# Fast and Accurate Reordering with ITG Transition RNN

**Hao Zhang**
Google
haozhang@google.com

**Axel Ng**
Google

**Richard Sproat**
Google

**Abstract**

Attention-based sequence-to-sequence neural network models learn to jointly align and translate. The quadratic-time attention mechanism is powerful as it is capable of handling arbitrary long-distance reordering, but computationally expensive. In this paper, with the goal of making neural translation both accurate and efficient, we follow the traditional pre-reordering approach to decouple reordering from translation. We add a reordering RNN that shares the input encoder with the decoder. The RNNs are trained jointly with a multi-task loss function and applied sequentially at inference time. The task of the reordering model is to predict the permutation of the input words following the target language word order. After reordering, the attention in the decoder becomes more peaked and monotonic. For reordering, we adopt Inversion Transduction Grammars (ITG) and propose a transition system to parse input to trees for reordering. We harness the ITG transition system with RNN. With the modeling power of RNNs, we achieve superior reordering accuracy without any feature engineering. In experiments, we apply the model to the task of text normalization. Compared to a strong baseline of attention-based RNN, our ITG RNN reordering model can reach the same reordering accuracy with only 1/10 of the training data and is 2.5x faster in decoding.

## 1 Introduction

The encoder-decoder neural network architecture for sequence-to-sequence problems has achieved enormous success especially after the introduction of the attention mechanism (Bahdanau et al., 2014). Its applications in NLP range from machine translation (Bahdanau et al., 2014) and sentence summarization (Rush et al., 2015) to text normalization (Sproat and Jaitly, 2017). The attention mechanism is effectively a random memory access mechanism, enabling access to any source sequence position at any decoding step. In principle, it can handle arbitrary reordering of any input length. But its power comes with a high computational cost. The time complexity of the attention mechanism is $O(N^2)$ if the number of decoding steps is proportional to the input length $N$. In addition to the computational concern, for dominantly-monotonic translation tasks, such as text normalization (Sproat and Jaitly, 2017), soft attention can be too relaxed and sub-optimal in terms of modeling efficiency. Hence, to reduce computational complexity as well as to improve model accuracy, recently there has been a surge of research interest in enforcing monotonic attention (Raffel et al., 2017) and hard attention (Aharoni and Goldberg, 2017) based on the observation that many sequence-to-sequence tasks are monotonic, including speech recognition and morphological inflection.

In reality, the monotonicity assumption has to be made carefully. Even in the highly monotonic translation task of text normalization, which is mapping written text to spoken text, there are systematic reordering patterns like from "*2018-10-03*" to "*October third, two thousand eighteen*", and from "*$100*" to "*one hundred dollars*". These reordering patterns can involve arbitrarily long chunks of input. Without handling the infrequent but systematic reordering, monotonic attention models are doomed. We claim there is hope for efficient and accurate systematic reordering by combining grammars with RNNs. The

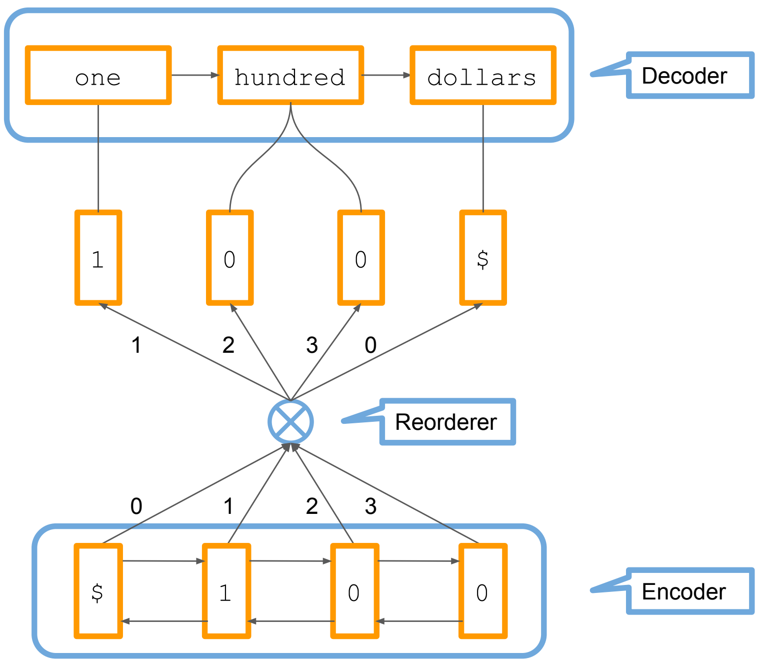

Figure 1: The encoder-reorderer-decoder architecture. The input encoder is shared by the reorderer and the decoder. The reorderer permutes the RNN states of the encoder.

key to our solution is the Inversion Transduction Grammars (Wu, 1997), a type of synchronous context free grammar limiting reordering to adjacent source spans. For machine translation across very different languages, ITGs have been reported to cover most of the alignments observed in parallel data (Zhang et al., 2006). For text normalization, we have not found a single example of reordering that cannot be covered by an ITG. This observation motivates us to factorize translation into two steps: an ITG reordering step followed by a monotonic translation step. The task of the reordering step is to handle systematic reordering through swapping of adjacent source spans. This step "normalizes" the input in terms of word order which opens up the capacity for improvement in translation accuracy. It also opens up the opportunity for more "lightweight" neural translation models such as monotonic attention models of Raffel et al. (2017) and Aharoni and Goldberg (2017). Of course, the burden of reordering has shifted from the attention mechanism to the dedicated reordering model. This idea has produced fruitful results in the era of phrase-based machine translation (Collins et al., 2005; Xu et al., 2009; Neubig et al., 2012; Lerner and Petrov, 2013; Nakagawa, 2015). In this paper, we revive the old idea with a neural treatment.

First of all, we modify the encoder-decoder architecture by adding a reorderer which shares the encoder with the decoder. Figure 1 shows the architecture. This change enables multi-task training of the encoder states and turns the inference into three steps: encoding, reordering, decoding, each of which is a linear chain RNN. Our main algorithmic contribution is an ITG-transition-based RNN reorderer. What we feed to the RNN for training are input strings paired with their permutations, for example *\$100* with $(1, 2, 3, 0)$ to indicate it should be reordered to *100\$* in order to be translated monotonically. Within the ITG framework, we will show the goal is equivalent to parsing the input into a parse tree annotated with reordering information, in this case $\langle$ *\$* [ [ *1 0* ] *0* ] $\rangle$, where $\langle\ \rangle$ indicates a swapping of two subtrees and [ ] indicates no swapping. For efficiency and compatibility with linear chain RNN, we use a transition system for parsing. The transition system has three actions: SHIFT, REDUCE S, and REDUCE I. In this example, the transition sequence to learn is SHIFT, SHIFT, SHIFT, REDUCE S, SHIFT, REDUCE S, REDUCE I. To learn the conditional distributions of the three actions, we rely on the strength of RNN to condition on the entire transition history without any feature engineering. We report results for both the intrinsic task of reordering and the end-to-end task of translation.

The paper is organized in the following way. In Section 2, we formally introduce ITG and its transition-based parsing system. In Section 3, we introduce the ITG-based RNN model. In Section 4, we formally introduce the end-to-end neural architecture. In Section 5, we discuss related works. In Section 6.1, we carry out two sets of reordering experiments: one on synthetic sequence permutation data, the other on money expression reordering data coming from the task of English text normalization (Sproat and Jaitly, 2017). In Section 6.2, we conduct an end-to-end experiment for text normalization and compare the two-step approach with the baseline approach.

## 2 ITG Transition System

An Inversion Transduction Grammar is a grammar for generating a pair of languages recursively and synchronously. In our problem of source language reordering, we essentially pair the source language with a virtual target language that shares the vocabulary with the source but follows the word order of the real target language. To be concrete, we use the simplest ITG with just one nonterminal and the following context free production rules.

$$
\begin{gathered}
X \rightarrow [X\ X] \\
X \rightarrow \langle X\ X \rangle \\
X \rightarrow w_0/w_0 \\
\ldots \\
X \rightarrow w_{n-1}/w_{n-1}
\end{gathered}
$$

The first rule is called the *straight rule* because it keeps the order of two constituents unchanged on the target side. The second rule is called the *inverted rule* because it inverts the order of two constituents on the target side. For example, a sentence pair $(w_0, w_1, w_2 \mid w_0, w_2, w_1)$ can be derived with three pre-terminal rules to link the words with the same subscripts on both sides, plus one inverted rule for grouping $w_1$ and $w_2$ and one straight rule on the top for grouping $w_0$ and the phrase of $w_1, w_2$.

We can also devise a transition system following (Nivre, 2003). The following is the deductive description of the transition system. Each configuration in the transition system consists of a stack and a pointer to the next word in the input buffer. At the beginning, we have an empty stack and a pointer to the first input word. At each time step, we can choose SHIFT if the buffer is not empty and choose to apply REDUCE S or REDUCE I if the stack has a height of at least two. We use four indices to uniquely represent each synchronous constituent that is constructed in the parsing process. The first two index into the source side and the last two index into the target side. The transition system stops when the buffer is empty and the stack has been reduced to size one.

$$
\begin{gathered}
\textsc{Initial} : [\phi,\ 0] \\
\textsc{Shift} : \frac{[S, i]}{[S|(i,i,i,i),\ i+1]} \\
\textsc{Reduce S} : \frac{[S|(i,j,l_1,r_1)(j+1,k,l_0,r_0),\ k+1]}{[S|(i,k,l_1,r_0),\ k+1]} \\
\textsc{Reduce I} : \frac{[S|(i,j,l_1,r_1)(j+1,k,l_0,r_0),\ k+1]}{[S|(i,k,l_0,r_1),\ k+1]} \\
\textsc{Final} : [(0, n-1, *, *),\ n]
\end{gathered}
$$

When training a model, we are given a sequence of words $w_0, ..., w_{n-1}$, along with the permutation $P$ indicating its corresponding target word order. It has the equivalent information as a pair of bijectively aligned sequences:

$$w_0, ..., w_{n-1} \quad w_{P(0)}, ...w_{P(n-1)}$$

Zhang et al. (2006) have shown for any $P$ there exists a greedy shift-reduce algorithm to produce a *canonical parse* along with a *canonical transition sequence*. If we take the *$100* example, the canonical transition sequence is SHIFT, SHIFT, SHIFT, REDUCE S, SHIFT, REDUCE S, REDUCE I, and the corresponding cannonical parse is $\langle\ \$\ [\ [\ 1\ 0\ ]\ 0\ ]\ \rangle$. Therefore, given $w_0, ..., w_{n-1}$ and $P$, the learning problem is mapping from $w_0, ...w_{n-1}$ to the canonical transition TRANSITION$(P)$. There are two good properties of the output space. First, the output sequence is always of length $2n-1$. Second, the output vocabulary is fixed to $\{$SHIFT, REDUCE S, REDUCE I$\}$.

Once a model is learned, at testing time, only the input sequence is given, the task is to predict a transition sequence which in turn corresponds to a permutation.

## 3 ITG RNN

A recurrent neural network computes the representation of a new state based on the representation of its previous state and input at the current step. Formally, this is

$$h_t = RNN(f(t), h_{t-1})$$

where $h$ is the hidden state representation and $f$ is the input function, and $RNN$ can be a LSTM or GRU cell for instance.

A transition system also has a timeline recurrence. The recurrence of the ITG transition system is the following.

$$s_t = ITG(a(t), s_{t-1})$$

where $s$ is the configuration and $a$ is the action.

To make an ITG controlled by an RNN, we need to unroll an ITG transition system along with an RNN. Figure 2 shows an example of completely unrolled computation graph of ITG-RNN. For input $f$, we want this to be a function of $s$. The information in $s$ is a stack along with an input buffer pointer. In Section 2, we specified each stack element as a tuple of $(i, j, l, r)$ that indexes into the source sequence and the reordered target sequence. It is clear that the deduction rules involve at most top two stack elements. Therefore, we extract the indices stored at the top of the stack which are potentially directly involved in any next action: $l_1, r_1, l_0, r_0$, and the buffer pointer $i$. With these input indices, we can create a concatenated embedding vector of the input sequence. This will be what the ITG feeds into the RNN as input.

The task of the RNN is to predict the possible next actions. As the ITG transition system constrains the valid next actions at $s_t$, $h_t$ should only emit action labels allowed by $s_t$. This will be what ITG places as constraints in the RNN output and will indirectly influence the training of the underlying RNN cells by moving probability mass away from invalid actions. The output constraints are implemented as masks over invalid output actions before applying softmax.

What RNN feeds back to ITG is the predicted actions after softmax. That is how the RNN controls the ITG at inference time. At training time, the action sequence is known.

## 4 Encoder-Reorderer-Decoder Architecture

As we discussed in Section 1 and demonstrated with Figure 1, a reorderer is designed to solve translation into two steps: reordering and monotonic decoding. In this section, we define the training and decoding objectives mathematically.

Given an input sequence $(x_1, \ldots, x_{T_x})$, a BiRNN, i.e., a pair of forward RNN $\overrightarrow{f}$ and backward RNN $\overleftarrow{f}$ is used to encode it into a pair of *forward hidden states* and *backward hidden states* $(\overrightarrow{h}_1, \ldots, \overrightarrow{h}_{T_x})$ and $(\overleftarrow{h}_1, \ldots, \overleftarrow{h}_{T_x})$, where $\overrightarrow{h}_t = \overrightarrow{f}(x_t, \overrightarrow{h}_{t-1})$ and $\overleftarrow{h}_t = \overleftarrow{f}(x_t, \overleftarrow{h}_{t+1})$. The forward and backward states are concatenated to summarize contexts in both directions: $h_t = [\overrightarrow{h}_t^\top; \overrightarrow{h}_t^\top]^\top$. This is how the encoder works.

Here we depart from the attention-based encoder-decoder architecture. The reorderer defines a probability of the ITG transition sequence $z_1, \ldots, z_{2T_x-1}$ as a product of conditionals.

$$p(z_1, \ldots, z_{2T_x-1} | x_1, \ldots, x_{T_x}) = \prod_{t=1}^{2T_x-1} p(z_t | z_1, \ldots, z_{t-1}, h_1, \ldots, h_{T_x}) \tag{1}$$

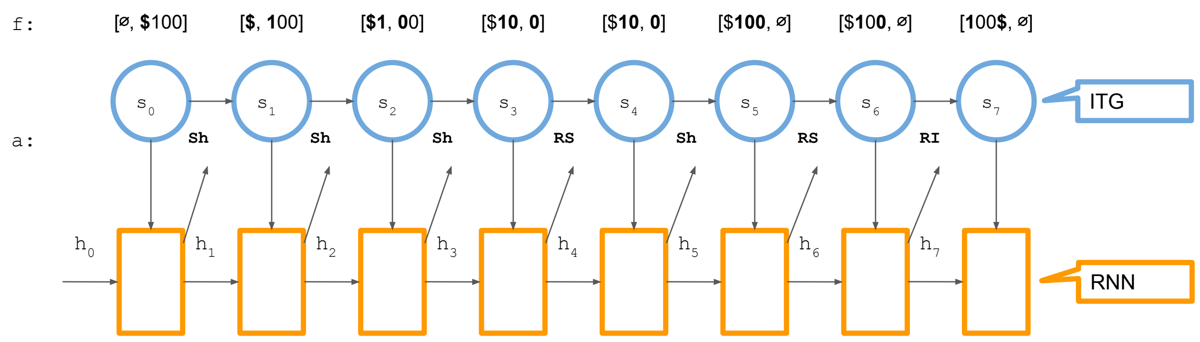

Figure 2: An unrolled computation graph of ITG-RNN. $s$ is for the states of the ITG. $h$ is for the states of the RNN. $f$ is for the input function to the RNN. $a$ is for actions of the ITG and softmax over output of the RNN. For conciseness, we use SH for SHIFT, RS for REDUCE S, and RI for REDUCE I. We use boldface to indicate function $f$ as input focus over the stack and the buffer.

For the ITG RNN, the conditional probability is based on softmax over the output at each step $t$. As we mentioned in Section 2, for each transition sequence, there exists a functional mapping to a permutation $P$. Therefore, the probability of reordering is derived in the following way.

$$p(x_{P(1)}, \dots, x_{P(T_x)} | x_1, \dots, x_{T_x}) = p(z_1, \dots, z_{2T_x - 1} | x_1, \dots, x_{T_x})$$

The key change following reordering is that the probability of the output sequence $(y_1, \dots, y_{T_y})$ is based on the permuted sequence $(x_{P(1)}, \dots, x_{P(T_x)})$ and we make an independence assumption with respect to the original input sequence.

$$p(y_1, \dots, y_{T_y} | x_1, \dots, x_{T_x}) = p(y_1, \dots, y_{T_y} | x_{P(1)}, \dots, x_{P(T_x)}) \cdot p(x_{P(1)}, \dots, x_{P(T_x)} | x_1, \dots, x_{T_x})$$

where

$$p(y_1, \dots, y_{T_y} | x_{P(1)}, \dots, x_{P(T_x)}) = \prod_{t=1}^{T_y} p(y_t | y_1, \dots, y_{t-1}, h_{P(1)}, \dots, h_{P(T_x)})$$

Usually, a soft attention is applied to attend enough context from $h$. But since the sequence of hidden states is reordered, the attention can be made monotonic.

Let's use $S = (x_1, \dots, x_{T_x})$, $P(S) = (x_{P(1)}, \dots, x_{P(T_x)})$, and $T = (y_1, \dots, y_{T_y})$.

The training objective is

$$1/\mathcal{S} \sum_{(T,S) \in \mathcal{S}} \log p(P|S) + \log p(T|P(S)) \tag{2}$$

and the decoding objective is

$$\hat{T} = \text{arg} \max_{T,P} p(P|S) \cdot p(T|P(S)) \tag{3}$$

The term $-\log p(P|S)$ is the reordering loss and the term $-\log p(T|P(S))$ is the translation loss. Both share the input $S$. In a computation graph, they share the BiRNN component. The hidden states $(h_1, \dots, h_{T_x})$ are trained for both tasks. At decoding time, we approximate the decoding objective with a beam search over $P$, followed by another beam search over $T$.

## 5 Related Work

There are three papers that are most relevant to ours that relate to ITG pre-ordering. DeNero and Uszkoreit (2011) induce binary source trees first and learn pre-reordering rules for these binary trees from parallel data. Neubig et al. (2012) discriminatively train an ITG parser with CYK parsing for pre-reordering, essentially combining the two steps in DeNero and Uszkoreit (2011) into one. Nakagawa (2015) improve upon Neubig et al. (2012) with a linear time top-down ITG parsing algorithm. They all rely on feature engineering as they use linear models for training. None of them does transition-based parsing for ITG.

There are two papers most relevant to ours that relate to combining transition systems with RNNs. Dyer et al. (2015) propose a stack-LSTM for transition-based parsing. The difference with our approach to modeling is that we do not encode the entire stack explicitly. At each time step, we only feed the context indexed by the current stack and buffer configuration. We do not have an stack RNN, which can be expensive. Our transition RNN can be viewed as a special case of DRAGNN (Kong et al., 2017) which combines fixed features as input with recurrence links at each time step. Our recurrence link is only to the previous time step.

For the approach of incorporating syntactic constraints into neural translation, the following papers are most relevant. Eriguchi et al. (2016) and Chen et al. (2017) assume the existence of source parse trees and enhance the encoder and the attention mechanism to attend to both words and syntactic phrases. We do not rely on external parsers. Stahlberg et al. (2016) let a hierarchical phrase-based decoder guide neural machine translation decoding. Reordering decisions can only be indirectly influenced by the hierarchical decoder. In contrast, we have an explicit hierarchical reordering model applied pre-translation. Eriguchi et al. (2017) train a joint parsing and translation model to maximize the log likelihood of output sequence and input parsing action sequence. This is similar to our multi-task training setup. The key difference is our subtask is ITG parsing for reordering instead of linguistically-motivated parsing.

The idea of adding a reordering layer into neural MT models has also been studied by Huang et al. (2018). They use a simple feed-forward soft and local reordering layer similar to the soft attention mechanism. A fixed window size is used for local reordering. Our RNN reordering layer can handle long distance reordering. Another important difference is that we use discrete variables (permutations) for reordering while the soft reordering mechanism has no latent variables. We leave it as future work to train the end-to-end system by treating ITG transitions and permutations as latent variables.

## 6 Experiments

### 6.1 Reordering Experiments

In this part, we analyze the effectiveness of the ITG RNN reordering model. We first create a baseline system using a standard attention-based RNN. This basically implements the right-hand-side conditional in Equation 1 with an attention-based RNN instead of an ITG-RNN. The encoder is a bidirectional GRU cell RNN with one layer in each direction. The input embedding size is 32. The number of GRU cells is 32. The decoder is again a one-layer GRU RNN with 32 cells. The output embedding size is also 32. To make side-by-side comparison fair, we duplicate the network specification in ITG RNN. The only difference is that Equation 1 is implemented with ITG RNN.

#### 6.1.1 Synthetic Reordering Data

The first set of experiments uses synthetic reordering data. We generated all permutations that have valid ITG transition action sequences up to permutation length 10. There are 258,563 such permutations. Then we scrambled letters “A” through “J” using these permutations and paired them up. So, the task is to sort letter sequences. But the models have no prior knowledge of the alphabet. We shuffled the examples and trained on 80% of the data, validated on 10% of the data, and tested on the rest 10%. Figure 3 shows the contrast between ITG RNN and attention-based RNN as we varied the training data size. ITG RNN is much more efficient in utilizing training examples. It only needs roughly 10% of the training data to get the same accuracy as the attention-based system. The attention-based model never reached zero error rate even after training 2 million steps with batch size 32 using the entire training set.

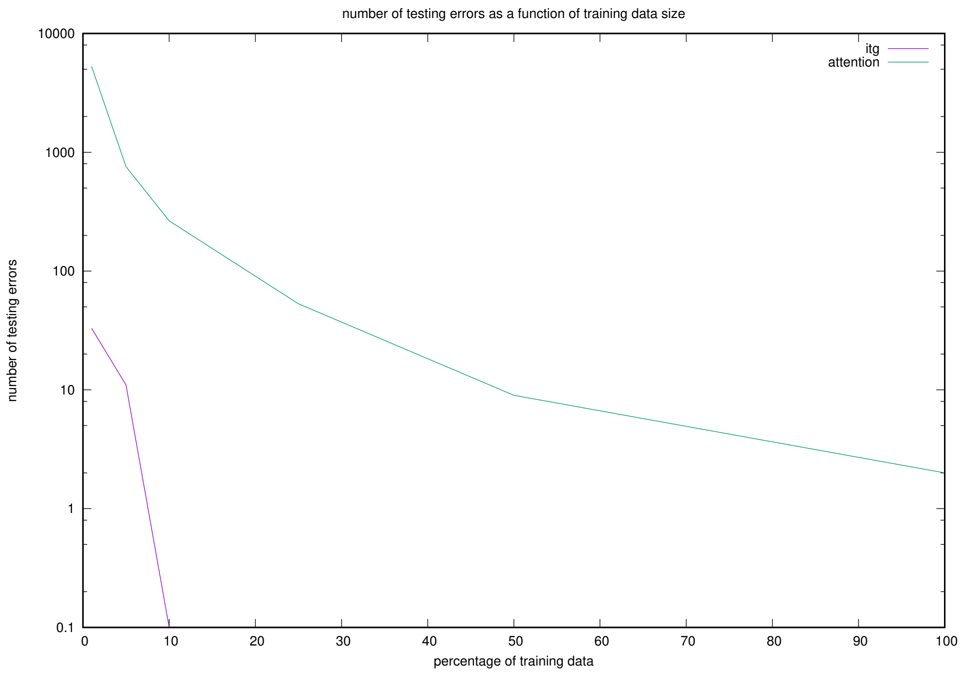

Figure 3: The ITG RNN model reaches zero error rate much faster than the attention-based RNN on the synthetic data set.

| | *attention-based RNN* | *ITG RNN* |
|---|---|---|
| all | $8 \times 10^{-5}$ | $6 \times 10^{-6}$ |
| non-identity | $8 \times 10^{-3}$ | $2 \times 10^{-3}$ |

Table 1: Text normalization reordering error rates.

### 6.1.2 Text Normalization Reordering

For text normalization, interesting reordering patterns include ISO dates such as *1990-09-01*, which should be verbalized as *September first, nineteen ninety*, which indicates a reordering of the last two date components. In measure expressions, interesting cases include *5* $km^2$, which should be read as *five square kilometers*. There are many more money expressions in real data such as *$100*, *USD100*, *RMB 2 million*, but also cases where no reordering is necessary, like *2 million dollars*. These cases make for a good mixture of reordering patterns.

Focusing on money expressions have a total of 615,642 such reordering examples. We did the same 80-10-10 split for train/validate/test. We use the same network configurations as in section 6.1.1. Figure 4 shows the accuracy comparison. We have a similar trend as with the synthetic data, namely that the ITG RNN is much more effective. We also measured the CPU decoding speed of the two models. Using a single core, the ITG RNN is 2.5x the speed of attention-based RNN. It is worth noting that the two models have encoder and decoder RNNs with the same network dimensions. The reduction of attention accounts for most of the saving.

## 6.2 End-to-end Text Normalization

For end-to-end translation, we annotated reordering information on 10,561,377 tokens of training data used by Sproat and Jaitly (2017). There are 349,537 DATE/MONEY expressions, and 14,795 non-identity reordering examples. This is a realistic data set because it reflects that only a tiny portion of all tokens require reordering. Table 1 shows the sequence error rate of the reordering model alone. It can reach 99.8% accuracy on the non-identity reordering subset, while an attention-based reordering component can only give 99.2%.

Table 2 summarizes the end-to-end text normalization results, comparing ITG RNN pre-reordering followed by hard monotonic attention (Raffel et al., 2017) against soft (unordered) attention (Bahdanau

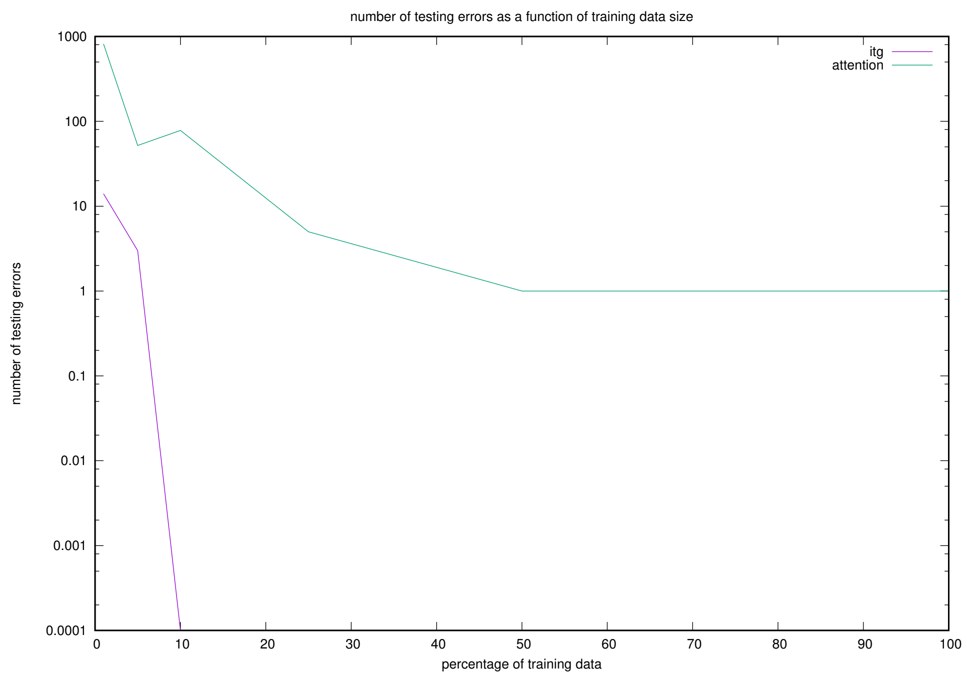

Figure 4: The ITG RNN model reaches zero error rate much faster than the attention-based RNN on the money expression text normalization data set.

| | *attention-based* | | *reorderer-based* | |
|---|---|---|---|---|
| Semiotic Class | Correct | Accuracy | Correct | Accuracy |
| ALL | 143797 | 99.73% | 143778 | 99.72% |
| DATE | 10438 | 99.94% | 10437 | 99.93% |
| MONEY | 5838 | 97.07% | 5815 | 96.69% |

Table 2: Text normalization results.

et al., 2014). The details of the baseline normalization model can be found in Sproat and Jaitly (2017). The new system differs only in the attention mechanism. Essentially, the two systems have the same overall accuracy and DATE accuracy. But unfortunately the reorderer-based one is worse in MONEY accuracy.

The overall result indicates that for neural text normalization pre-reordering is feasible with comparable accuracy and has the potential for higher computational efficiency as shown by Raffel et al. (2017). But why do we not observe improvements in accuracy on this data set even though the intrinsic reordering accuracy has reached 99.8%? We suspect that interesting reordering is too rare on this data set and over time, the soft-attention model is fully capable of learning the reordering. We also suspect that the beam search for the reorderer-based system is more prune to search errors because it has two passes: one for the permutation and the other for the actual output based on the permutation produced by the first pass.

## 7 Conclusion

We propose a neural pre-reordering approach. It reorders the encoder output of the input sequence before feeding it to the decoder RNN. As a result, the burden of modeling reordering is shifted from the attention network to a dedicated reordering model. In particular, we find that combining an Inversion Transduction Grammar based transition system with a RNN results a reordering model that is both fast and accurate. It requires only 1/10 of samples to get to the same reordering sequence accuracy compared to an attention-based RNN reorderer. This demonstrates that reordering can be done more efficiently and accurately with proper structural constraints.

Our future work has two directions. First, we will apply the model to machine translation data, using automatically aligned data like pre-reordering models for phrase-based machine translation models. Second, we will treat the reordering decisions as latent variables and train the model without alignment annotation and decode without an additional beam search.

## Acknowledgements

We thank Shankar Kumar, Xiaochang Peng, and Michael Riley for discussions and comments on this work.

## References

Roee Aharoni and Yoav Goldberg. 2017. Morphological inflection generation with hard monotonic attention. In *ACL*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Huadong Chen, Shujian Huang, David Chiang, and Jiajun Chen. 2017. Improved neural machine translation with a syntax-aware encoder and decoder. In *ACL*.

Michael Collins, Philipp Koehn, and Ivona Kučerová. 2005. Clause restructuring for statistical machine translation. In *ACL*.

John DeNero and Jakob Uszkoreit. 2011. Inducing sentence structure from parallel corpora for reordering. In *EMNLP*.

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

Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Tree-to-sequence attentional neural machine translation. In *ACL*.

Akiko Eriguchi, Yoshimasa Tsuruoka, and Kyunghyun Cho. 2017. Learning to parse and translate improves neural machine translation. In *ACL*.

Po-Sen Huang, Chong Wang, Dengyong Zhou, and Li Deng. 2018. Neural phrase-based machine translation. In *ICLR*.

Lingpeng Kong, Chris Alberti, Daniel Andor, Ivan Bogatyy, and David Weiss. 2017. DRAGNN: A transition-based framework for dynamically connected neural networks. *arXiv preprint arXiv:1703.04474*.

Uri Lerner and Slav Petrov. 2013. Source-side classifier preordering for machine translation. In *EMNLP*.

Tetsuji Nakagawa. 2015. Efficient top-down btg parsing for machine translation preordering. In *ACL*.

Graham Neubig, Taro Watanabe, and Shinsuke Mori. 2012. Inducing a discriminative parser to optimize machine translation reordering. In *EMNLP-CoNLL*.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *IWPT*.

Colin Raffel, Thang Luong, Peter J. Liu, Ron J. Weiss, and Douglas Eck. 2017. Online and linear-time attention by enforcing monotonic alignments. *arXiv preprint arXiv:1704.00784*.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*.

Richard Sproat and Navdeep Jaitly. 2017. An RNN model of text normalization. In *Interspeech*.

Felix Stahlberg, Eva Hasler, Aurelien Waite, and Bill Byrne. 2016. Syntactically guided neural machine translation. In *ACL*.

Dekai Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational. Linguistics*, 23(3).

Peng Xu, Jaeho Kang, Michael Ringgaard, and Franz Och. 2009. Using a dependency parser to improve smt for subject-object-verb languages. In *NAACL*.

Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous binarization for machine translation. In *HLT-NAACL*.