The Concordia NLG Surface Realizer at SR'19

Farhood Farahnak and **Laya Rafiee** and **Leila Kosseim** and **Thomas Fevens**
Dept. of Computer Science and Software Engineering
Concordia University
Montreal, Canada
firstname.lastname@concordia.ca

Abstract

This paper presents the model we developed for the shallow track of the 2019 NLG Surface Realization Shared Task. The model reconstructs sentences whose word order and word inflections were removed. We divided the problem into two sub-problems: reordering and inflecting. For the purpose of reordering, we used a pointer network integrated with a transformer model as its encoder-decoder modules. In order to generate the inflected forms of tokens, a Feed Forward Neural Network was employed.

1 Introduction

The goal of Natural Language Generation (NLG) is to produce natural texts given structured data. Typically, NLG is sub-divided into two tasks: Content Planning and Surface Realization (Hovy et al., 1996; Reiter and Dale, 2000). While Content Planning focuses on selecting the most appropriate content to convey, Surface Realization produces the linear form of the text from this selected data following a given grammar.

Although the field of Natural Language Processing (NLP) has witnessed significant progress in the last few years, NLG, and surface realization in particular, still performs significantly below human performance.

Recently, several shared tasks have been proposed to improve the state of the art in specific NLG tasks (eg. Dušek et al. (2019); May and Priyadarshi (2017)). In particular, the Surface Realization Shared Task 2019 (SR'19) (Mille et al., 2019) aims to provide common-ground datasets for developing and evaluating NLG systems. Similarly to SR'18 (Mille et al., 2018), SR'19 proposed two tracks: a shallow track and a deep track. In the shallow track, unordered and lemmatized tokens with universal dependency (UD) structures (de Marneffe et al., 2014) were provided to participants and systems were required to reorder and inflect the tokens to produce final sentences. The deep track is similar to the shallow track but functional words and surface-oriented morphological information were removed as well. In addition to determining token order and inflections, systems participating in the deep track also had to determine the omitted words.

We decided to only participate in the shallow track. We used a model based on the transformer encoder-decoder architecture (Vaswani et al., 2017) combined with a pointer network (Vinyals et al., 2015) to reconstruct the word order from the input provided and a Feed Forward Neural Network to produce inflections. Based on the human evaluation, our model has an average score of 48.1% and 60.9% on all the English datasets for Readability/Quality and Meaning Similarity respectively.

2 Background

Pointer networks are types of encoder-decoder models where the output corresponds to a position in the input sequences (Vinyals et al., 2015). One of the main advantages of pointer networks compared to standard sequence-to-sequence models is that the number of output classes depends on the length of the input. This feature can be useful to address problems involving sorting variable sized sequences such as required at SR'19.

In Vinyals et al. (2015), Recurrent Neural Networks (RNNs) are used as encoder and decoder. RNNs compute the context representation based on the order of the input sequences. In cases where there is no information regarding the correct order of the input sequences, using an RNN-based encoder cannot provide a proper context representation for the decoder.

Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019), pages 63–67

https://doi.org/10.18653/v1/P17

Transformer models constitute an alternative to RNNs as they entirely rely on the self-attention mechanism (Vaswani et al., 2017). Transformer models have achieved state of the art performance in many NLP tasks such as machine translation (Vaswani et al., 2017) and language modeling (Devlin et al., 2019; Radford et al., 2019).

The encoder and decoder modules of transformer models consist of multiple layers of stacked self-attention and point-wise fully connected layers. The encoder of the transformer consists in several encoder layers, each of which is composed of two sub-layers. The first sub-layer has a multi-head attention which consists of several layers of self-attention computing on the same input, and the second sub-layer is a feed-forward network. The output of each sub-layer is added with a residual connection from their input followed by a normalization layer. The decoder module consists of several layers similar to the encoder, where the decoder layers have an extra sub-layer of encoder-decoder attention.

Transformer models have no information regarding the order of the input sequence. Hence "Mary killed John" and "John killed Mary" have the same internal representations. To alleviate this issue, Vaswani et al. (2017) considered using positional encoding summed to the embedding of each word. Because the transformer without positional encoding does not rely on the order of the input sequence, this architecture constitutes a promising option for the SR'19 where the correct order of the input sequence were removed (see Section 3).

3 Dataset

For the shallow track, training and development sets were provided for 11 different languages. These were taken from the Universal Dependency (UD) datasets (de Marneffe et al., 2014). The correct token order within the sentences was removed by shuffling the tokens. In total, 7 features were provided by the organizers. Out of these features, `FEATS` contained more than 40 morphological sub-features from the universal feature inventory and the relative linear order with respect to the governor (`Lin`). Table 1 lists the 8 features used by our model: 6 features of the UD structure, in addition to 2 features for `Lin` (the `Lin` feature divided into its absolute value and its sign).

In particular, we worked only on the English datasets, which consists of four training and development pairs. We concatenated all four training sets (`en_ewt-ud-train`, `en_gum-ud-train`, `en_lines-ud-train` and `en_partut-ud-train`) into a single one containing 19,976 sentences, with the longest sentence containing 209 words.

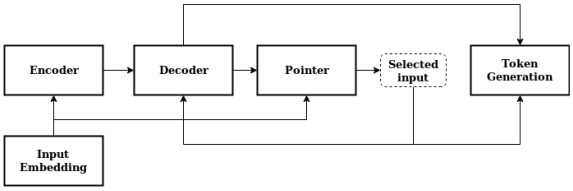

Figure 1: The model architecture used for the shallow track at SR'19

Because the development sets provided by the SR'19 organizers are not labeled, we divided the training data in two parts; training ($18,000$ sentences) and validation ($1,967$ sentences). We removed all sentences longer than 100 tokens for efficiency reasons.[1]

4 Model

Inspired by previous work from SR'18 (Mille et al., 2018) that used pointer networks to reconstruct unordered sentences (Elder and Hokamp, 2018), we developed a similar model using a pointer network integrated with a transformer as its encoder and decoder. As shown in Figure 1, our model is composed of five modules: input embedding, encoder, decoder, pointer, and token generation. In the following, we describe each module in more detail.

4.1 Input embedding

In order to train the model, we embedded each feature separately into vectors and then concatenated them. Table 1 indicates the embedding size of each feature. For the token embeddings, we employed the GloVe pretrained embeddings of size 300 (Pennington et al., 2014), while the remaining feature embeddings are trained from scratch. At the end, the concatenated vector (of size 393) is linearly mapped into the desired embedding size (512 in our case, see Section 5).

4.2 Encoder

The embedded input is fed into the encoder to compute its representation. The encoder mod-

[1] This removed 9 sentences from the training sets.

#	Feature	Feature description	Embedding size
1	Token	Lemma or stem of word form	300
2	UPOS	Universal part-of-speech tag	10
3	XPOS	Language-specific part-of-speech tag	10
4	Deprel	Universal dependency relation to the Head	10
5	Head	Head of the current word	20
6	Index	Word index	20
7	Lin	Relative linear order with respect to the governor	20
8	Lin sign	The sign of the Lin feature	3
All		Concatenation of all features	393

Table 1: The 8 features used in our model with their corresponding embedding sizes

ule uses the transformer architecture described in Vaswani et al. (2017). Since the input data does not provide any ordering information, we directly feed the embedded input into the encoder without summing it with the positional encoding of the tokens.

4.3 Decoder

The decoder in the transformer model receives the previously generated tokens alongside the encoded representation from the encoder as its input to generate the next token. However, since our task is to produce an ordering rather than generate tokens, we decided to feed the same embeddings used for the encoder (see Section 4.1) in its correct order. Since the correct order for the previously generated tokens is determined in the decoding phase, we add the positional encoding with the embedded input.

4.4 Pointer

To find the next token, we deploy the attention mechanism described in Vaswani et al. (2017) and in Equation 1.

$$\text{Attention}(Q, V, K) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

In each decoding step, the keys (K) and values (V) come from the embedded input and the query (Q) is defined by the output of the decoder (in this setup the keys and query have a dimension of d_k). The pointer selects the most probable embedded input as the next input to the decoder. During test time, we mask out the previously selected embedded inputs so that they are not selected again.

4.5 Token generation

The Pointer Network orders the given lemmatized tokens based on the input features. However, the desired output should be the inflected form of the input tokens. To this end, the token generation module (see Figure 1) is designed to generate the inflected form of the tokens, where its input is the concatenation of the selected embedded input token with the decoder's output. The output of this module is the probability over all the words in the vocabulary. This module consists of two feed-forward layers with a ReLU activation function. The last layer is initialized with pretrained GloVe embeddings in order to provide a better generalization on unseen tokens.

5 Experiments and Results

5.1 Model Configuration

The model submitted to SR'19, under the team name CLaC, has the following configuration optimized on the validation set.

The encoder and decoder of the model have 4 layers each with 8 heads (number of attention in each layer). All the embedding sizes of the encoder and decoder layers as well as input embedding are set to 512.

To preserve the GloVe embeddings, we froze the weights of both token embeddings and the last layer of the token generation module.

We suspected that the Head and Index features (see Table 1) constituted valuable information regarding the dependency tree structure of the sentences. Therefore, to ensure the model does not memorize the actual value of these features, but rather their relationship, in each training iteration, we randomly changed the values of these features while keeping the tree structure relationship intact.

The model was trained using two cross-entropy loss functions: order loss for the pointer and token loss for the generation module. We observed training with the order loss and then fine tuning on both losses increased the performance of the model.

The final model trained for 60 epochs, where training on the order loss is done for 30 epochs, and fine tuning the order and token losses were

#		Dataset	Tokenized			Detokenized		
			BLEU	NIST	DIST	BLEU	NIST	DIST
1	In-domain	en_ewt-ud-test	**22.08**	9.77	45.99	14.62	7.21	44.7
2		en_gum-ud-test	15.32	8.64	38.13	10.55	6.53	36.97
3		en_lines-ud-test	15.30	8.23	40.40	9.81	6.10	39.08
4		en_partut-ud-test	10.07	7.14	36.21	7.45	5.57	35.32
5	Out-of-domain	en_pud-ud-test	12.36	8.83	36.26	9.11	6.66	35.23
6	Predicted	en_ewt-Pred-HIT-edit	21.21	9.69	43.59	14.14	7.23	42.41
7		en_pud-Pred-LATTICE	12.89	8.82	36.67	9.29	6.69	35.53

Table 2: Results of our submission in the shallow track task of SR'19

done on the remaining 30 epochs. The initial learning rate was set to 1×10^{-4}. We also took advantage of learning rate decay with the factor of 0.5 when there is no improvement on the validation loss. A dropout rate of 0.3 was used on the encoder, decoder, and input embedding module.

The model was implemented using the PyTorch 1.1 framework. For the transformer encoder and decoder, we modified the fairseq transformer implementation of Ott et al. (2019).

In the test phase, we used tokens generated by the model as the final output. When encountering unknown tokens, the model uses the input token where the pointer points at.

5.2 Results

At SR'19, three types of test sets were given: In-domain, Out-of-domain, and Predicted. The In-domain datasets share the same domain as the training data, while Out-of-domain dataset does not. The Predicted datasets are those where the annotation were built using parser outputs from the Universal Dependency Parsing shared task 2018 (Zeman et al., 2018) instead of the gold syntactic annotations. Evaluation was performed in both a tokenized and detokenized fashion.

Table 2 shows the results of our model on the test data. As shown in Table 2, our model achieved its highest performance on the en_ewt-ud-test dataset with a BLEU score of 22.08 for tokenized among the In-domain datasets. Whereas the lowest score is for en_partut-ud-test with a BLEU of 10.07. Clearly, the performance of the model on these four datasets is directly related to the relative size of corresponding training set in the concatenated training set used to train our model. For example, the en_ewt-ud-train dataset accounts for the greatest proportion of our training set (63%) and achieves the highest BLEU; whereas en_partut-ud-train accounts for only 9% of the training samples yielding the lowest BLEU of 10.07 with en_partut-ud-test.

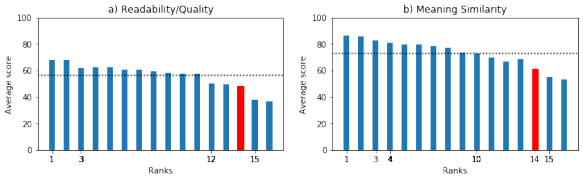

Figure 2: Human evaluation results compared to all participants of the shallow track at SR'19

Based on human evaluation, our submitted system achieved average Readability/Quality score of 48.1% and a Meaning Similarity score of 60.9% with the rank of 12 and 14 respectively among the 16 participating systems. The results are shown in the Figure 2.

6 Conclusions

In this paper, we have presented the model we developed for SR'19. The proposed system is composed of a pointer network where its encoder and decoder modules borrowed from transformer, aim to reconstruct the tokens' order and inflection. The model achieved its best performance on the English datasets with the average scores of 48.1 and 60.9 for the Readability/Quality and Meaning Similarity respectively. Although this performance is lower than expected, the lack of training data is observable. It was noticeable that training the model in an end-to-end fashion without feature engineering could not lead the model to learn meaningful representation of the input features.

As future work, it would be interesting to investigate further the model's sensitivity to the training size, as we noted in our submission's results, by training it on a much larger dataset. We will also further investigate the use of other features provided in the universal dependency structure. Another avenue worth looking into is the use of pre-trained language models. Finally, a thorough error analysis would provide us with hints as to where

the model is weaker, in the ordering task or in the inflection task.

Acknowledgements

The authors would like to thank the anonymous reviewers for their comments on an earlier version of this paper.

This work was financially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

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 (NAACL/HLT 2019)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Ondřej Dušek, Jekaterina Novikova, and Verena Rieser. 2019. Evaluating the state-of-the-art of end-to-end natural language generation: The E2E NLG Challenge. *arXiv preprint arXiv:1901.11528*.

Henry Elder and Chris Hokamp. 2018. Generating high-quality surface realizations using data augmentation and factored sequence models. In *Proceedings of the First Workshop on Multilingual Surface Realisation*, pages 49–53, Melbourne, Australia. Association for Computational Linguistics.

Eduard Hovy, Gertjan van Noord, Guenter Neumann, and John Bateman. 1996. Language generation. *Survey of the State of the Art in Human Language Technology*, pages 131–146.

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 *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*, pages 4585–4592, Reykjavik, Iceland. European Languages Resources Association (ELRA).

Jonathan May and Jay Priyadarshi. 2017. SemEval-2017 task 9: Abstract meaning representation parsing and generation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 536–545, Vancouver, Canada. Association for Computational Linguistics.

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, Tilburg University, The Netherlands. Association for Computational Linguistics.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL/HLT 2019): Demonstrations*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

Ehud Reiter and Robert Dale. 2000. *Building Natural Language Generation Systems*. Cambridge University Press, New York, NY, USA.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pages 5998–6008. Curran Associates, Inc.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer Networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, pages 2692–2700. Curran Associates, Inc.

Daniel Zeman, Filip Ginter, Jan Hajič, Joakim Nivre, Martin Popel, and Milan Straka. 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*, Brussels, Belgium. Association for Computational Linguistics.