Can the Transformer Learn Nested Recursion with Symbol Masking?

Jean-Philippe Bernardy **Adam Ek** **Vladislav Maraev**
Centre for Linguistic Theory and Studies in Probability
Department of Philosophy, Linguistics and Theory of Science
University of Gothenburg
firstname.lastname@gu.se

Abstract

We investigate if, given a simple symbol masking strategy, self-attention models are capable of learning nested structures and generalise over their depth. We do so in the simplest setting possible, namely languages consisting of nested parentheses of several kinds. We use encoder-only models, which we train to predict randomly masked symbols, in a BERT-like fashion. We find that the accuracy is well above random baseline, with accuracy consistently above 50% both when increasing nesting depth and distances between training and testing. However, we find that the predictions made correspond to a simple parenthesis counting strategy, rather than a push-down automaton. This suggests that self-attention models are not suitable for tasks which require generalisation to more complex instances of recursive structures than those found in the training set.

1 Introduction

Self-attention models (Vaswani et al., 2017) enjoy broad use in NLP tasks. The best attention-based models can tackle several tasks using a unified sentence encoding (and perhaps decoding) module (Raffel et al., 2020), with applications ranging from classification to inference and generation. They provide state of the art results for all such tasks, displacing the already very successful recurrent neural networks, in particular the LSTM and its variants. The availability of large pretrained models (Devlin et al., 2019) is another strong point in their favour.

However, the generalisation capabilities of self-attention models are still not well understood, and the present work is part of an ongoing effort to understand their capabilities. We study in particular their ability to learn context-free languages, which are characterised by the nested structures. For this purpose, we control the inputs to the model to the maximum, while focusing on the defining characteristic of context-free languages, namely matching opening and closing brackets. This corresponds to learning *generalised Dyck languages* (see table 2). In particular, we investigate the following questions:

1. Can self-attention generalise to matching open/close parenthesis at longer distances?
2. Can self-attention generalise to matching open/close parenthesis at deeper nesting levels distances?

There is a already a small body of work dealing with this question (see sec. 5), but our contribution is specific in the following two respects: i) We use the popular BERT-like training regime (predict a percentage of randomly masked tokens), ii) We concentrate on generalising to (much) deeper nesting.

Beyond theoretical considerations, matching brackets have applications in the NLP-style treatment of constructed languages (in particular) programming languages, for example translating between programs and their natural language descriptions.

2 Data Sets

We define the language $\mathcal{D}_n$ as the set of strings generated by the following context-free rules: $E ::= \varepsilon; E ::= EE; E ::= oEc$, where (o, c) stands for a pair of matching parenthesis pairs. The index n stands for the number of possible pairs. In all of our tests, we will use $n = 5$ (corresponding for example to the pairs `()` , `[]`, `{}`, <> and «»), and thus we drop the subscript from now on.

We are interested in various characteristics of the strings of $\mathcal{D}$. First, we consider the distance between a closing parenthesis and the cor-

Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, pages 753–760

responding opening parenthesis. Given a string s of length $2N$ (N is the number of matching pairs), we will call $\delta(s)$ an array of length $2N$ such that if s_i is a closing parenthesis, $\delta(s)_i$ is the distance between s_i and the closing parenthesis. If s_i is an opening parenthesis, $\delta(s)_i$ is 0. For example, if $s =$"{ ( ) < [ ] (« ») > }", $\delta(s) = [0, 0, 1, 0, 0, 1, 0, 0, 1, 3, 9, 11]$. The second characteristic that we consider is the amount of nesting between closing and opening parentheses. We call this characteristic η(s), and likewise we define it for each closing parenthesis, and let it be zero for opening parentheses.

For example, if $s =$"{ ( ) < [ ] (« ») > }", $\eta(s) = [0, 0, 1, 0, 0, 1, 0, 0, 1, 2, 3, 4]$.

To generate a string with N matching pairs, we perform a random walk between opposite corners of a square grid of width and height N, such that one is not allowed to cross the diagonal. When not restricted by the boundary, a step can be taken either along the x or y axis with equal probability. A step along the x axis corresponds to open a parenthesis, and one along the y axis corresponds to closing one. The kind of parenthesis pair is chosen randomly and uniformly. We call the distribution of input strings sampled by this procedure D. In all our experiments we set $N = 10$ (which is enough to illlustrate our points) and we thus omit the superscript in what follows.

We also want control the maximum distance between opening and closing parentheses (so that we never train on too long distances). We do so by discarding elements s of D such that $\delta(s)_i > d$ for some i, and call the resulting distribution $D[MaxDist = d]$.

Often we want to control the maximum depth that our model is trained or tested on. For this purpose, we generate strings s which exhibit at least one index i such that $\eta(s)_i = d$, but no index j such that $\eta(s)_j > d$. These paths can be generated by constraining the path on the grid to touch a diagonal at distance d to the origin diagonal, and we call the corresponding distribution $D[MaxDepth = d]$.

3 Model and masking strategy

We implement a variation of the transformer model as introduced by (Vaswani et al., 2017). In the model each input symbol is associated with a vector embedding of size K. A sequence of opening and closing brackets is represented by a matrix of size (N, K).

Following Devlin et al. (2019), our model then applies a series of multi-head self-attention layers organised in a hierarchical structure, such that the second layer operates on the representations generated in the first layer, and so on. We use a BERT-like, non auto-regressive architecture: each layer attends to every position in the input, including itself. Then a softmax classifier is employed to predict the symbol at the current position. Hence, we use a masking strategy to train and test the model (otherwise it could simply use the current symbol for prediction).

For training, we follow the masking strategy presented by Devlin et al. (2019). We mask 15% of the closing parenthesis tokens at random, where in 80% of the cases we replace the token with a mask token, in 10% of the cases with a random token, and in the remaining 10% of the cases we replace it with the same token.

For testing, after sampling a string s, we pick a random position i such that s_i is a closing parenthesis. Then we mask all subsequent symbols, and let the model predict s_i. There is a single possible closing parenthesis type for s_i, corresponding to the opening parenthesis found earlier in the string. The prediction is considered successful if the model predicts the right type of closing parenthesis.

4 Experiments & Results

Our experiments consists in training the language model for a limited version of the Dyck family (for example by limiting nesting depth (η) or maximum distance (δ)), and testing what the performance is in a more general case. Thus, because there are five types of parenthesis pairs in all our experiments, the random baseline is $\frac{1}{5} = 20\%$.

4.1 Generalisation to Longer Distances

In the first experiment we investigate whether the model is capable of predicting closing parenthesis at long distance from the corresponding opening parentheses, whereas it has only seen short-distances in the training data. More precisely, we train the model on strings from $D[MaxDist = 9]$ and test it on $D[MaxDist = 19]$.

We present an overview of the results in table 1. Our experiments show that the (2 layers, 8 heads) model generalises the best. Using fewer heads appears to be more detrimental to the model's accuracy than the number of layers. This is true even though the (8,2) model has many more parameters

Table 1: Mean accuracy and standard deviation over 10 runs on generalisation to longer distances for each model configuration.

Layers	Heads	Accuracy
4	4	0.814(± 0.013)
8	2	0.643(± 0.005)
2	8	0.844(± 0.008)

than the (2,8) model (see appendix).

The aggregated numbers however hide much of the reality of the generalisation capabilities as a function of distance. Therefore we further break down the accuracy by distance to the corresponding opening parenthesis in figure 1. The (8,2) model fails to learn parenthesis matching at short distances, but its accuracy is better for longer distances. In contrast the (4,4) and (2,8) models do well for adjacent parentheses, but their accuracy drops quickly until reaching a minimum at distance 13, dipping below 50% accuracy —however still above chance. Perhaps surprisingly, all models do very well at very long distances. These very long distances correspond to matching parentheses at the beginning of the input with parentheses at the end (that is, when we mask the fewest number of input symbols).

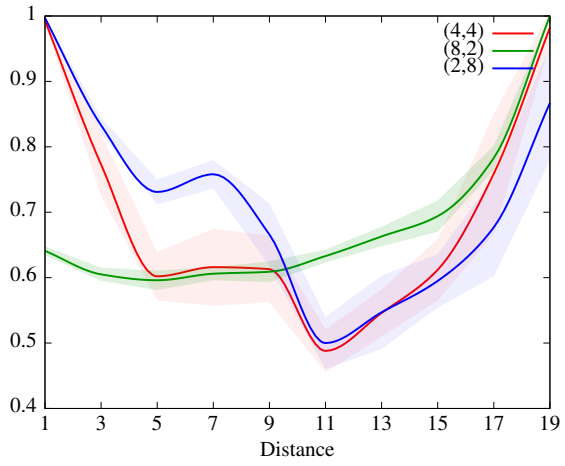

Figure 1: Mean model accuracy for closing parenthesis depending on a distance to corresponding opening parenthesis, over 10 runs. Shaded areas correspond to standard deviation.

4.2 Generalisation to Deeper Nesting

In the second experiment we test whether the model can generalise to deeper nesting depths. That is, we train the model on $D[MaxDepth = 3]$ and test it on $D[MaxDepth = 9]$

Table 2: Mean accuracy and standard deviation over 10 runs on generalisation to deeper nesting for each model configuration.

Layers	Heads	Accuracy
4	4	0.654(± 0.012)
8	2	0.518(± 0.005)
2	8	0.672(± 0.008)

We present an overview of the results in table 2. Looking at the results we see a similar pattern in terms of aggregated accuracy as in the previous experiment: the (2,8) setup performs the best, followed by (4,4) and finally (8,2). Breaking down accuracy by nesting depth (figure 2) reveals that the difference resides chiefly in the (8,2) model failing to predict shallow nesting.

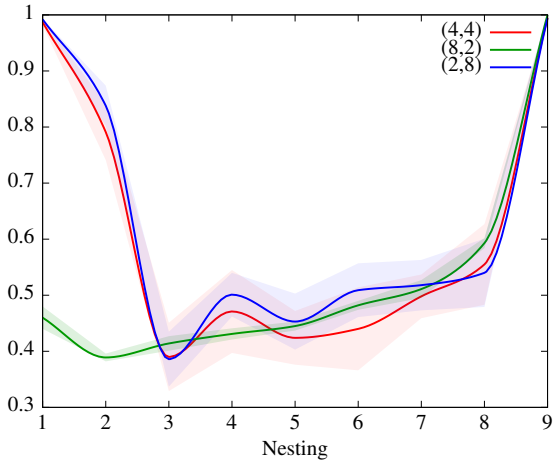

Figure 2: Mean model accuracy for closing parenthesis depending on a distance to corresponding opening parenthesis over 10 runs. Shaded areas correspond to standard deviation.

4.3 Analysis of attention heads

We have analysed attention heads by manual inspection of softmax score for attention heads for each layer, on several sequence from our training set (see Appendix for the corresponding heat maps).

Looking at the behaviour of the attention heads we note that the first layer in the (2,8) and (4,4) models focuses its attention on the previous symbol. Then, in the final layer of the (2,8) model the attention of the start of the sequence focuses on the end, and vice-versa.

In the (4,4) model, the second layer appears to often focus on the non-masked symbols while in the third layer the attention is distributed more evenly between masked and non-masked symbols. A no-

table feature of the third layer is that a lot of self-attention occurs on the masked symbols. In the final layer, the attention of all symbols is put almost exclusively on the masked symbols.

The (8,2) model is the only model which does not have a clear layer that looks at the preceding token. It appears that in the (8,2) model, the earlier layers focus their attention on the beginning of the sequence, then it moves towards the latter part of the sequence. The heat maps also show that the (8,2) model focuses heavily on certain symbols, which are the least frequent symbols used in the sequence, for later layers. In earlier layers the model appears to focus on the frequent symbols. This analysis is compatible with the (8,2) model using a symbol counting method.

In summary, the (4,4) model appears to first look at the previous symbol in the sequence. There are two steps of searching where first the model ignores the masked symbols and distributes the attention over the other symbols. In the second step, the model again focuses all around the sequence, but the masked symbols receive a lot of attention. For the (2,8) model, the behaviour is more straightforward. First it looks at the previous symbol, then all around the sequence. To the best of our knowledge, the (8,2) model is counting symbols by distributing its attention on frequent and less frequent symbols.

5 Related work

Studying the ability of language models to learn Dyck languages is emerging as a standard way to test the ability to generalise to deeper nesting levels. Before self-attention, this test was applied to RNNs. Bernardy (2018) proposed non-standard stack-based RNN models, which can approach perfect accuracy for generalised Dyck-language, although the accuracy of standard RNNs was higher than random but far from perfect. Hewitt et al. (2020) presented a theoretical proof that RNNs are able to learn Dyck languages with maximum nesting depth m using $O(m)$ memory. Sennhauser and Berwick (2018) present contrasting evidence, concluding that LSTMs can learn very limited range of rules.

A number of studies have considered self-attention models, especially in the past year. Ebrahimi et al. (2020) investigated self-attention models using Dyck languages, and claimed that self-attention models with a starting symbol are able to generalise to longer sequences and deeper structures without learning recursion, as competitive LSTM models do. In contrast to us, they studied models trained autoregressively only. Bhattamishra et al. (2020) studies how autoregressive Transformer architecture learns a subset of formal languages, including Dyck language and its generalisations. In contrast to our study, they examine Shuffle-Dyck languages, which allows constructions like "([ )]" and provide theoretical and experimental evidence that the Transformer is capable of learning such a language. On the other hand, Hahn (2020) points at the limitation of using self-attention models. He indicates that in theory the LSTM should perform better than the autoregressive Transformer, because the transformer cannot emulate a stack, general finite-state automata, or use recursion.

6 Conclusion and future work

Our experiments show that, with a random masking strategy, the transformer is able to discover a way to make good predictions when generalising to longer distances and deeper nesting. However, this strategy is not using the history of opening and closing parentheses in a way a push-down automaton would.

Indeed, the analysis reveals that the best accuracy is obtained when few symbols have been masked. This can be explained by the model having learned a counting strategy. When a single symbol is masked, predicting the kind of missing parenthesis can be done by subtracting the number of closing parentheses by the number of opening parentheses for each type, and predict the type which exhibits a discrepancy. For short distances our (2,8) and (4,4) models were able to learn to remember preceding symbols and act accordingly. We suspect that for intermediate levels of nesting and distance, the models act according to a mixture of the above two strategies.

In consequence, we recommend not to use a BERT-like masking strategy for applications where generalising to longer distances or deeper nesting is critical. Rather, auto-regressive models should be used, such as auto-regressive attention or RNNs.

Acknowledgments

The research reported in this paper was supported by grant 2014-39 from the Swedish Research Council, which funds the Centre for Linguistic Theory and Studies in Probability (CLASP) in the Depart-

ment of Philosophy, Linguistics, and Theory of Science at the University of Gothenburg.

References

Bernardy, J.-P. (2018). Can recurrent neural networks learn nested recursion? In *Linguistic Issues in Language Technology, Volume 16, 2018*. CSLI Publications.

Bhattamishra, S., Ahuja, K., and Goyal, N. (2020). On the Ability and Limitations of Transformers to Recognize Formal Languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116, Online. Association for Computational Linguistics.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (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, Minneapolis, Minnesota. Association for Computational Linguistics.

Ebrahimi, J., Gelda, D., and Zhang, W. (2020). How can self-attention networks recognize Dyck-n languages? In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4301–4306, Online. Association for Computational Linguistics.

Hahn, M. (2020). Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171.

Hewitt, J., Hahn, M., Ganguli, S., Liang, P., and Manning, C. D. (2020). RNNs can generate bounded hierarchical languages with optimal memory. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1978–2010, Online. Association for Computational Linguistics.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Sennhauser, L. and Berwick, R. (2018). Evaluating the ability of LSTMs to learn context-free grammars. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 115–124, Brussels, Belgium. Association for Computational Linguistics.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

A Experimental setup and reproducibility information

Our implementation is based on pytorch, with a custom re-implementation of the transformer architecture, exactly following (Vaswani et al., 2017). The runtime is under one day for the whole set of experiments using a Titan X (Pascal) GPU.

The hyperparameters we use are listed in table 3.

Table 3: Hyperparameters used and the number of data examples used.

Parameter	Value
Optimiser	Adam
Learning rate	0.0001
Epochs	10
Batch size	512
Training examples	102400
Validation examples	20480

In our experiments we consider three different transformer architectures, corresponding to different values for the number of multi-head self-attention layers, and the size of the heads. Specifically, we considers the setups presented in section 4

Table 4: Model configurations and the number of parameters in each configuration

Layers	Heads	Parameters
8	2	897 292
4	4	1 191 820
2	8	1 781 452

In each case, we have used 64-dimensional embeddings throughout the models.

B Attention heat-maps

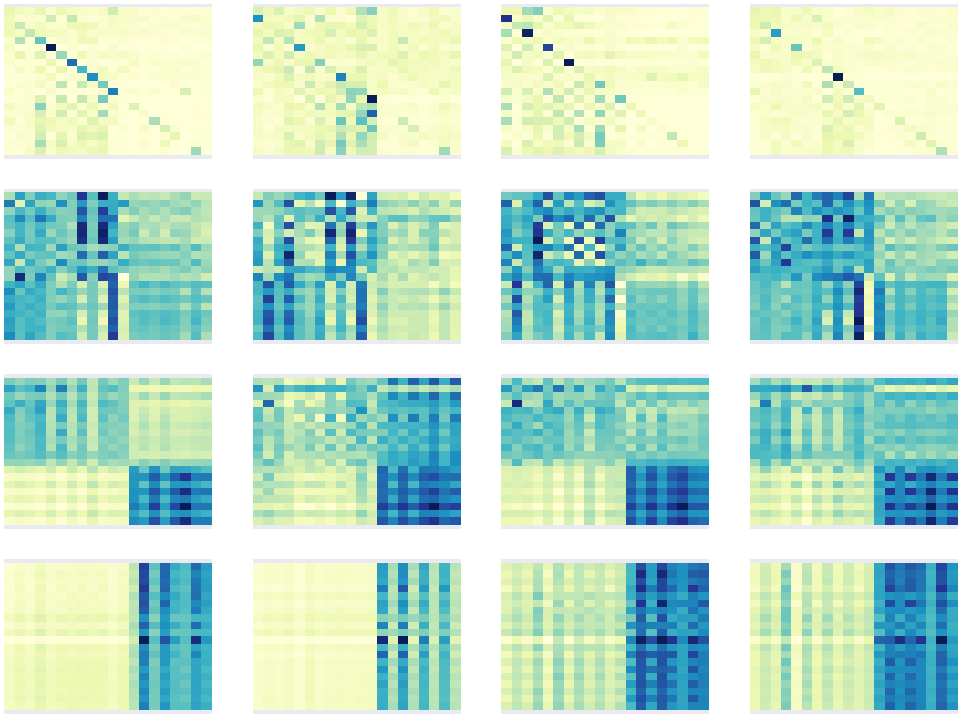

Figure 3: Attention heatsmaps for the model with 4 heads and 4 layers on the input +-+<[+[([()])]-]>-.

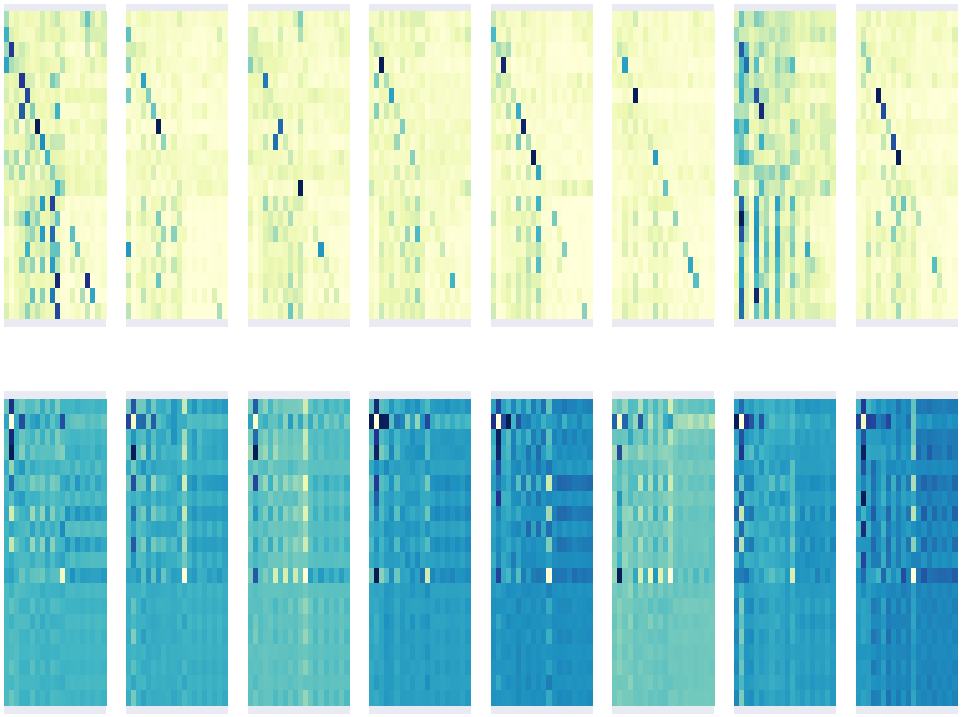

Figure 4: Attention heatsmaps for the model with 2 heads and 8 layers on the input +-+<[+[([()])]-]>-.

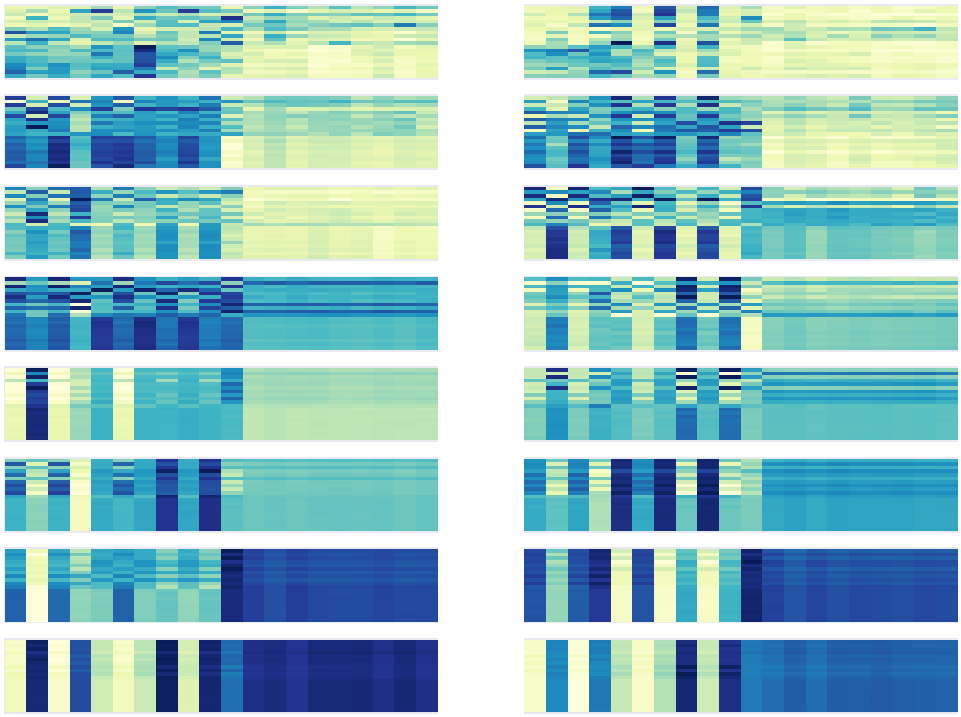

Figure 5: Attention heatsmaps for the model with 8 heads and 2 layers on the input +-+<[+[([()])]-]>-.