figure). The bottom figure shows the decoder's probability of ending the target string (<EOS>).

k	Best subset of LSTM's 1000 units	R^2
1	109	0.894
2	334, 109	0.936
3	334, 442, 109	0.942
4	334, 442, 109, 53	0.947
5	334, 442, 109, 53, 46	0.951
6	334, 442, 109, 53, 46, 928	0.953
7	334, 442, 109, 53, 46, 433, 663	0.955

Table 2: Sets of k units chosen by beam search to optimally track length in the NMT encoder. These units are from the LSTM's second layer.

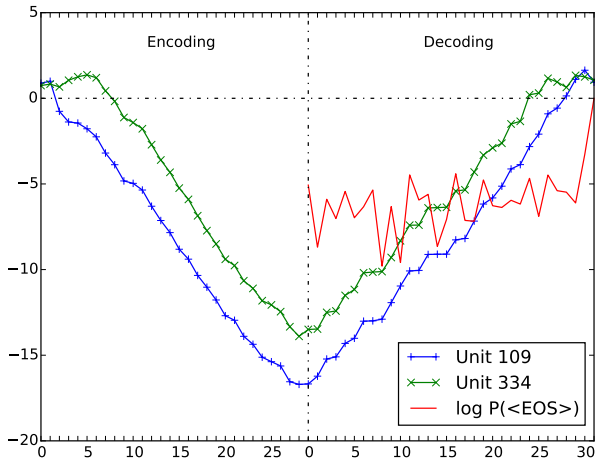

Figure 4: Action of translation unit$_{109}$ and unit$_{334}$ during the encoding and decoding of a sample sentence. Also shown is the softmax log-prob of output <EOS>.

a subset of units responsible for tracking length. We select the top k units according to: (1) individual R^2 scores, (2) greedy search, which repeatedly adds the unit which maximizes the set's R^2 value, and (3) beam search. Table 1 shows different subsets we obtain. These are quite predictive of length. Table 2 shows how R^2 increases as beam search augments the subset of units.

4 Mechanisms for Decoding

For the toy problem, Figure 3 (middle part) shows how the cell value of unit$_1$ moves back to zero as the target string is built up. It also shows (lower part) how the probability of target word <EOS> shoots up once the correct target length has been achieved.

MT decoding is trickier, because source and target strings are not necessarily the same length, and

target length depends on the words chosen. Figure 4 shows the action of unit_{109} and unit_{334} for a sample sentence. They behave similarly on this sentence, but not identically. These two units do not form a simple switch that controls length—rather, they are high-level features computed from lower/previous states that contribute quantitatively to the decision to end the sentence.

Figure 4 also shows the log P(<EOS>) curve, where we note that the probability of outputting <EOS> rises sharply (from 10^{-8} to 10^{-4} to 0.998), rather than gradually.

5 Conclusion

We determine how target length is regulated in NMT decoding. In future work, we hope to determine how other parts of the translator work, especially with reference to grammatical structure and transformations.

Acknowledgments

This work was supported by ARL/ARO (W911NF-10-1-0533), DARPA (HR0011-15-C-0115), and the Scientific and Technological Research Council of Turkey (TÜBİTAK) (grants 114E628 and 215E201).

References

D. Bahdanau, K. Cho, and Y. Bengio. 2014. Neural machine translation by jointly learning to align and translate. In *Proc. ICLR*.

P. Brown, S. della Pietra, V. della Pietra, and R. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

P. F. Brown, J. Cocke, S. della Pietra, V. della Pietra, F. Jelinek, J. C. Lai, and R. L. Mercer. 1995. Method and system for natural language translation. US Patent 5,477,451.

M. A. Castaño and F. Casacuberta. 1997. A connectionist approach to machine translation. In *EUROSPEECH*.

S. Hochreiter and J. Schmidhuber. 1997. Lstm can solve hard long time lag problems. *Advances in neural information processing systems*, pages 473–479.

M. Luong, H. Pham, and C. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proc. EMNLP*.

R. Neco and M. Forcada. 1997. Asynchronous translations with recurrent neural nets. In *International Conf. on Neural Networks*, volume 4, pages 2535–2540.

F. J. Och. 2003. Minimum error rate training in statistical machine translation. In *Proc. ACL*.

K. Papineni, S. Roukos, T. Ward, and W. Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proc. ACL*.

I. Sutskever, O. Vinyals, and Q. V. Le. 2014. Sequence to sequence learning with neural networks. In *Proc. NIPS*.

P. J. Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.