Syntax-Aware Model for Neural Machine Translation with Morpheme Structure in Korean

Hyungrok Kim, Jubeen Lee, and **Donghyun Kim**
School of Computing, KAIST, Republic of Korea
{q0115643, ljb7977, internet}@kaist.ac.kr

Abstract

We propose hybrid models for neural machine translation to incorporate syntactic structures into neural translation models. We rely on two recent structures incorporating linguistic prior to machine translation, graph-convolutional networks (GCNs), and NMT+RNNG, that learns to parse and translate by combining the recurrent neural network grammar into the attention-based neural machine translation. Here, we propose a hybrid models, that uses both source side and target side linguistic priors into machine translation, and especially when Korean is target-side language, we propose to use NMT+RNNG's multitask learning structure and use in-process-generated target dependencies for piecing up Korean Morpheme Based outputs into complete sentence form to organically generate dependencies and learn to have better target output sentences simultaneously, and expect over direct effect of both sides' linguistic priors. We looked forward to observing improvements over traditional syntax-agnostic NMT models in all the considered setups.

1 Introduction

Neural machine translation (NMT) has been making successes with deep learning in natural language processing. The recent NMT systems outperformed traditional phrase-based approaches on many language pairs without any prior linguistic knowledge. But recently, some approaches were coming up with incorporating linguistic priors into the NMT models. We get motivation from two approaches that exploits syntactic features into NMT and combine those into 1 model and having novel application in to morpheme-based structure to suggest a new model for Korean translation.

There have been researches about analyzing target-side Morpheme structures to feed better learning in neural machine translation (Dalvi et al., 2017). But there are none of other approaches like the model, which is proposing the best use of in-translate generated dependency by multitask learning structure by applying that dependency into composing morpheme outputs into complete sentence and feed the whole model to get better output sentence at simultaneously.

2 Related Work

Until last year, NMT systems relied on sequential encoder-decoder model Sutskever et al. (2014); Bahdanau et al. (2014). On the other hand, there has been some recent approaches showing the potential benefit of explicitly encoding the linguistic prior into NMT. Sennrich and Haddow (2016) proposed to augment each source word with its corresponding part-of-speech tag.
Eriguchi et al. (2017) designed a hybrid decoder for NMT, called NMT+RNNG, which combines a usual conditional language model and a recently proposed Recurrent Neural Network Grammars (Dyer et al., 2016). Bastings et al. (2017) presented an effective approach to integrating syntax into neural machine translation models. As we provide descriptions later in section 3, they proposed RNN(or CNN) + GCN structure in encoder of NMT, to integrate syntactic information of the input sequence into the base NMT model.

3 Approach

To make a new approach able to improve current NMT models, we propose a hybrid model using both features from (Bastings et al., 2017) and (Dyer et al., 2016) to provide both the encoder and the decoder rich syntactic information and let them decide which aspects of syntax are beneficial for translation, without placing rigid constraints on the interaction between syntax and the translation task. To easily combine two different structrue, we reconstructed final model implementation with PyTorch framework.

3.1 Background

3.1.1 Attention based Encoder-to-Decoder Neural Translation

The encoder reads a sequence of words $\boldsymbol{x} = (x_1, x_2, \ldots, x_N)$ of source sentence. Then the encoder returns a encoded sequence (hidden states) $\boldsymbol{h} = (h_1, h_2, \ldots, h_N)$. Each hidden state h_i is a concatenation of each directional encoder RNN's output: $h_i = \left[\overrightarrow{h_i}; \overleftarrow{h_i}\right]$ where

$$\overrightarrow{h}_i = \overrightarrow{f}_{enc}(\overrightarrow{h}_{i-1}, V_x(x_i)),$$

$$\overleftarrow{h}_i = \overleftarrow{f}_{enc}(\overleftarrow{h}_{i-1}, V_x(x_i))$$

$V_x(x_i)$ refers to the embedding of the i-th source word. The decoder generates each target word y_j by predicting the probability of y_j given x.

$$p(y_j = y|\mathbf{y}_{<j}, \boldsymbol{x}) = softmax(W_y^\top \tilde{s}_j)$$

$$\tilde{s}_j = \tanh(W_c[s_j; c_j])$$

$$s_j = f_{dec}(s_{j-1}, [V_y(y_{j-1}); \tilde{s}_{j-1}])$$

where f_{dec} is a LSTM unit and W_y is y's output word embedding.
c_j is a context vector computed by the global attention model using the hidden states of encoder (Luong et al., 2015). It can be computed from multiplicating alignment vector and encoder's output vectors. An alignment vector $\boldsymbol{a}_t$ is derived by comparing the current target hidden state $\boldsymbol{h}_t$ with each source hidden state $\boldsymbol{h}_s$.

$$\begin{aligned} \boldsymbol{a}_t(s) &= \text{align}(\boldsymbol{h}_t, \boldsymbol{h}_s) \\ &= \frac{exp(score(\boldsymbol{h}_t, \boldsymbol{h}_s))}{\sum_{s'} exp(score(\boldsymbol{h}_t, \boldsymbol{h}_{s'}))} \end{aligned} \tag{1}$$

By multiplicating this and hidden states of encoder, the context vector c_j can be derived: $c_j = \sum_i a_{i,j} h_i$.

3.1.2 Recurrent Neural Network Grammars

Recurrent neural network grammar (RNNG) is a trasition-based parser having buffer, stack and action sequence, which are implemented as a Stack LSTM to model the parser state and preserve complete history of parser.

At each step, the action sLSTM predicts next action based on hidden states of buffer and stack.

$$p(a_t = a|\boldsymbol{a}_{<t}) \propto e^{W_a^\top f_{action}(h_t^{buffer}, h_t^{stack}, h_t^{action}))}$$

And each hidden states of stackLSTMs can be updated by following:

$$h_t^{buffer} = \text{stackLSTM}(h_{top}^{buffer}, V_y(y_{t-1}))$$

$$h_t^{stack} = \text{stackLSTM}(h_{top}^{stack}, r_t)$$

$$h_t^{action} = \text{stackLSTM}(h_{top}^{action}, V_a(a_{t-1})))$$

where V_y and V_a are functions returning the target word and action vectors. The input vector r_t of the stack sLSTM is computed by

$$r_t = \tanh W_r[r^d; r^p; V_a(a_t)]]$$

where r_d is parent phrase vector and r_p is dependent phrase vector. (Dyer et al., 2015)

3.1.3 NMT+RNNG

NMT+RNNG(Eriguchi et al., 2017) learns to both parse and translate by combining the recurrent neural network grammar (RNNG) into the attention-based neural machine translation model. They replaced the buffer of RNNG (h_t^{buffer}) to hidden state of original decoder (s_j) so it can simultaneously summarize the shifted words as well as generates future words.

NMT+RNNG models the conditional distribution over all possible pairs of translation and its parse given a source sentence (i.e. $p(y, a|x)$). Assuming the availability of parse annotation in the target-side of a parallel corpus, train the whole model jointly to maximize $E_{(x,y,a)\sim data}[\log p(\mathbf{y}, \mathbf{a}|\mathbf{x})]$.

3.1.4 Syntactic Graph Convolutional Network

The GCNs(Bastings et al., 2017) use syntactic dependency trees of source sentences to produce representations of words that are sensitive to their syntactic neighborhoods.

Different weight matrices should be applied according to the direction of edge (incoming or outgoing). Now, the recursive computation of original GCN could be revised as following:

$$\boldsymbol{h}_v^{j+1} = \rho\left(\sum_{u \in N(v)} W_{dir(u,v)}^j \boldsymbol{h}_u^j + \boldsymbol{b}_{dir(u,v)}^j\right)$$

$$\boldsymbol{h}_v^{j+1} = \rho\left(\sum_{u \in N(v)} W_{lab(u,v)}^j \boldsymbol{h}_u^j + \boldsymbol{b}_{lab(u,v)}^j\right)$$

The different weight matrices were used according to combination of label and direction. To prevent over-parameterization, they used the same weight matrix W regardless of label but only regard to direction. Instead, they gave difference on bias vector with regard to label.

Syntactic GCNs have gates to control noise by ignoring erroneous edges. For that, a scalar gate value can be calculated as following:

$$g_{u,v}^j = \sigma(\boldsymbol{h}_u^j \cdot \hat{\boldsymbol{w}}_{dir(u,v)}^j + \hat{b}_{lab(u,v)}^j))$$

Two parameters are learned.

$$\hat{\boldsymbol{w}}_{dir(u,v)}^j \in R^d, \hat{b}_{lab(u,v)}^j \in R$$

Finally, hidden states can be computed as:

$$\boldsymbol{h}_v^{j+1} = \rho\left(\sum_{u \in N(v)} g_{u,v}^j (W_{dir(u,v)}^j \boldsymbol{h}_u^j + \boldsymbol{b}_{dir(u,v)}^j)\right)$$

3.2 Our Approach

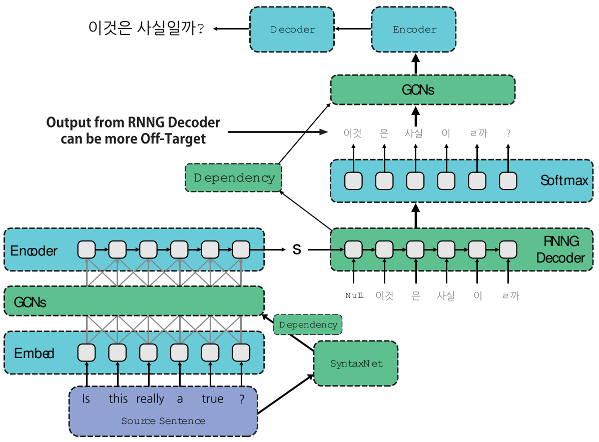


Figure 1: Complete model with GCN encoder + RNNG decoder + externel encoder-decoder fed by RNNG outputs to build complete sentence.

1 shows our proposal in detailed Architecture. Instead of building whole structure and proceeding desired experiments, we made 2 different model to test on.

Model 1. GCN Encoder-to-RNNG Decoder which is for analyzing the translation with incorporating both source and target side syntactic dependencies.

Model 2. NMT+RNNG-to-Morpheme Composition GCN Encoder (same as the figure without the GCNs at source-side), to analyze on in-process parsed dependency usage into target-side Morphemes Composition.

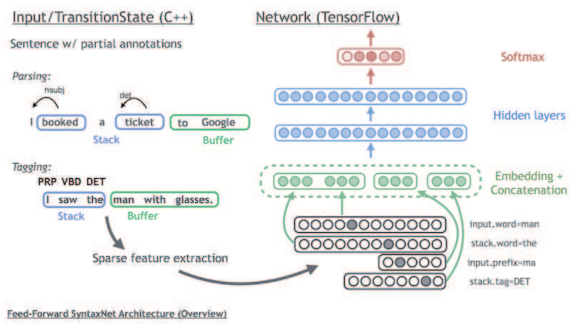
If that morpheme composition was just for finding correct compositions for some morphemes, it wouldn't need too complex structure. But here, because we are in phase of Neural Machine Translation, and the output morphemes from RNNG decoder might not be correctly predicted.

That is why we expect more from that morpheme composition, because it is trained with the whole model, last GCN and Encoder-Decoder should not act like accurate morpheme composer, those should be trained as influencing the whole process. In some degree, similar to how NMT+RNNG (Eriguchi et al., 2017) learns to parse and to translate simultaneously with dependency parsed data as linguistic priors.

4 Experiments

4.1 Syntactic Parser

Before incorporating En-Kr parallel corpus into our models, we had to feed the corpus to syntactic parser to get parsed data in tokenized, CONLL-formatted text.



Input/TransitionState (C++)
Network (TensorFlow)
Sentence w/ partial annotations
Parsing:
nsubj
det
I booked a ticket to Google
Stack
Buffer
Tagging:
PRP VBD DET
I saw the man with glasses.
Stack
Buffer
Sparse feature extraction
Softmax
Hidden layers
Embedding + Concatenation
input.word=man
stack.word=the
input.prefix=ma
stack.tag=DET
Feed-Forward SyntaxNet Architecture (Overview)

Figure 2: SyntaxNet Architecture

SyntaxNet[1] A TensorFlow based framework that provides a foundation for Natural Language Understanding systems. By SyntaxNet trained with treebank corpus, we can do POS-tagging an Transition-based dependency parsing to appropriate language sentences. For tokenizing and parsing English sentences, we trained it with the standard corpora of the Penn Treebank[2]. But for Korean sentences, we needed to get a different corpus since UD-Korean corpus was based on

[1] https://opensource.google.com/projects/syntaxnet
[2] https://catalog.ldc.upenn.edu/LDC99T42

the word unit and not the morpheme units, so we trained the parser with Sejong Treebank Corpus by Korean Language Institute. Which has dependencies based on morpheme units. Also, to apply our Morpheme-based Korean SyntaxNet on our Korean corpus for translation, we used KoNLPy (Park and Cho, 2014) to tokenize Korean sentences based on morpheme units.

4.2 Corpora

	Train.	Dev.	Test.	Voc. *(kr, en, act)*
En $\leftrightarrow$ Kr	99,999	10,000	10,000	(25,869, 27,732, 81)

Table 1: Statistics of parallel corpora.

We compare the two proposed GCN+RNNG models against the baseline model, NMT+RNNG on two different language pairs Kr-En, En-Kr. The basic statistics of the training data are presented in Table 1. We mapped all the low-frequency words to the unique symbol UNK and inserted a special symbol EOS at the end of both source and target sentences.

En-Kr We used English-Korean parallel corpus[3] which was collected from the Web by (Kim et al., 2010) This corpus has about 410,000 sentences, but we excluded sentences with rare special characters(e.g. *, #, {, ...) and pairs those have length longer than 80 characters, extracted 99,999 sentences for train data, 10,000 for both tuning data and test data. We mapped all the low-frequency tokens to the unique symbol UNK, built our vocabulary with tokens with more than 3 appearances and inserted a special symbol EOS at the end of both source and target sentences.

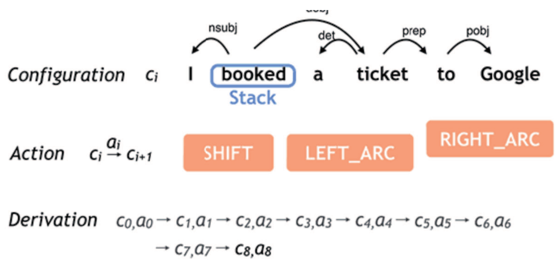

nsubj
det
prep
pobj
Configuration
c_i
I booked a ticket to Google
Stack
Action
$c_i \xrightarrow{a_i} c_{i+1}$
SHIFT
LEFT_ARC
RIGHT_ARC
Derivation
$c_0,a_0 \rightarrow c_1,a_1 \rightarrow c_2,a_2 \rightarrow c_3,a_3 \rightarrow c_4,a_4 \rightarrow c_5,a_5 \rightarrow c_6,a_6 \rightarrow c_7,a_7 \rightarrow c_8,a_8$

Figure 3: Arc-Standard Transition Overview

Nivre's Arc-Standard Transition To feed the parsed dependencies to RNNG structures, we needed to build our algorithm extracting transition-actions based on Nivre's Arc-standard Transition rules, from SyntaxNet's output(CONLL-format.)

[3]The dataset and the parallel corpus are available on the authors website, http://isoft.postech.ac.kr/~megaup/research/resources/

4.3 Model 1: GCN-to-RNNG on Kr-En

GCN-RNNG Our first trial was experiments with GCN-to-RNNG model, which has source sentence$\rightarrow$ GCN-encoder $\rightarrow$ RNNG-decoder $\rightarrow$ target sentence structure

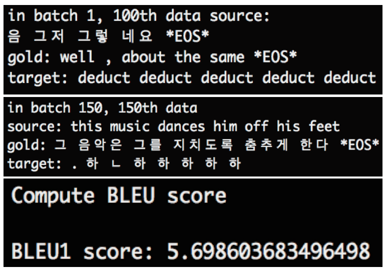
In all our experiments, since we had very large size of vocabularies, we wanted to increase the token embedding dimensions to capture those differences on tokens, but due to computational power, we had to lower it to 128, in the word vectors and the action vectors were of 128 and 32 dimensions respectively. Action vectors and action LSTMs were with small dimension since it didn't have many variations. Each recurrent network has a single layer of LSTM units of 128 hidden dimensions, only the LSTM for action embeddings have 16 hidden dimension. All LSTMs' hidden states and weights were initialized with zero, and forget biases were initially set to 0.01. In GCNs, weight matrices were initialized with Xavier Weight Initialization with gain 1. All biases were initialized with zero. Weights for gates were initialized with uniform distribution between 0 and 0.01.

Our mini-batch size was 256, loss function was computed by negative log likelihood loss, for optimizer we used RMSprop. The learning rate was set to 0.01 and epochs were set to 30.

When every epoch ends, we computed BLEU1 score with the develop data and saved model in every high-score by that. In LSTMs acting as a decoder, generating target words or transition actions, we made it not to choose *unknown* (saved when loading corpus.)

4.4 Model 2: NMT+RNNG-to-GCN on En-Kr

NMT+RNNG-to-GCN With this model, basically all hyper-parameters were set same as in Model 1. To feed the dependencies made from RNNG, to the GCNs behind RNNG, we needed to convert the transition actions to dependency tree structure, but just by using actions from action-decoder, it was not suitable to make a tree in most of the times, so we constrained in decoder, to generate SHIFT or REDUCE less than target words, and rejected every unsuitable action sequences.



```
in batch 1, 100th data source:
음 그저 그렇 네요 *EOS*
gold: well , about the same *EOS*
target: deduct deduct deduct deduct deduct
```

```
in batch 150, 150th data
source: this music dances him off his feet
gold: 그 음악은 그를 지치도록 춤추게 한다 *EOS*
target: . 하 ㄴ 하 하 하 하 하
```

```
Compute BLEU score

BLEU1 score: 5.698603683496498
```

Figure 4: example outputs from our experiment, first one is from Model 1, and second one is from Model 2, and an example BLEU1 score that was computed.

5 Conclusion and Future Work

Even with the enormous efforts on complete construction of our complex model (especially RNNG), from 4 we can clearly see our experiment didn't give us any good results. According to our analysis, we were able to bring up corresponding reasons.

5.1 Problem 1: Very Large Vocabularies

We had 25,869 different tokens from Korean, 27,732 for English. Currently we used a random initialized weight matrix to build our token embeddings, making the matrix into trainable parameter in our overall training, and that is fragile to deal with rare words and phenomena such as inflection and compounding.

There are clear improvements we can make, Learning **Byte-Pair Encodings (BPE)** as described by Sennrich et al.. We expect in our result, this can solve phenomenons that output tokens are converging to few tokens.

5.2 Problem 2: Gap between Dependency Tree and Transition Action Sequence

This was actually the main reason we had experiments with 2 models. We were not confident in adjusting Action Decoder in RNNG to get sequences that are always able to convert into Dependency Tree. Like we decribed, we have done it with simple algorithm, but we expect it is disturbing the models quality.

There can be solutions, designing better loss function for this action decoding since rejecting the output and looping is not common phenomenons. And building better algorithm for constraining the action decoder to only generate sequences that can be converted into Dependency Tree.

5.3 Problem 3: Lack of Computational Power and Incomplete Baseline Model

Our baseline model was actually two, NMT+RNNG(Eriguchi et al., 2017) and NMT with GCN encoder(Bastings et al., 2017). We knew it would be a very complex model, especially for RNNG. As we started working in earnest, we found out that RNNG code[4] was not built properly. it took much time in building the whole model from scratch, this eventually became the biggest reason for other problems we still have. And for GCN and other NMT papers, we can see that those authors have done the experiments with huge corpus(4,500,966 train sentences by Bastings et al..) It was not possible for us to come up with trained performances that can be compared with base papers.

5.4 Problem 4: Too Complex Neural Net Structure

Even though we had computational problems, actually our model was over-parameterized if we look inside the NMT+RNNG model and extra encoder-decoder with GCN.

5.5 Problem 5: Inappropriate Corpus for Translation

For our translation task, we used web-collected corpus by Kim et al.. But it was not developed for translation, it was developed for Relation Detection with Cross-lingual Approach. If we look through the corpus we can see too many abnormal sentences like technical manuals for complex machines(those have too many special characters.)

5.6 Conclusion

We faced many problems, but we believe our idea of using NMT+RNNG's dependency output with extra GCN encoder of Morpheme composer has a great worth to tryout.

We expect, with applying subword-unit encoding (Sennrich and Haddow, 2016) and those possible solutions, we have quite a possibility of proving the performance of our novel idea. We look forward to work on our project after this semester with all solutions we listed.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly

[4] https://github.com/tempra28/nmtrnng

learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. 2017. Graph convolutional encoders for syntax-aware neural machine translation. *arXiv preprint arXiv:1704.04675*.

Fahim Dalvi, Nadir Durrani, Hassan Sajjad, Yonatan Belinkov, and Stephan Vogel. 2017. Understanding and improving morphological learning in the neural machine translation decoder. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 142–151.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. 2015. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. 2016. Recurrent neural network grammars. *arXiv preprint arXiv:1602.07776*.

Akiko Eriguchi, Yoshimasa Tsuruoka, and Kyunghyun Cho. 2017. Learning to parse and translate improves neural machine translation. *arXiv preprint arXiv:1702.03525*.

Seokhwan Kim, Minwoo Jeong, Jonghoon Lee, and Gary Geunbae Lee. 2010. A cross-lingual annotation projection approach for relation detection. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 564–571. Association for Computational Linguistics.

Korean Language Institute. 2012. Sejong treebank. http://www.sejong.or.kr.

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.

Eunjeong L. Park and Sungzoon Cho. 2014. Konlpy: Korean natural language processing in python. In *Proceedings of the 26th Annual Conference on Human Cognitive Language Technology*, Chuncheon, Korea.

Rico Sennrich and Barry Haddow. 2016. Linguistic input features improve neural machine translation. *arXiv preprint arXiv:1606.02892*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

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.

A Source Code

https://github.com/q0115643/GCN-RNNG

B Individual Contribution

Hyungrok Kim:

- Developed our idea to be a complete model we can develop.
- Implemeted our NMT model's base structure with attention-based encoder-decoder model.
- Implemented Graph Convolutional Network Encoder.
- Trained SyntaxNet with appropriate treebank and built dependency parsed corpus.
- Implement preprocessing and data-loading into our model.

Jubeen Lee:

- Suggested motivation of our project idea.
- Analyzed original RNNG+NMT source code written in CPP.
- Converted the original source code into PyTorch, with attention mechanism and stackLSTM.

Donghyun Kim:

- Analyzed NMT+RNNG code from original author.
- Developed our idea to be an organized model.
- Implemented our evaluation method.