Does String-Based Neural MT Learn Source Syntax?

Xing Shi, Inkit Padhi, and Kevin Knight
Information Sciences Institute & Computer Science Department
University of Southern California
xingshi@isi.edu, ipadhi@usc.edu, knight@isi.edu

Abstract

We investigate whether a neural, encoder-decoder translation system learns syntactic information on the source side as a by-product of training. We propose two methods to detect whether the encoder has learned local and global source syntax. A fine-grained analysis of the syntactic structure learned by the encoder reveals which kinds of syntax are learned and which are missing.

1 Introduction

The sequence to sequence model (*seq2seq*) has been successfully applied to neural machine translation (NMT) (Sutskever et al., 2014; Cho et al., 2014) and can match or surpass MT state-of-art. Non-neural machine translation systems consist chiefly of phrase-based systems (Koehn et al., 2003) and syntax-based systems (Galley et al., 2004; Galley et al., 2006; DeNeefe et al., 2007; Liu et al., 2011; Cowan et al., 2006), the latter of which adds syntactic information to source side (tree-to-string), target side (string-to-tree) or both sides (tree-to-tree). As the seq2seq model first encodes the source sentence into a high-dimensional vector, then decodes into a target sentence, it is hard to understand and interpret what is going on inside such a procedure. Considering the evolution of non-neural translation systems, it is natural to ask:

1. Does the encoder learn syntactic information about the source sentence?
2. What kind of syntactic information is learned, and how much?
3. Is it useful to augment the encoder with additional syntactic information?

In this work, we focus on the first two questions and propose two methods:

- We create various syntactic labels of the source sentence and try to predict these syntactic labels with logistic regression, using the learned sentence encoding vectors (for sentence-level labels) or learned word-by-word hidden vectors (for word-level label). We find that the encoder captures both global and local syntactic information of the source sentence, and different information tends to be stored at different layers.
- We extract the whole constituency tree of source sentence from the NMT encoding vectors using a retrained linearized-tree decoder. A deep analysis on these parse trees indicates that much syntactic information is learned, while various types of syntactic information are still missing.

2 Example

As a simple example, we train an English-French NMT system on 110M tokens of bilingual data (English side). We then take 10K separate English sentences and label their *voice* as active or passive. We use the learned NMT encoder to convert these sentences into 10k corresponding 1000-dimension encoding vectors. We use 9000 sentences to train a logistic regression model to predict voice using the encoding cell states, and test on the other 1000 sentences. We achieve 92.8% accuracy (Table 2), far above the majority class baseline (82.8%). This means that in reducing the source sentence to a

Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1526–1534,
Austin, Texas, November 1-5, 2016. ©2016 Association for Computational Linguistics

Model	Accuracy
Majority Class	82.8
English to French (E2F)	92.8
English to English (E2E)	82.7

Table 1: Voice (active/passive) prediction accuracy using the encoding vector of an NMT system. The majority class baseline always chooses active.

fixed-length vector, the NMT system has decided to store the voice of English sentences in an easily accessible way.

When we carry out the same experiment on an English-English (auto-encoder) system, we find that English voice information is no longer easily accessed from the encoding vector. We can only predict it with 82.7% accuracy, no better than chance. Thus, in learning to reproduce input English sentences, the seq2seq model decides to use the fixed-length encoding vector for other purposes.

3 Related work

Interpreting Recurrent Neural Networks. The most popular method to visualize high-dimensional vectors, such as word embeddings, is to project them into two-dimensional space using *t-SNE* (van der Maaten and Hinton, 2008). Very few works try to interpret recurrent neural networks in NLP. Karpathy et al. (2016) use a character-level LSTM language model as a test-bed and find several activation cells that track long-distance dependencies, such as line lengths and quotes. They also conduct an error analysis of the predictions. Li et al. (2016) explore the syntactic behavior of an RNN-based sentiment analyzer, including the compositionality of negation, intensification, and concessive clauses, by plotting a 60-dimensional heat map of hidden unit values. They also introduce a first-order derivative based method to measure each unit's contribution to the final decision.

Verifying syntactic/semantic properties. Several works try to build a good distributional representation of sentences or paragraph (Socher et al., 2013; Kalchbrenner et al., 2014; Kim, 2014; Zhao et al., 2015; Le and Mikolov, 2014; Kiros et al., 2015). They implicitly verify the claimed syntactic/semantic properties of learned representations by applying them to downstream classification tasks such as sentiment analysis, sentence classification, semantic relatedness, paraphrase detection, image-sentence ranking, question-type classification, etc.

Novel contributions of our work include:

- We locate a subset of activation cells that are responsible for certain syntactic labels. We explore the concentration and layer distribution of different syntactic labels.
- We extract whole parse trees from NMT encoding vectors in order to analyze syntactic properties directly and thoroughly.
- Our methods are suitable for large scale models. The models in this work are 2-layer 1000-dimensional LSTM seq2seq models.

4 Datasets and models

We train two NMT models, English-French (E2F) and English-German (E2G). To answer whether these translation models' encoders to learn store syntactic information, and how much, we employ two benchmark models:

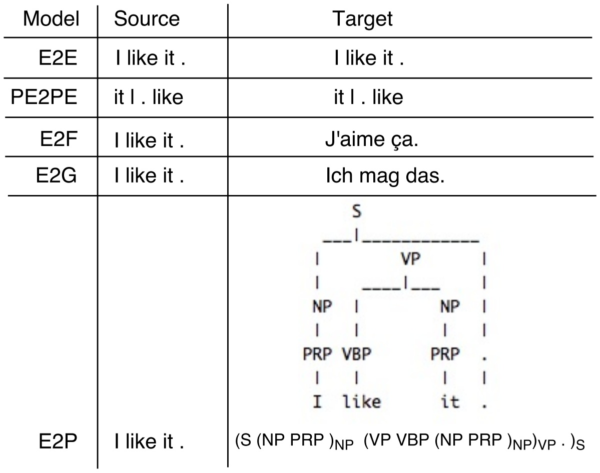
- An upper-bound model, in which the encoder learns quite a lot of syntactic information. For the upper bound, we train a neural parser that learns to "translate" an English sentence to its linearized constitutional tree (E2P), following Vinyals et al. (2015).
- An lower-bound model, in which the encoder learns much less syntactic information. For the lower bound, we train two sentence auto-encoders: one translates an English sentence to itself (E2E), while the other translates a permuted English sentence to itself (PE2PE). We already had an indication above (Section 2) that a copying model does not necessarily need to remember a sentence's syntactic structure.

Figure 1 shows sample inputs and outputs of the E2E, PE2PE, E2F, E2G, and E2P models.

We use English-French and English-German data from WMT2014 (Bojar et al., 2014). We take 4M English sentences from the English-German data to train E2E and PE2PE. For the neural parser (E2P), we construct the training corpus following the recipe of Vinyals et al. (2015). We collect 162K training sentences from publicly available treebanks, including Sections 0-22 of the Wall Street Journal Penn Treebank (Marcus et al., 1993), Ontonotes version 5

Model	Target Language	Input vocabulary size	Output vocabulary size	Train/Dev/Test Corpora Sizes (sentence pairs)	BLEU
E2E	English	200K	40K	4M/3000/2737	89.11
PE2PE	Permuted English	200K	40K	4M/3000/2737	88.84
E2F	French	200K	40K	4M/6003/3003	24.59
E2G	German	200K	40K	4M/3000/2737	12.60
E2P	Linearized constituency tree	200K	121	8162K/1700/2416	n/a

Table 2: Model settings and test-set BLEU-n4r1 scores (Papineni et al., 2002).



Model	Source	Target
E2E	I like it .	I like it .
PE2PE	it I . like	it I . like
E2F	I like it .	J'aime ça.
E2G	I like it .	Ich mag das.
E2P	I like it .	$(S\ (NP\ PRP\ )_{NP}\ \ (VP\ VBP\ (NP\ PRP\ )_{NP})_{VP}\ .\ )_S$

S
VP
NP
NP
PRP VBP PRP .
I like it .

Figure 1: Sample inputs and outputs of the E2E, PE2PE, E2F, E2G, and E2P models.

(Pradhan and Xue, 2009) and the English Web Treebank (Petrov and McDonald, 2012). In addition to these gold treebanks, we take 4M English sentences from English-German data and 4M English sentences from English-French data, and we parse these 8M sentences with the Charniak-Johnson parser[1] (Charniak and Johnson, 2005). We call these 8,162K pairs the CJ corpus. We use WSJ Section 22 as our development set and section 23 as the test set, where we obtain an F1-score of 89.6, competitive with the previously-published 90.5 (Table 4).

Model Architecture. For all experiments[2], we use a two-layer encoder-decoder with long short-term memory (LSTM) units (Hochreiter and Schmidhuber, 1997). We use a minibatch of 128, a hidden state size of 1000, and a dropout rate of 0.2.

Parser	WSJ 23 F1-score	# valid trees (out of 2416)
CJ Parser	92.1	2416
E2P	89.6	2362
(Vinyals et al., 2015)	90.5	unk

Table 3: Labeled F1-scores of different parsers on WSJ Section 23. The F1-score is calculated on valid trees only.

For auto-encoders and translation models, we train 8 epochs. The learning rate is initially set as 0.35 and starts to halve after 6 epochs. For E2P model, we train 15 epochs. The learning rate is initialized as 0.35 and starts to decay by 0.7 once the perplexity on a development set starts to increase. All parameters are re-scaled when the global norm is larger than 5. All models are non-attentional, because we want the encoding vector to summarize the whole source sentence. Table 4 shows the settings of each model and reports the BLEU scores.

5 Syntactic Label Prediction

5.1 Experimental Setup

In this section, we test whether different seq2seq systems learn to encode syntactic information about the source (English) sentence.

With 1000 hidden states, it is impractical to investigate each unit one by one or draw a heat map of the whole vector. Instead, we use the hidden states to predict syntactic labels of source sentences via logistic regression. For multi-class prediction, we use a one-vs-rest mechanism. Furthermore, to identify a subset of units responsible for certain syntactic labels, we use the recursive feature elimination (RFE) strategy: the logistic regression is first trained using

[1] The CJ parser is here https://github.com/BLLIP/bllip-parser and we used the pretrained model ”WSJ+Gigaword-v2”.

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

Label	Train	Test	Number of classes	Most frequent label
Voice	9000	1000	2	Active
Tense	9000	1000	2	Non-past
TSS	9000	1000	20	NP-VP
POS	87366	9317	45	NN
SPC	81292	8706	24	NP

Table 4: Corpus statistics for five syntactic labels.

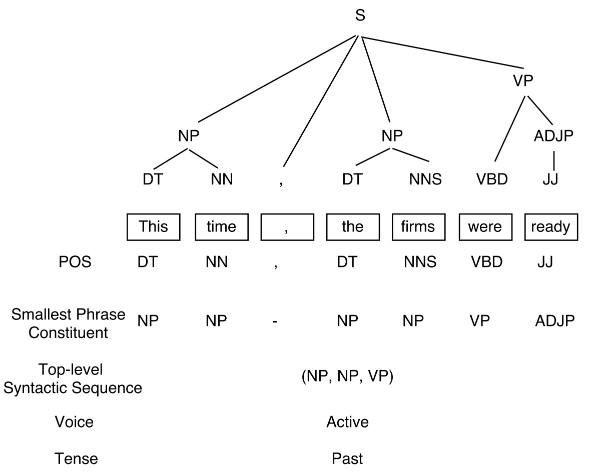


Figure 2: The five syntactic labels for sentence “This time , the firms were ready”.

all 1000 hidden states, after which we recursively prune those units whose weights’ absolute values are smallest.

We extract three sentence-level syntactic labels:

1. Voice: active or passive.
2. Tense: past or non-past.
3. TSS: Top level syntactic sequence of the constituent tree. We use the most frequent 19 sequences (“NP-VP”, “PP-NP-VP”, etc.) and label the remainder as “Other”.

and two word-level syntactic labels:

1. POS: Part-of-speech tags for each word.
2. SPC: The smallest phrase constituent that above each word.

Both voice and tense labels are generated using rule-based systems based on the constituent tree of the sentence.

Figure 2 provides examples of our five syntactic labels. When predicting these syntactic labels using corresponding cell states, we split the dataset into training and test sets. Table 4 shows statistics of each labels.

For a source sentence s,

$$s = [w_1, ..., w_i, ..., w_n]$$

the two-layer encoder will generate an array of cell vectors c during encoding,

$$c = [(c_{1,0}, c_{1,1}), ..., (c_{i,0}, c_{i,1}), ..., (c_{n,0}, c_{n,1})]$$

We extract a sentence-level syntactic label L_s, and predict it using the encoding cell states that will be fed into the decoder:

$$L_s = g(c_{n,0}) \text{ or } L_s = g(c_{n,1})$$

where $g(\cdot)$ is the logistic regression.

Similarly, for extracting word-level syntactic labels:

$$L_w = [L_{w1}, ..., L_{wi}, ..., L_{wn}]$$

we predict each label L_{wi} using the cell states immediately after encoding the word w_i:

$$L_{wi} = g(c_{i,0}) \text{ or } L_{Wi} = g(c_{i,1})$$

5.2 Result Analysis

Test-set prediction accuracy is shown in Figure 3. For voice and tense, the prediction accuracy of two auto-encoders is almost same as the accuracy of majority class, indicating that their encoders do not learn to record this information. By contrast, both the neural parser and the NMT systems achieve approximately 95% accuracy. When predicting the top-level syntactic sequence (TSS) of the whole sentence, the Part-of-Speech tags (POS), and smallest phrase constituent (SPC) for each word, all five models achieve an accuracy higher than that of majority class, but there is still a large gap between the accuracy of NMT systems and auto-encoders. These observations indicate that the NMT encoder learns significant sentence-level syntactic information—it can distinguish voice and tense of the source sentence, and it knows the sentence’s structure to some extent. At the word level, the NMT’s encoder also tends to cluster together the words that have similar POS and SPC labels.

Different syntactic information tends to be stored at different layers in the NMT models. For word-level syntactic labels, POS and SPC, the accuracy of the lower layer’s cell states ($C0$) is higher than that of the upper level ($C1$). For the sentence-level

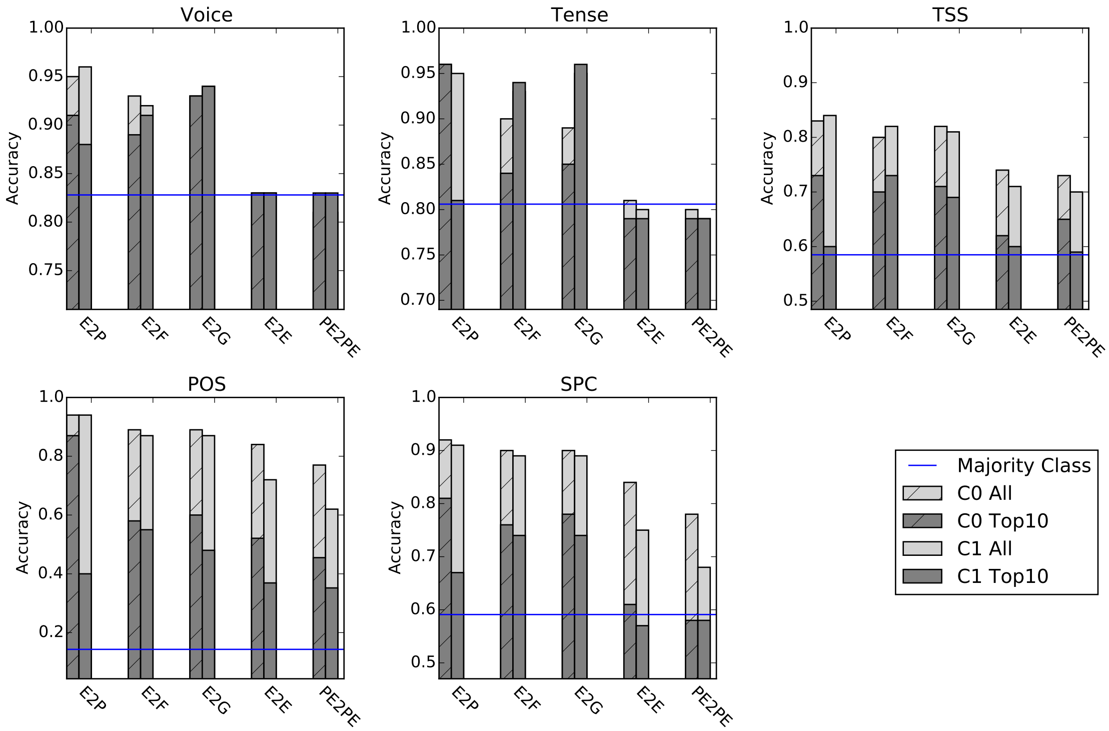


Figure 3: Prediction accuracy of five syntactic labels on test. Each syntactic label is predicted using both the lower-layer cell states (C0) and higher-layer cell states (C1). For each cell state, we predict each syntactic label using all 1000 units (All), as well as the top 10 units (Top10) selected by recursive feature elimination. The horizontal blue line is the majority class accuracy.

labels, especially tense, the accuracy of $C1$ is larger than $C0$. This suggests that the local features are somehow preserved in the lower layer whereas more global, abstract information tends to be stored in the upper layer.

For two-classes labels, such as voice and tense, the accuracy gap between all units and top-10 units is small. For other labels, where we use a one-versus-rest strategy, the gap between all units and top-10 units is large. However, when predicting POS, the gap of neural parser (E2P) on the lower layer ($C0$) is much smaller. This comparison indicates that a small subset of units explicitly takes charge of POS tags in the neural parser, whereas for NMT, the POS info is more distributed and implicit.

There are no large differences between encoders of E2F and E2G regarding syntactic information.

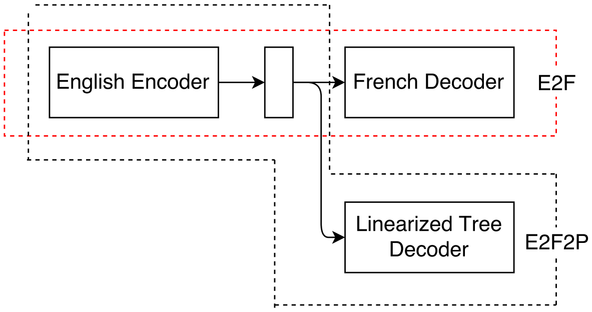


Figure 4: E2F and E2F2P share the same English encoder. When training E2F2P, we only update the parameters of linearized tree decoder, keeping the English encoder's parameters fixed.

6 Extract Syntactic Trees from Encoder

6.1 Experimental Setup

We now turn to whether NMT systems capture deeper syntactic structure as a by-product of learning to translate from English to another language. We do this by predicting full parse trees from the information stored in encoding vectors. Since this is a structured prediction problem, we can no longer use logistic regression. Instead, we extract a constituency parse tree from the encoding vector of a model E2X by using a new neural parser E2X2P with the following steps:

1. Take the E2X encoder as the encoder of the new model E2X2P.
2. Initialize the E2X2P decoder parameters with a uniform distribution.
3. Fine-tune the E2X2P decoder (while keeping its encoder parameters fixed), using the CJ corpus, the same corpus used to train $E2P$.

Figure 4 shows how we construct model E2F2P from model E2F. For fine-tuning, we use the same dropout rate and learning rate updating configuration for E2P as described in Section 4.

6.2 Evaluation

We train four new neural parsers using the encoders of the two auto-encoders and the two NMT models respectively. We use three tools to evaluate and analyze:

1. The EVALB tool[3] to calculate the labeled bracketing F1-score.
2. The zxx package[4] to calculate Tree edit distance (TED) (Zhang and Shasha, 1989).
3. The Berkeley Parser Analyser[5] (Kummerfeld et al., 2012) to analyze parsing error types.

The linearized parse trees generated by these neural parsers are not always well-formed. They can be split into the following categories:

- Malformed trees: The linearized sequence can not be converted back into a tree, due to missing or mismatched brackets.
- Well-formed trees: The sequence can be converted back into a tree. Tree edit distance can be calculated on this category.
  - Wrong length trees: The number of tree leaves does not match the number of source-sentence tokens.
  - Correct length trees: The number of tree leaves *does* match the number of source-sentence tokens.

Before we move to results, we emphasize the following points:

First, compared to the linear classifier used in Section 5, the retrained decoder for predicting a linearized parse tree is a highly non-linear method. The syntactic prediction/parsing performance will increase due to such non-linearity. Thus, we do not make conclusions based only on absolute performance values, but also on a comparison against the designed baseline models. An improvement over the lower bound models indicates that the encoder learns syntactic information, whereas a decline from the upper bound model shows that the encoder loses certain syntactic information.

Second, the NMT's encoder maps a plain English sentence into a high-dimensional vector, and our goal is to test whether the projected vectors form a more syntactically-related manifold in the high-dimensional space. In practice, one could also predict parse structure for the E2E in two steps: (1) use E2E's decoder to recover the original English sentence, and (2) parse that sentence with the CJ parser. But in this way, the manifold structure in the high-dimensional space is destroyed during the mapping.

6.2.1 Result Analysis

Table 5 reports perplexity on training and development sets, the labeled F1-score on WSJ Section 23, and the Tree Edit Distance (TED) of various systems.

Tree Edit Distance (TED) calculates the minimum-cost sequence of node edit operations (delete, insert, rename) between a gold tree and a test tree. When decoding with beam size 10, the four new neural parsers can generate well-formed trees for almost all the 2416 sentences in the WSJ section 23. This makes TED a robust metric to evaluate the overall performance of each parser. Table 5 reports the average TED per sentence. Trees extracted from E2E and PE2PE encoding vectors (via models E2E2P and PE2PE2P, respectively) get TED above 30, whereas the NMT systems get

[3] http://nlp.cs.nyu.edu/evalb/

[4] https://github.com/timtadh/zhang-shasha

[5] https://github.com/jkkummerfeld/berkeley-parser-analyser

Model	Perplexity on Train	Perplexity on WSJ 22	Labeled F1 on WSJ23	# EVALB-trees (out of 2416)	Average TED per sentence	# Well-formed trees (out of 2416)
PE2PE2P	1.83	1.92	46.64	818	34.43	2416
E2E2P	1.69	1.77	59.35	796	31.25	2416
E2G2P	1.39	1.41	80.34	974	17.11	2340
E2F2P	1.36	1.38	79.27	1093	17.77	2415
E2P	1.11	1.18	89.61	2362	11.50	2415

Table 5: Perplexity, labeled F1-score, and Tree Edit Distance (TED) of various systems. Labeled F1-scores are calculated on EVALB-trees only. Tree edit distances are calculated on the well-formed trees only. EVALB-trees are those whose number of leaves match the number of words in the source sentence, and are otherwise accepted by standard Treebank evaluation software.

approximately 17 TED.

Among the well-formed trees, around half have a mismatch between number of leaves and number of tokens in the source sentence. The labeled F1-score is reported over the rest of the sentences only. Though biased, this still reflects the overall performance: we achieve around 80 F1 with NMT encoding vectors, much higher than with the E2E and PE2PE encoding vectors (below 60).

6.2.2 Fine-grained Analysis

Besides answering whether the NMT encoders learn syntactic information, it is interesting to know what kind of syntactic information is extracted and what is not.

As Table 5 shows, different parsers generate different numbers of trees that are acceptable to Treebank evaluation software (“EVALB-trees”), having the correct number of leaves and so forth. We select the intersection set of different models’ EVALB-trees. We get a total of 569 shared EVALB-trees. The average length of the corresponding sentence is 12.54 and the longest sentence has 40 tokens. The average length of all 2416 sentences in WSJ section 23 is 23.46, and the longest is 67. As we do not apply an attention model for these neural parsers, it is difficult to handle longer sentences. While the intersection set may be biased, it allows us to explore how different encoders decide to capture syntax on short sentences.

Table 6 shows the labeled F1-scores and Part-of-Speech tagging accuracy on the intersection set. The NMT encoder extraction achieves around 86 percent tagging accuracy, far beyond that of the autoencoder based parser.

Model	Labeled F1	POS Tagging Accuracy
PE2PE2P	58.67	54.32
E2E2P	70.91	68.03
E2G2P	85.36	85.30
E2F2P	86.62	87.09
E2P	93.76	96.00

Table 6: Labeled F1-scores and POS tagging accuracy on the intersection set of EVALB-trees of different parsers. There are 569 trees in the intersection, and the average length of corresponding English sentence is 12.54.

Besides the tagging accuracy, we also utilize the Berkeley Parser Analyzer (Kummerfeld et al., 2012) to gain a more linguistic understanding of predicted parses. Like TED, the Berkeley Parser Analyzer is based on tree transformation. It repairs the parse tree via a sequence of sub-tree movements, node insertions and deletions. During this process, multiple bracket errors are fixed, and it associates this group of node errors with a linguistically meaningful error type.

The first column of Figure 5 shows the average number of bracket errors per sentence for model E2P on the intersection set. For other models, we report the ratio of each model to model E2P. Kummerfeld et al. (2013) and Kummerfeld et al. (2012) give descriptions of different error types. The NMT-based predicted parses introduce around twice the bracketing errors for the first 10 error types, whereas for “Sense Confusion”, they bring more than 16 times bracket errors. “Sense confusion” is the case where the head word of a phrase receives the wrong POS,

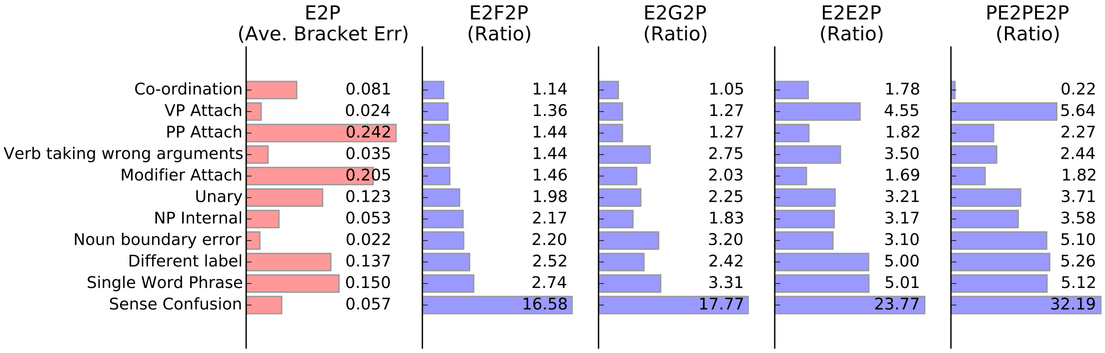


Figure 5: For model $E2P$ (the red bar), we show the average number of bracket errors per sentence due to the top 11 error types. For other models, we show the ratio of each model's average number of bracket errors to that of model $E2P$. Errors analyzed on the intersection set. The table is sorted based on the ratios of the $E2F2P$ model.

resulting in an attachment error. Figure 6 shows an example.

Even though we can predict 86 percent of parts-of-speech correctly from NMT encoding vectors, the other 14 percent introduce quite a few attachment errors. NMT sentence vectors encode a lot of syntax, but they still cannot grasp these subtle details.

7 Conclusion

We investigate whether NMT systems learn source-language syntax as a by-product of training on string pairs. We find that both local and global syntactic information about source sentences is captured by the encoder. Different types of syntax is stored in different layers, with different concentration degrees. We also carry out a fine-grained analysis of the constituency trees extracted from the encoder, highlighting what syntactic information is still missing.

Acknowledgments

This work was supported by ARL/ARO (W911NF-10-1-0533) and DARPA (HR0011-15-C-0115).

References

Ondrej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Matous Machacek, Christof Monz, Pavel Pecina, Matt Post, Herv Saint-Amand, Radu Soricut, and Lucia Specia, editors. 2014. *Proc. Ninth Workshop on Statistical Machine Translation*.

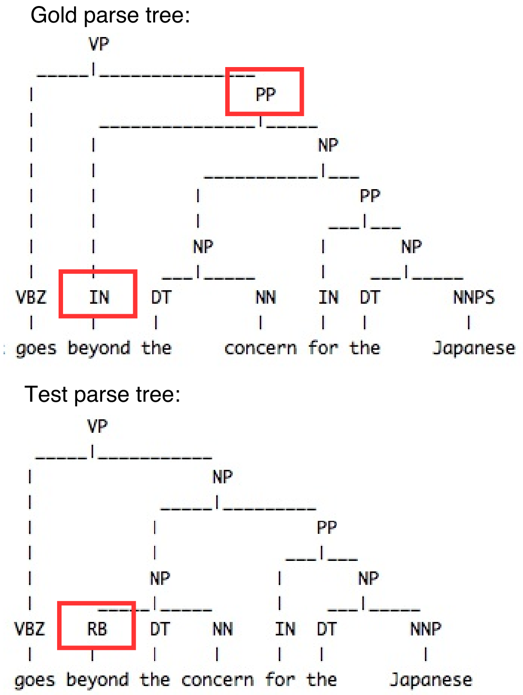


Figure 6: Example of Sense Confusion. The POS tag for word "beyond" is predicted as "RB" instead of "IN", resulting in a missing prepositional phrase.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proc. ACL*.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proc. EMNLP*.

Brooke Cowan, Ivona Kučerová, and Michael Collins. 2006. A discriminative model for tree-to-tree translation. In *Proc. EMNLP*.

Steve DeNeefe, Kevin Knight, Wei Wang, and Daniel Marcu. 2007. What can syntax-based MT learn from phrase-based MT? In *Proc. EMNLP-CoNLL*.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What ' s in a translation rule ? *Information Sciences*, 2004.

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. 2006. Scalable inference and training of context-rich syntactic translation models. In *Proc. ACL*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8).

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. In *Proc. ACL*.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2016. Visualizing and understanding recurrent networks. In *Proc. ICLR*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proc. EMNLP*.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *Proc. NIPS*.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proc. NAACL*.

Jonathan K. Kummerfeld, David Hall, James R. Curran, and Dan Klein. 2012. Parser showdown at the Wall Street Corral: An empirical investigation of error types in parser output. In *Proc. EMNLP-CoNLL*.

Jonathan K. Kummerfeld, Daniel Tse, James R Curran, and Dan Klein. 2013. An empirical examination of challenges in Chinese parsing. In *Proc. ACL*.

Qv Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proc. ICML*.

Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. 2016. Visualizing and understanding neural models in nlp. In *Proc. NAACL*.

Yang Liu, Qun Liu, and Yajuan Lü. 2011. Adjoining tree-to-string translation. In *Proc. ACL*.

Mitchell P Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2).

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

Slav Petrov and Ryan McDonald. 2012. Overview of the 2012 shared task on Parsing the Web. In *Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL)*.

Sameer S Pradhan and Nianwen Xue. 2009. Ontonotes: the 90% solution. In *Proc. NAACL*.

Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proc. EMNLP*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Proc. NIPS*.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Proc. NIPS*.

Kaizhong Zhang and Dennis Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6).

Han Zhao, Zhengdong Lu, and Pascal Poupart. 2015. Self-adaptive hierarchical sentence model. In *Proc. IJCAI*.