Learning to Order Graph Elements with Application to Multilingual Surface Realization

Wenchao Du
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213
wenchaod@cs.cmu.edu

Alan W Black
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213
awb@cs.cmu.edu

Abstract

Recent advances in deep learning have shown promises in solving complex combinatorial optimization problems, such as sorting variable-sized sequences. In this work, we take a step further and tackle the problem of ordering the elements of sequences that come with graph structures. Our solution adopts an encoder-decoder framework, in which the encoder is a graph neural network that learns the representation for each element, and the decoder predicts the ordering of each local neighborhood of the graph in turn. We apply our framework to multilingual surface realization, which is the task of ordering and completing sentences with their dependency parses given but without the ordering of words. Experiments show that our approach is much better for this task than prior works that do not consider graph structures. We participated in 2019 Surface Realization Shared Task (SR'19) , and we ranked second out of 14 teams while outperforming those teams below by a large margin.

1 Introduction

Sorting and ordering a sequence of items is a fundamental problem to computer science and artificial intelligence. When under the problem setting where a pairwise comparison function is not clear, one may wish to adopt a learning approach to rank any two elements in the sequence, and heuristically find the ordering that optimally agrees with the ranking function (Cohen et al., 1998). However, in many real world problems, the ordering of two elements may largely depend on the other items in the sequence (e.g. word order in natural languages), which makes learning a good pairwise ranking function very hard.

Recent advances in deep learning have opened many doors in solving sequence prediction problems. These neural-network-based frameworks typically involve an encoder that learns a context-sensitive representation for each element, and a decoder that predicts a probability distribution over possible outputs at each time step in an auto-regressive manner (Sutskever et al., 2014). This framework has been adapted to performing the task of sorting by using an encoder that is "order-less" and a decoder that predicts the indices of elements in the sequence (Vinyals et al., 2016).

Many important machine learning problems involves graph-structured data, such as social networks, citation networks, or parse trees in natural language processing (NLP). There has been a surging interest in modelling graphs with deep neural networks in the last few years. Unlike traditional spectral approaches that work with the spectral representations of graphs (Belkin and Niyogi, 2002), deep learning has the flexibility that it provides end-to-end solutions to much more complex problems such as graph generation and transduction.

Surface realization is a natural language generation task in which sentences are generated given input meanings. In particular, the Multilingual Surface Realization Task (Mille et al., 2019) derived inputs from universal dependency (UD) treebank (De Marneffe et al., 2014), a framework that aims to facilitate cross-lingually consistent grammatical annotations. The task consists of two tracks. The shallow track starts from UDs with word order information removed and words are lemmatized. The task consists in determining the word order and inflecting the words. The deep track further removes function words that are leaves in the dependency structures, and the task additionally consists in introducing removed function words.

In this paper, we are interested in the scenario where the sequences to be sorted have graph structures embedded. Our main contribution is a novel

Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019), pages 18–24
Hong Kong, China, November 3rd, 2019.
https://doi.org/10.18653/v1/P17

graph-to-graph learning framework for ordering graph elements. Furthermore, we evaluated our framework on a downstream task – surface realization – that has many impactful applications. We believe our approach is applicable to other problem domains.

2 Related Work

We survey the work related to our approach from the following three aspects: deep learning for combinatorial optimization, deep learning for graphs, and structured language generation.

2.1 Neural Combinatorial Optimization

Most sequence-to-sequence models only handles outputs of fixed vocabulary. Pointer Network (Vinyals et al., 2015) was first proposed for predicting the indices of elements of any given output set, and was applied to solving combinatorial problems suchs as Travelling Salesman Problem (TSP). A set-to-sequence (Vinyals et al., 2016) model was developed for investigation on the importance of order in various machine learning problems. This framework learns a holistic representation of the input set by repeatedly applying attention on the entire set. Last but not least, deep reinforcement learning has also been investigated for solving more complex combinatorial problems such as TSP (Bello et al., 2016).

2.2 Graph Neural Networks

Learning representations of graphs with deep neural networks has attracted a lot of attention in the recent years, and deep learning has achieved success in graph-related tasks such as classification (Kipf and Welling, 2017) and generation (You et al., 2018). Graph neural networks generally follow a recursive neighborhood aggregation scheme, where the embedding of each node is computed with the embeddings of its neighbors and itself. After k iterations, the embedding contains the information of its k-hop neighborhood in the graph. Besides graph representation learning, extensive research has been conducted on the transduction between sequences and graphs, including sequence-to-graph (Aharoni and Goldberg, 2017), graph-to-sequence (Beck et al., 2018), and graph-to-graph (Sun and Li, 2019) learning problems.

2.3 Structured Language Generation

Graph structures are ubiquitous in representations of natural language. Despite the success of sequence-to-sequence learning for language generation (especially machine translation), NLP researchers have started to pay a substantial amount of efforts into incorporating tree structures into neural language generation in the recent years. Application domains include machine translation (Wang et al., 2018), dialog response generation (Du and Black, 2019), and document summarization (Liu et al., 2019). Tree-based language generation models typically sequentialize the parse tree of sentences by some pre-defined traversal order, and generate the tree node in an autoregressive manner. The most common traversal orders are depth-first, left-to-right (pre-order) and breadth-first, left-to-right (level-order).

3 Methods

3.1 Problem Definition

We first give the mathematical formulation of the problem. Given graph $G = (V, E)$ where V is the vertices and E is the edges, we try to learn an ordering function $\pi : V \longrightarrow \{n \in \mathbb{N} \mid 1 \leq n \leq |V|\}$. Each vertex has multiple attributes of discrete type. Denote the attribute a of v_i by $a(v_i)$.

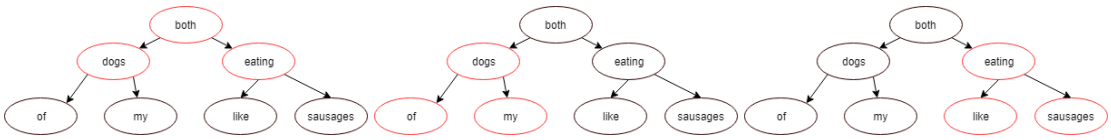
3.2 Encoder Architecture

Our encoder uses a Graph Attention Network (GAT) (Veličković et al., 2018). Attention mechanism has been widely used for handling orderless inputs, such as sets (Vinyals et al., 2016) and memory (Sukhbaatar et al., 2015). GAT learns representations for each graph node through a stack of graph attention layers. At each layer, the information of local neighborhood of each node is aggregated through attention and integrated to the embedding of the node. More specifically, let $\mathbf{g}_{il}$ be the embedding of node i at layer l. Embeddings of layer 0 are input features, which are the sum of embeddings of each node attributes:

$$\mathbf{h}_{i0} = \sum_a \mathbf{E}_{a(v_i)}$$

where $\mathbf{E}$ is the embedding matrix of attributes.

For each node i, the importance of neighbor j is computed with the scaled inner product of projected features of i and j and normalized by soft-



(a) (b) (c)

Figure 1: The order of visits for sentence "Both of my dogs like eating sausages." in our decoding algorithm. The nodes being considered for ordering are highlighted with red.

max:

$$e_{ijl} = \frac{1}{\sqrt{d}} \langle \mathbf{W}_l^k \mathbf{g}_{i,l-1}, \mathbf{W}_l^k \mathbf{g}_{j,l-1} \rangle$$

$$\alpha_{ijl} = \frac{\exp(e_{ijl})}{\sum_{j,(i,j)\in E} \exp(e_{ijl})}$$

and the local neighborhood is aggregated through the weighted sum of the embeddings. Similar to Transformer (Vaswani et al., 2017), we employ multi-head attention

$$\mathbf{g}_{il}^k = \sum_{j,(i,j)\in E} \alpha_{ijl}^k \mathbf{g}_{j,l-1}^k$$

$$\mathbf{g}'_{il} = Concat(\mathbf{g}_{il}^k)$$

where k is the index of attention heads. The attention values are transformed by a one-layered feed-forward network with residual connection.

$$\mathbf{g}_{il} = \mathbf{g}'_{il} + FFN_l(\mathbf{g}'_{il})$$

It is clear that these graph embeddings are invariant to permutation through attention mechanism. When modelling graphs with GAT, all vertices are considered adjacent to itself. Consequently, the embeddings of each node at all graph attention layers include the embedding of itself. Since the inner product between oneself is always maximal, the embeddings of the node itself will always dominate the embeddings of its neighbors when aggregating its local neighborhood. It is also clear that the embeddings at layer k represent the information of k-hop neighborhood of each node. We use the summation of embeddings from all layers as the final representation of graph nodes:

$$\mathbf{g}_i = \sum_l \mathbf{g}_{il}$$

We also tried concatenation followed by linear transform for aggregating all layers, but this approach is no better than simple summation.

Since dependency graphs are directed, it might be natural to perform *unilateral* attention (i.e. the parent uses the information of its children but not the other way around). However, we found from experiments that ignoring the directed-ness of graphs and using *bilateral* attention (i.e. the children also use the information of their parent) would perform much better than the first approach.

3.3 Decoder Architecture

At decoding stage, we order the graph elements by selecting the next element one by one. At each step of selection, we have the sequence of past elements that are already ordered available. A natural choice would be using a recurrent neural network (RNN) to encode the ordered elements. The hidden state from RNN decoder at each step would be used for selecting the next element. This idea is developed as Pointer Network (Vinyals et al., 2015). Let $\mathbf{h}_t$ be the hidden state of RNN decoder at time t, Pointer Network predicts the distribution over the next element through attention:

$$u_{it} = v^T tanh(W_1 \mathbf{e}_i + W_2 \mathbf{h}_t)$$

$$P(i \mid v_{\pi(1)} \dots v_{\pi(t)}) = softmax_i(u_{it})$$

where $\mathbf{e}_i$ is the embedding of element i. In our architecture, we propose two modifications. First, we use dot product for computing attention. Second, we add *candidate component* when computing attention.

$$u_{it} = \langle \mathbf{g}_i, W_1 \mathbf{h}_t \rangle + \langle \mathbf{g}_i, \sum_{j\in C_t} \mathbf{g}_j \rangle$$

where $C_t = \{j \mid j \notin \{\pi(1) \dots \pi(t-1)\}\}$ is the set of candidate indices that are not selected in previous steps. It is intuitive to add the embedding of candidates to bias the model towards selecting from the remaining nodes. We found this modification significantly improves the performance.

The model described above is not inherently designed to handle graphs. To incorporate graph structures in decoding, we predict the order of

nodes of each neighborhood in turn. We start with the root of the graph, and order the set consisting of the root and its neighbors. Then we recursively repeat the process for each of the children of the parent from left to right based their predicted order. An example is provided in the figure above. We also provide pseudocode for our decoding algorithm. *TreeSort* is used for arranging the order of each local neighborhood, and *TreeLinearize* is for converting the sorted tree into a sentence. The *sort* function sorts the input elements using the decoder architecture described above. It takes two arguments: the first one is the set to be sorted, and the second one is the initial state of the RNN decoder. It returns the sorted set and the last hidden state from the RNN decoder. *TreeLinearize* simply merges subtrees by interpolating spaces.

One limitation of using *TreeLinearize* is that it cannot generate non-projective trees, i.e. when nodes are put in linear order, there are edges crossing over each other. There are about 2.5% parses are non-projective in the English dataset, so the degradation in performance is negligible. In order to generate non-projective trees, one may want to use a linearization algorithm alternative to *TreeLinearize* that generates the whole sequence based on the topological order of nodes returned by *TreeSort*. At each step, the choices of nodes should be limited to those that are not preceded by any unselected nodes with higher topological order.

To see why tree decoding is advantageous over ordering the whole sequence at a time, consider the size of the search space of both approaches. The hypothesis space of sequence decoding is factorial in the number of vertices (i.e. $|V|!$). In graph decoding, at each node v, the number of nodes to be ordered is the degree of v, $d(v)$ (since the nodes to be ordered include v but not parent of v). In the example shown above, the total number of permutations of the sentence is $7! = 5040$, while with tree decoding, the number of ordering to be considered reduced to $(3!)^3 = 216$. The difference is even larger when there are more words in the sentence. So tree decoding gains performance by reducing the size of search space.

The full model is trained by maximizing the likelihood of choices of indices. We apply candidate masking, i.e. the candidates that are already selected in previous steps are assigned zero probability.

Algorithm 1 Pseudocode for tree decoding procedures.

```
procedure TREESORT(node, h)
    if node is a leaf then
        return
    else
        n ← node
        n.children ← []
        l ← n ∪ node.children
        l, h ← sort(l, h)
        node.sorted ← l
        for ch in l do
            TreeSort(ch, h)

procedure TREELINEARIZE(node)
    if node is a leaf then
        return node.word
    else
        s ← empty string
        for ch in node.sorted do
            s ← s + ' ' + TreeLinearize(ch)
        return s
```

3.4 Morphology

The MSR challenge requires realizaton of lemmatized words. Since this is not the main focus of our paper, we briefly describe our approach here. After the words are ordered into sentences in the first stage described above, we obtain BERT embeddings (Devlin et al., 2019) of each word. We train a character-level sequence-to-sequence model with attention for morphological inflection, where the source is the lemmatized word and the target is the realization. The BERT embedding is used for constructing the initial state of the decoder, so that the morphology model may use contextual information. The concatenation of decoder hidden states and embeddings of syntactic information such as tense and number is used for predicting characters.

4 Experiments

4.1 Data and Preprocessing

We use SR'19 challenge dataset (Mille et al., 2018). The data is obtained from the universal dependency treebank (Zeman et al., 2018). The dataset includes many major languages, such as English, Spanish, and Chinese. Word orders are hidden and words are randomly shuffled. Our model uses the following attributes for graph at-

Table 1: Average performance on English dataset (dev) of shallow track with different hyperparameters. All results are evaluated without morphological inflections.

	BLEU	DIST	NIST
$L = 0$	76.4	85.3	14.4
$L = 1, H = 8$	79.9	90.6	14.5
$L = 2, H = 8$	82.8	91.5	14.7
$L = 3, H = 8$	83.2	91.4	14.7
$L = 4, H = 8$	83.5	91.1	14.7
$L = 5, H = 8$	83.6	92.1	14.7
$L = 6, H = 8$	83.3	91.0	14.7
$L = 5, H = 4$	82.3	91.3	14.6
$L = 5, H = 16$	83.9	92.0	14.7

tention networks: lexicons, part-of-speech tags, types of the dependency relation with the governor, depths in the dependency graph, and relative positions to the parent. All relative positions farther than 2 are considered as one type.

4.2 Metrics

Three quantitative measure are used for evaluating performance. BLEU measures the average precisions of n-gram overlap with references. NIST is similar to BLEU but gives more weights to less frequent n-grams. DIST measures the edit distance between hypotheses and references, in which either insertion, deletion, or substitution of a word is considered an edit.

4.3 Model and Training Details

All models are implemented in PyTorch [1]. The Graph Attention Networks has output size of 512 for each attention layer, and the feedforward networks' middle layers have 1024 dimension. RNN decoders have hidden size of 512. Dropout is applied with rate 0.5. We use Adam optimizer with learning rate 0.001. We did not use learning rate warm-up as we did not find much improvement.

4.4 Main Results

We show the effects of hyperparameters of GAT on performance. We vary the number of attention layers and the number of heads in GAT. The results are summarized in Table 1. Note that these results are for hypotheses without morphological inflections, which are measured against lemmatized references. We first examine the impact of number of graph attention layers on performance. With only one layer of graph attention, performances are significantly worse than multiple layers. On the other hand, the improvement beyond using 3 layers is marginal. Without any graph attention layer, the model is essentially taking inputs as mathematical sets, in which case its performance is the worst. We are also interested in the effect of number of attention heads. It appears that the best performance is achieved with 8 heads, while attentions with 4 heads and 16 heads are slightly worse off.

Table 2: Comparison between different learning paradigms on English dev sets. All results are evaluated without morphological inflections.

	BLEU	DIST	NIST
set-to-graph (no attention)	76.4	85.3	14.4
set-to-graph (global attention)	79.9	90.6	14.5
graph-to-sequence	63.6	88.9	13.7
graph-to-graph	83.2	91.4	14.7

Table 1 shows that given local neighborhood information, the model learns more useful graph embeddings than without. We are interested in the other end of the spectrum: what if each node has global information of the set from the beginning? We apply attention over the whole input graph for each node without masking. This ignores the local structures in graphs and is essentially treating the inputs as sets. Results are listed in Table 2. It seems that learning the embeddings of set elements holistically is better than learning for each element in isolation, but still not as good as learning with graph structures.

We are also interested in the advantage of graph-to-graph decoding over graph-to-sequence. Graph-to-sequence baseline uses the graph embeddings from graph encoder, and follows the normal sequence decoding procedure. The results are shown in Table 2. The difference between graph-to-sequence and graph-to-graph is huge. Even if graph-to-sequence decoding is capable of producing non-projective trees, learning is much more difficult due to much larger hypothesis space. On the other hand, graph-to-graph decoding deals with smaller search space and exploits the hierarchical structure of sentences.

We also include the results with morphological inflections in Table 3. It seems that the morphology of English and Spanish are relatively easy,

[1] https://github.com/wenchaodudu/MSR

Table 3: Final performance on dev sets after word inflections. Results without considering inflections are parenthesized. We did not perform word inflection for Chinese datasets.

	BLEU	DIST	NIST
English	80.0 (83.6)	87.4 (92.1)	14.4 (14.7)
Chinese	66.5	63.4	12.3
Spanish	81.1 (83.2)	84.3 (85.2)	15.2 (15.4)
French	75.4 (85.5)	86.0 (88.6)	13.8 (14.8)
Japanese	67.1 (84.3)	72.5 (72.9)	10.8 (12.2)

hence the differences between the final numbers and the results without inflections are smallest. Solving morphological inflections for French is harder than English and Spanish, and Japanese is the hardest. Our approach achieved worst results on Chinese dataset. We hypothesize this is because 1) the Chinese dataset contains more non-projective trees and 2), the Chinese dataset is the smallest one comparing to other languages.

5 Conclusion

In this work, we proposed a novel graph-to-graph framework for ordering graph elements and generating sentences with projective dependency structure. Empirical results show competitive performance on surface realization task. Furthermore, exploiting graph structures is indeed helpful for such task. One future direction would be finding an end-to-end approach for jointly finishing and ordering graphs, as required in the deep track of surface realization challenge. Another direction would be finding an end-to-end approach for generating both projective and non-projective trees.

References

Roee Aharoni and Yoav Goldberg. 2017. Towards string-to-tree neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 132–140.

Daniel Beck, Gholamreza Haffari, and Trevor Cohn. 2018. Graph-to-sequence learning using gated graph neural networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 273–283.

Mikhail Belkin and Partha Niyogi. 2002. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. 2016. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

William W Cohen, Robert E Schapire, and Yoram Singer. 1998. Learning to order things. In *Advances in Neural Information Processing Systems*, pages 451–457.

Marie-Catherine De Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D Manning. 2014. Universal stanford dependencies: A cross-linguistic typology. In *LREC*, volume 14, pages 4585–4592.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Wenchao Du and Alan W Black. 2019. Top-down structurally-constrained neural response generation with lexicalized probabilistic context-free grammar. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3762–3771.

Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *ICLR*.

Yang Liu, Ivan Titov, and Mirella Lapata. 2019. Single document summarization as tree induction. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1745–1755.

Simon Mille, Anja Belz, Bernd Bohnet, Yvette Graham, and Leo Wanner. 2019. The Second Multilingual Surface Realisation Shared Task (SR'19): Overview and Evaluation Results. In *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR), 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Hong Kong, China.

Simon Mille, Anja Belz, Bernd Bohnet, and Leo Wanner. 2018. Underspecified universal dependency structures as inputs for multilingual surface realisation. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 199–209.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448.

Mingming Sun and Ping Li. 2019. Graph to graph: a topology aware approach for graph structures learning and generation. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2946–2955.

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. *ICLR*.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. 2016. Order matters: Sequence to sequence for sets. *ICLR*.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700.

Xinyi Wang, Hieu Pham, Pengcheng Yin, and Graham Neubig. 2018. A tree-based decoder for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4772–4777.

Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. 2018. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*, pages 5694–5703.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. 2018. Conll 2018 shared task: multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21.