Learning Adaptive Control Flow in Transformers for Improved Systematic Generalization

Anonymous Author(s)
Affiliation
Address
email

Abstract

Despite successes across a broad range of applications, Transformers have limited capability in systematic generalization. The situation is especially frustrating for algorithmic tasks, where they often fail to find intuitive solutions that can be simply expressed in terms of attention patterns. Here we propose two modifications to the Transformer architecture, copy gate and geometric attention, which facilitate learning such intuitive and interpretable solutions to algorithmic problems. Our novel Transformer, called Transformer Control Flow (TCF) achieves 100% length generalization accuracy on the classic compositional table lookup task. The resulting attention and gating patterns are interpretable, demonstrating that the model implements adaptive control flow.

1 Introduction

Systematic generalization [1] is a property of learning systems that enable them to generalize beyond their training distribution, by learning compositional rules that are generally applicable. Examples include generalization to sequences much longer than those seen during training, and algorithmic combinations of known rules. Despite recent efforts [2, 3, 4, 5, 6, 7], systematic generalization generally remains unsolved [8, 9, 10, 11, 12]. On some datasets, the best performing models are neuro-symbolic hybrids [13, 14] which make use of task specific symbolic functions. However, their applicability to other datasets remains limited [15, 16].

Typically many NNs solve the training set, but only few of them generalize to systematically different test sets. A big question is: which type of external pressure or architectural inductive bias would encourage the training process to select a "real" solution which systematically generalizes? While some approaches such as meta-optimization [17] address this issue, the gain is typically limited.

The popular Transformer [18] architecture also suffers from such problems [19, 7, 20]. Frustratingly, Transformers fail to generalize in many algorithmic tasks (e.g. [10, 21]), even those that have intuitive solutions which can be simply expressed in terms of attention patterns in Transformers. Indeed, given an input sequence of length N and a Transformer encoder of depth T, solving an algorithmic task is often all about routing the right information to the right node/operation at the right time in the T-by-N grid represented by Transformer columns. Effectively the task is to learn to draw an *adaptive control flow* on the canvas of Transformer columns. In fact, a recent work by Weiss et al. [22] introduced a programming language called RASP which is specifically designed to express solutions for sequence processing problems, and which has a direct equivalent to the operations in Transformers. However, it is also shown that making Transformers learn a solution expressed in RASP is only possible with intermediate supervision of attention patterns. In some cases, even such supervision fails [22]. In general, Transformers fail to find solutions of algorithmic tasks which are easily interpretable and/or symbolic. We conversely hypothesize that attention-based NNs able to find intuitive solutions (achieving interpretable attention patterns) can improve systematic generalization.

Submitted to 35th Conference on Neural Information Processing Systems (aiplans 2021). Do not distribute.

Here we argue that regular Transformers lack a few basic ingredients for learning such "intuitive" solutions to algorithmic problems. We propose simple architectural modifications to help them learn data routing and easily interpretable solutions. As a first step towards validating our model, we focus on the popular length generalization task of compositional table lookup (CTL) [10, 23, 21]. Our model achieves 100% generalization accuracy on this task, which is previously unsolved by NNs [21], and we show that the resulting attention patterns are interpretable as plausible control flows.

2 Improving Transformers for learning adaptive control flow

We argue that the following components are needed to build Transformers capable of learning adaptive control flow. **First**, composing known operations in an arbitrary order requires that all operations are available at every computational step. This can be easily achieved by sharing the weights of the layers, as is done in Universal Transformers [24]. **Second**, the network should be sufficiently deep, at least as deep as the deepest data dependency in the computational graph (e.g., in case of a parse tree, this is the depth of the tree). Otherwise, multiple operations would be fused into a single layer and hinder natural and elegant compositions. **Third**, inputs in some columns should be kept unchanged until the right point in time when they can be processed in a later step/layer. The regular Transformer lacks a mechanism for skipping the whole transformation step by simply copying the input to the next step/layer. We propose a special gating function, *copy gate*, to implement such a mechanism (Sec. 2.1). **Finally**, many algorithmic tasks require combining a number of local computations in the right order. This typically implies that at a given time, attention should not focus on all possible matches, but only on the closest match. We propose and investigate a new type of attention with the corresponding inductive bias (Sec. 2.2), which we call *geometric attention*. We refer to the resulting new Transformer as Transformer Control Flow (TCF).

2.1 Copy gate: learning to skip operations (vertical flow)

Each layer of the regular Transformer consists of one self-attention and one feedforward block. The input to each of these blocks are directly connected to the corresponding output via a residual connection [25, 26]. However, such a connection does not allow for skipping the transformation of the entire layer and simply passing the unchanged input to the next layer. Here we propose to add an explicit gate, which we call *copy gate*, to facilitate such a behavior.

We consider a T-layer Transformer encoder and an input sequence of length N. Since each layer corresponds to one *computational step*, we often refer to a layer as a step t. We denote the Transformer state of column i in layer t as $\boldsymbol{h}^{(i,t)} = \mathbf{H}_{t,i} \in \mathbb{R}^d$ where d is the state size, and $\mathbf{H}_t \in \mathbb{R}^{N \times d}$ denotes the states of all N columns in layer t. In the copy gate-augmented Transformer, each column i in layer $(t+1)$ processes the input $\mathbf{H}_t$ like in the regular Transformer:

$$\boldsymbol{a}^{(i,t+1)} = \mathrm{LayerNorm}(\mathrm{MultiHeadAttention}(\boldsymbol{h}^{(i,t)}, \mathbf{H}_t, \mathbf{H}_t) + \boldsymbol{h}^{(i,t)}) \tag{1}$$

$$\hat{\boldsymbol{h}}^{(i,t+1)} = \mathrm{LayerNorm}(\mathrm{FFN}^{\mathrm{data}}(\boldsymbol{a}^{(i,t+1)})) \tag{2}$$

but the output is gated as:

$$\boldsymbol{g}^{(i,t+1)} = \sigma(\mathrm{FFN}^{\mathrm{gate}}(\boldsymbol{a}^{(i,t+1)})) \tag{3}$$

$$\boldsymbol{h}^{(i,t+1)} = \boldsymbol{g}^{(i,t+1)} \odot \hat{\boldsymbol{h}}^{(i,t+1)} + (1 - \boldsymbol{g}^{(i,t+1)}) \odot \boldsymbol{h}^{(i,t)} \tag{4}$$

We use the basic two-layer feedforward block [18] for both $\mathrm{FFN}^{\mathrm{data}}$ and $\mathrm{FFN}^{\mathrm{gate}}$ which transforms input $\boldsymbol{x} \in \mathbb{R}^d$ to:

$$\mathrm{FFN}(\boldsymbol{x}) = \boldsymbol{W}_2 \max(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1, 0) + \boldsymbol{b}_2 \tag{5}$$

but with separate parameters and different dimensionalities: for $\mathrm{FFN}^{\mathrm{data}}$ $\boldsymbol{W}_1^{\mathrm{data}} \in \mathbb{R}^{d_{\mathrm{FF}} \times d}$, $\boldsymbol{W}_2^{\mathrm{data}} \in \mathbb{R}^{d \times d_{\mathrm{FF}}}$, while for $\mathrm{FFN}^{\mathrm{gate}}$ $\boldsymbol{W}_1^{\mathrm{gate}}, \boldsymbol{W}_2^{\mathrm{gate}} \in \mathbb{R}^{d \times d}$, with biases $\boldsymbol{b}_1^{\mathrm{data}} \in \mathbb{R}^{d_{\mathrm{FF}}}$ and $\boldsymbol{b}_2^{\mathrm{data}}, \boldsymbol{b}_1^{\mathrm{gate}}, \boldsymbol{b}_2^{\mathrm{gate}} \in \mathbb{R}^d$.

When the gate is closed i.e. $\boldsymbol{g}^{(i,t+1)} = 0$, the entire transformation is skipped and the input is copied over to the next layer $\boldsymbol{h}^{(i,t+1)} = \boldsymbol{h}^{(i,t)}$. Crucially, we parameterize the gate (Eq. 3) as a function of the output of the self-attention (Eq. 1), such that the decision to copy or to transform the input for each column depends on the states of all columns. This is a crucial difference compared to previously proposed gating in Transformers which are solely motivated by the training stability [27] or a common practice from convolution based models [19].

2.2 Geometric attention: learning to attend to the closest match (horizontal flow)

We propose *geometric attention* designed to attend to the closest matching element. Like in regular self-attention, given an input sequence $[\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(N)}]$ with $\boldsymbol{x}^{(i)} \in \mathbb{R}^{d_{\text{in}}}$, each input is projected to key $\boldsymbol{k}^{(i)} \in \mathbb{R}^{d_{\text{key}}}$, value $\boldsymbol{v}^{(i)} \in \mathbb{R}^{d_{\text{value}}}$, query $\boldsymbol{q}^{(i)} \in \mathbb{R}^{d_{\text{key}}}$ vectors, and the dot product is computed for each key/query combination. In our geometric attention, the dot product is followed by a sigmoid function to obtain a score between 0 and 1:

$$\boldsymbol{P}_{i,j} = \sigma(\boldsymbol{k}^{(j)\top} \boldsymbol{q}^{(i)}) \tag{6}$$

which will be treated as a probability of the key at (source) position j matching the query at (target) position i. These probabilities are finally converted to the attention scores $\boldsymbol{A}_{i,j}$ as follows:

$$\boldsymbol{A}_{i,j} = \boldsymbol{P}_{i,j} \prod_{k \in \mathbb{S}_{i,j}} (1 - \boldsymbol{P}_{i,k}) \tag{7}$$

where $\mathbb{S}_{i,j}$ denotes the set of all (source) indices which are closer to i than j to i, and when two indices have the same distance to i, we consider the one which is to the right of i (i.e., greater than i) to be closer, i.e.,

$$\mathbb{S}_{i,j} = \begin{cases} k \in \{1, ..., N\} \setminus \{i, j\} : |i - k| \leq |i - j|, & \text{if } i < j \\ k \in \{1, ..., N\} \setminus \{i, j\} : |i - k| < |i - j|, & \text{if } j < i \end{cases} \tag{8}$$

In addition, we explicitly zero-out the diagonal by setting $\boldsymbol{A}_{i,i} = 0$ for all $i = 1, ..., N$. The ordering of source indices is illustrated in Figure 1/Right. The resulting scores $\boldsymbol{A}_{i,j}$ are the attention scores used to compute the weighted average of the value vectors.

By using the term $(1 - \boldsymbol{P}_{i,k})$ in Eq. 7, when there is a match, it downscales any other matches which are more distant. Two recent works [28, 29] use such a parameterized geometric distribution in the form of Eq. 7. In practice, Eq. 6 is augmented by an additional *direction encoding* (See Appendix B).

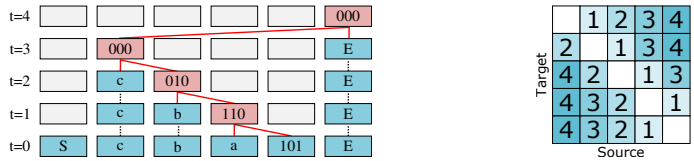

Figure 1: Left: an ideal sequence of computations in a Transformer for an example CTL task. Right: the order of source positions for each target, counting from the closest, used in geometric attention.

3 Experiments

Task. Compositional table lookup task [10, 23, 21] is constructed from a set of symbols and unitary functions defined over these symbols. Each example in the task is defined with one input symbol and a list of functions to be applied sequentially, i.e. the first function is applied to the input symbol, and the resulting output becomes the input to the second function, and so forth. There are 8 possible symbols. Each symbol is traditionally represented by a 3-bit bitstring [10], however, in practice, they are simply processed as one token [21]. The functions are bijective and randomly generated. Each function is represented by a letter. An example input is '`101 d a b`', which corresponds to the expression $b(a(d(101)))$; the model has to predict the correct output symbol. We note that there exists a sequence-to-sequence variant of this task [21] where the model has to predict all intermediate steps (thus trained with intermediate supervision). We directly predict the final output. An ideal model should be able to solve this task independently of the presentation order, i.e., whether the task is encoded as '`101 d a b`' or '`b a d 101`'. We thus study both forward (former) and backward (latter) variants of the task. To evaluate systematic generalization, the train/valid/test sets have different numbers of compositions: samples of up to 5/6-8/9-10 operations respectively.

Results. We consider four different baselines: an LSTM [30], DNC [31, 32], Universal Transformers [18, 24] and its variants with relative positional encodings [7]. For Transformers, the prediction is based on the last column in the final layer. Results are shown in Table 1. The LSTM and DNC

perform well in the forward variant, achieving perfect generalization for longer sequences, but fail on the backward variant. In contrast, basic Transformers fail in both cases.

By introducing the copy gate (Sec. 2.1), the relative Transformer is able to solve the forward, but not the backward task. Further analysis showed that the network learns to attend to the last operation based on the relative position information. Since the result is read from the last column, this position changes with the sequence length. The model thus fails to generalize to such arbitrary offsets. To remediate this issue, we introduce a simple mechanism to let the model choose between absolute and relative positional encodings at each position (see Appendix A). The resulting model effectively manages to make use of the absolute position for the prediction, and to perform well in both directions. However, such a combination of absolute/relative positional encoding might be an overly specific bias. A more generic solution, geometric attention (Sec. 2.2), also achieves perfect generalization, and was found easier to train.

Table 1: Performance of different models on CTL dataset

	IID		Longer	
Model	Forward	Backward	Forward	Backward
LSTM	1.00 ± 0.00	0.59 ± 0.03	1.00 ± 0.00	0.22 ± 0.03
DNC	1.00 ± 0.00	0.57 ± 0.06	1.00 ± 0.00	0.18 ± 0.02
Transformer	1.00 ± 0.00	0.82 ± 0.39	0.13 ± 0.01	0.12 ± 0.01
+ rel	1.00 ± 0.00	1.00 ± 0.00	0.23 ± 0.05	0.13 ± 0.01
+ rel + gate	1.00 ± 0.00	1.00 ± 0.00	0.99 ± 0.01	0.19 ± 0.04
+ abs/rel + gate	1.00 ± 0.00	1.00 ± 0.00	0.98 ± 0.02	0.98 ± 0.03
+ geom. att.	0.96 ± 0.04	0.93 ± 0.06	0.16 ± 0.02	0.15 ± 0.02
+ geom. att. + gate (TCF)	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00

Analysis. Figure 2 shows the gating and attention patterns of our model with both copy gate and geometric attention for an example of the backward presentation task. As shown in Fig. 2/Right, the gates of different columns open sequentially one after another, when the input is available for them. Fig. 2/Left shows the attention map. Each column attends to the neighboring one, waiting for its computation to be finished. The behavior of the last column is different: it always attends to the second position of the sequence, which corresponds to the last operation to be performed.

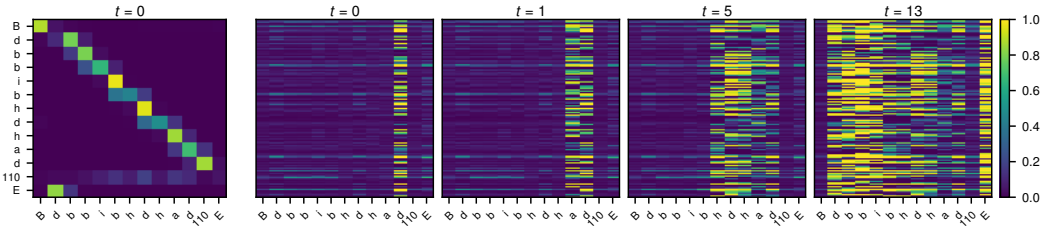

Figure 2: Example visualization for TCF. For other models, see Appendix D. Left: Attention map for step $t = 0$. The x/y-axis corresponds to source/target positions respectively. The attention map remains similar over time. Each position focuses on the column to the right, except the last one where the result is read from, which focuses on the last operation. Right: gate activations for different steps/layers. The gates remain closed until the data dependencies are satisfied.

4 Conclusion

We proposed a new view on the internal operations of Transformers, introducing a dynamic dataflow architecture between Transformer columns. This overcomes two shortcomings of traditional Transformers: the problem of keeping and routing data in unaltered fashion, which we solve by an additional copy gate, and the problem of learning length-independent attention patterns, which we solve by geometric attention. Our new model, TCF generalizes to longer lengths on the popular compositional lookup table task in both forward and backward directions. The gates and the attention maps collectively make the architecture more interpretable compared to the baselines.

References

[1] Jerry A Fodor, Zenon W Pylyshyn, et al. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71, 1988.

[2] Dzmitry Bahdanau, Harm de Vries, Timothy J O'Donnell, Shikhar Murty, Philippe Beaudoin, Yoshua Bengio, and Aaron Courville. CLOSURE: Assessing systematic generalization of CLEVR models. In *ViGIL workshop, NeurIPS*, Vancouver, Canada, December 2019.

[3] Kris Korrel, Dieuwke Hupkes, Verna Dankers, and Elia Bruni. Transcoding compositionally: Using attention to find more generalizable solutions. In *Proc. BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, ACL*, pages 1–11, Florence, Italy, 2019.

[4] Brenden M Lake. Compositional generalization through meta sequence-to-sequence learning. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pages 9788–9798, Vancouver, Canada, December 2019.

[5] Yuanpeng Li, Liang Zhao, Jianyu Wang, and Joel Hestness. Compositional generalization for primitive substitutions. In *Proc. Conf. on Empirical Methods in Natural Language Processing and Int.Joint Conf. on Natural Language Processing (EMNLP-IJCNLP)*, pages 4292–4301, Hong Kong, China, November 2019.

[6] Jake Russin, Jason Jo, Randall C O'Reilly, and Yoshua Bengio. Compositional generalization in a deep seq2seq model by separating syntax and semantics. *Preprint arXiv:1904.09708*, 2019.

[7] Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. The devil is in the detail: Simple tricks improve systematic generalization of transformers. In *Proc. Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, Punta Cana, Dominican Republic, November 2021.

[8] Jerry Fodor and Brian P McLaughlin. Connectionism and the problem of systematicity: Why smolensky's solution doesn't work. *Cognition*, 35(2):183–204, 1990.

[9] Brenden M. Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 2873–2882, Stockholm, Sweden, July 2018.

[10] Adam Liska, Germán Kruszewski, and Marco Baroni. Memorize or generalize? searching for a compositional RNN in a haystack. In *AEGAP Workshop ICML*, Stockholm, Sweden, July 2018.

[11] Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. On the binding problem in artificial neural networks. *Preprint arXiv:2012.05208*, 2020.

[12] Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, pages 757–795, 2020.

[13] Xinyun Chen, Chen Liang, Adams Wei Yu, Dawn Song, and Denny Zhou. Compositional generalization via neural-symbolic stack machines. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Virtual only, December 2020.

[14] Qian Liu, Shengnan An, Jian-Guang Lou, Bei Chen, Zeqi Lin, Yan Gao, Bin Zhou, Nanning Zheng, and Dongmei Zhang. Compositional generalization by learning analytical expressions. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Virtual only, December 2020.

[15] Daniel Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärli. Compositional generalization in semantic parsing: Pre-training vs. specialized architectures. *Preprint arXiv:2007.08970*, 2020.

[16] Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? *Preprint arXiv:2010.12725*, 2020.

[17] Henry Conklin, Bailin Wang, Kenny Smith, and Ivan Titov. Meta-learning to compositionally generalize. In *Proc. of Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing, ACL-IJCNLP*, pages 3322–3335, Virtual only, August 2021.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 5998–6008, Long Beach, CA, USA, December 2017.

[19] Rahma Chaabouni, Roberto Dessì, and Eugene Kharitonov. Can transformers jump around right in natural language? assessing performance transfer from scan. *Preprint arXiv:2107.01366*, 2021.

[20] Santiago Ontañón, Joshua Ainslie, Vaclav Cvicek, and Zachary Fisher. Making transformers solve compositional tasks. *Preprint arXiv:2108.04378*, 2021.

[21] Yann Dubois, Gautier Dagan, Dieuwke Hupkes, and Elia Bruni. Location attention for extrapolation to longer sequences. In *Proc. Association for Computational Linguistics (ACL)*, pages 403–413, Virtual only, July 2020.

[22] Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers. In *Proc. Int. Conf. on Machine Learning (ICML)*, volume 139, pages 11080–11090, Virtual only, July 2021.

[23] Dieuwke Hupkes, Anand Singh, Kris Korrel, German Kruszewski, and Elia Bruni. Learning compositionally through attentive guidance. In *Proc. Int. Conf. on Computational Linguistics and Intelligent Text Processing*, La Rochelle, France, April 2019.

[24] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *Int. Conf. on Learning Representations (ICLR)*, New Orleans, LA, USA, May 2019.

[25] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 2368–2376, Montreal, Canada, December 2015.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, June 2016.

[27] Emilio Parisotto, H. Francis Song, Jack W. Rae, Razvan Pascanu, Çaglar Gülçehre, Siddhant M. Jayakumar, Max Jaderberg, Raphaël Lopez Kaufman, Aidan Clark, Seb Noury, Matthew Botvinick, Nicolas Heess, and Raia Hadsell. Stabilizing transformers for reinforcement learning. In *Proc. Int. Conf. on Machine Learning (ICML)*, volume 119, pages 7487–7498, Virtual only, July 2020.

[28] Ethan A. Brooks, Janarthanan Rajendran, Richard L. Lewis, and Satinder Singh. Reinforcement learning of implicit and explicit control flow instructions. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 1082–1091, Virtual only, July 2021.

[29] Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. *Preprint arXiv:2107.05407*, 2021.

[30] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, pages 1735–1780, 1997.

[31] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwinska, Sergio Gomez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John P. Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626): 471–476, 2016.

[32] Róbert Csordás and Jürgen Schmidhuber. Improving differentiable neural computers through memory masking, de-allocation, and link distribution sharpness control. In *Int. Conf. on Learning Representations (ICLR)*, New Orleans, LA, USA, May 2019.

[33] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proc. Association for Computational Linguistics (ACL)*, pages 2978–2988, Florence, Italy, 2019.

[34] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Int. Conf. on Learning Representations (ICLR)*, New Orleans, LA, USA, May 2019.

[35] Stephen José Hanson. A stochastic version of the delta rule. *Physica D: Nonlinear Phenomena*, 42(1-3):265–272, 1990.

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

A Details of attention with absolute/relative positional encoding combination

The addition of copy gates enables Transformers to generalize to longer lengths in the forward presentation order, but not in the backward order. Examining the attention maps reveals that the model uses position-based attention for reading out the result, as opposed to content-based attention. In the backward presentation order, the last column of the transformer should focus on the second column, whose relative position changes dynamically with the length of the sequence. We solve this issue by adding an option to the attention head to choose between absolute and relative positional encodings.

We use the relative positional embedding variant of self-attention Dai et al. [33]. In what follows, we describe the operation within a single layer/step. We thus omit the layer/step index t for better readability, and denote the state at column/position i as $\boldsymbol{h}_i$ instead of $\boldsymbol{h}^{(i,t)}$. Our attention matrix is decomposed as follows:

$$r_i = \sigma(\boldsymbol{h}_i \boldsymbol{W}_{ar} + b_{ar}) \tag{9}$$

$$\hat{\boldsymbol{A}}_{i,j} = \underbrace{\boldsymbol{h}_i^\top \boldsymbol{W}_q^\top \boldsymbol{W}_{k,E} \boldsymbol{h}_j}_{(a)} + \underbrace{\boldsymbol{u}^\top \boldsymbol{W}_{k,E} \boldsymbol{h}_j}_{(c)} + \big(\underbrace{\boldsymbol{h}_i^\top \boldsymbol{W}_q^\top \boldsymbol{W}_{k,P}}_{(b)} + \underbrace{\boldsymbol{v}^\top \boldsymbol{W}_{k,P}}_{(d)} \big) \underbrace{\big(\boldsymbol{p}_{i-j} r_i + \boldsymbol{p}_j (1 - r_i) \big)}_{(e)} \tag{10}$$

where the matrix $\boldsymbol{W}_q \in \mathbb{R}^{d_{\text{head}} \times d}$ maps the states to queries, $\boldsymbol{W}_{k,E} \in \mathbb{R}^{d_{\text{head}} \times d}$ maps states to keys, while $\boldsymbol{W}_{k,P} \in \mathbb{R}^{d_{\text{head}} \times d}$ maps positional embedding to keys. d_{head} is the size of the key, query and value vectors for each head, set as $d_{\text{head}} = \frac{d}{n_{\text{heads}}}$. $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^{d_{\text{head}}}$ are learned vectors. Softmax is applied to the second dimension of $\hat{\boldsymbol{A}}$ to obtain the final attention scores, $\boldsymbol{A}$. Component (a) corresponds to content-based addressing, (b, e) to content based positional addressing, (c) represents a global content bias, while (d, e) represents a global position bias.

We introduce the term (e) for the positional embedding which can switch between absolute and relative positional encodings using the gate r_i (Eq. 9; parameterized by $\boldsymbol{W}_{ar} \in \mathbb{R}^{d \times 1}$ and $b_{ar} \in \mathbb{R}$), which is the function of the state at target position i. $\boldsymbol{p}_i \in \mathbb{R}^d$ is a sinusoidal embedding for position i, which can be can be both positive and negative. Following Vaswani et al. [18], we scale $\boldsymbol{A}_{i,j}$ by $\frac{1}{\sqrt{d_{\text{head}}}}$ prior to applying the softmax, and we define $\boldsymbol{p}_{i,j}$ as:

$$\boldsymbol{p}_{i,j} = \begin{cases} \sin(i/10000^{2j/d}), & \text{if } j = 2k \\ \cos(i/10000^{2j/d}) & \text{if } j = 2k+1 \end{cases} \tag{11}$$

B Details of geometric attention

In Sec. 2.2, we introduce a basic form of geometric attention (Eqs. 6-8). In practice, Eq. 6 can be augmented by an additional *directional encoding*. In fact, the only positional information available in the geometric attention presented above is the ordering used to define the product in Eqs. 7-8. In practice, we found it crucial to augment the score computation of Eq. 6 with an additional *directional information*. The directional information we introduce is encoded as a scalar $\boldsymbol{D}_{i,j}$ for each target/source position pair (i, j) computed as:

$$\boldsymbol{D}_{i,j} = \begin{cases} \boldsymbol{W}_{\text{LR}} \boldsymbol{h}_i + b_{\text{LR}}, & \text{if } i \leq j \\ \boldsymbol{W}_{\text{RL}} \boldsymbol{h}_i + b_{\text{RL}}, & \text{if } i > j \end{cases} \tag{12}$$

where $\boldsymbol{h}_i \in \mathbb{R}^d$ denotes the input/state at position i and $\boldsymbol{W}_{\text{LR}}, \boldsymbol{W}_{\text{RL}} \in \mathbb{R}^{1 \times d}$, $b_{\text{LR}}, b_{\text{RL}} \in \mathbb{R}$ are trainable parameters. This directional information is integrated into the score computation of Eq. 6 as follows (akin to how Dai et al. [33] introduce the relative positional encoding as an extra term in the computation of attention scores):

$$\boldsymbol{P}_{i,j} = \sigma\big(\alpha\big(\boldsymbol{h}_i \boldsymbol{W}_q + \boldsymbol{u}\big)^\top \boldsymbol{W}_{k,E} \boldsymbol{h}_j + \beta \boldsymbol{D}_{i,j} + \gamma\big) \tag{13}$$

where the matrix $\boldsymbol{W}_q \in \mathbb{R}^{d_{\text{head}} \times d}$ maps the states to queries, $\boldsymbol{u} \in \mathbb{R}^{d_{\text{head}}}$ is a bias for queries, $\boldsymbol{W}_{k,E} \in \mathbb{R}^{d_{\text{head}} \times d}$ maps states to keys (we note that d_{head} is typically the size of the key, query and

value vectors for each head, $d_{\text{head}} = \frac{d}{n_{\text{heads}}}$), and $\alpha, \beta, \gamma \in \mathbb{R}$ are learned scaling coefficients and bias, initialized to $\alpha = \frac{1}{\sqrt{d_{\text{head}}}}, \beta = 1, \gamma = 0$. Using this additional directional information, each query (position i) can potentially learn to restrict its attention to either the left or right side.

The proposed attention function has a complexity of $O(N^2)$, similar to the one of regular self-attention. Eq. 7 can be implemented easily in a numerically stable way in log-space. Products can then be calculated using cumulative sums, subtracting the elements for the correct indices in each position.

C Implementation details

A PyTorch implementation of our models together with the experimental setup is available under `https://github.com/after_review`. The performance of all models are reported as mean and standard deviation over 5 different seeds.

C.1 Dataset details

Our implementation uses 8 symbols (all 3 bit patterns) as input arguments and 9 randomly sampled bijective functions denoted by small letters of the English alphabet. All functions are included in the train set in combination with all possible input symbols. The rest of the training set consists of random combinations of functions applied to a random symbol as an argument, up to length 5. The total size of the train set is 53,704 samples. The samples are roughly balanced such that there are similar number of samples for each depth. There are different validation sets: an IID, which matches the distribution of the train set, and a depth validation, which includes samples of length 6, 7 and 8. The test set consists of sequences of length 9 and 10.

C.2 Model details

We use the AdamW optimizer [34] for all of our models. Standard hyperparameters are listed in Tab. 2. Additionally, models with gating use dropout [35, 36] of 0.1 applied to the content-based query and the position-query components. In the case of geometric attention, since the channels of the positional encoding does not have any redundancy, dropout is applied just to the content-query.

In the case of Transformers with the copy gate but without geometric attention, we use $\tanh$ instead of $\mathrm{LayerNorm}$ in Eq. 2.

The hyperparameters of the gateless Transformers differ significantly from the gated ones. This is because they are very hard to train to achieve good performance even on the IID set, and they needed extensive hyperparameter optimisation. One might argue that fewer layers (11 instead of 14) makes them less competitive on longer sequences, however, we were unable to train them to perform well even on the IID data with 14 layers. Similarly, increasing the size of the baseline Transformer models made them fail to train.

All Transformer variants have a begin (B) and end (E) token included on sequence boundaries. RNNs (LSTM and DNC) have no such addition. All Transformers are encoder-only, and the result is read from the last column (corresponding to the end token). A single feedforward layer maps it to the 8 classes corresponding to the 8 possible output symbols.

The DNC has 21 memory cells, 4 read heads and an LSTM controller, and contains recently introduced improvements [32].

Bias of the gates $\boldsymbol{b}_2^{\text{gate}}$ are initialized to -3. This ensures that initially no updates happen, and creates a better gradient flow between layers.

We use gradient clipping with magnitude 5 for all of our models.

Hyperparameters were obtained by a Bayesian hyperparameter search of Weights & Biases[1] over the systematically different validation set for the `+abs/rel + gate` models and was reused for all other gated models. For the non-gated models we used the `+rel` variant for tuning. This failed to yield to

[1] `https://wandb.ai/`

any usable configurations, so we used the mixture of the IID and systematically different validation set to obtain the parameters in Tab. 2.

We train all models to a fixed number of n_{iters} iterations and measure its validation performance every 1000 iterations. For each model, we select the best checkpoint according to the validation performance, and report its test accuracy.

Table 2: Hyperparameters used for different models. We denote the feedforward size as d_{FF}, weight decay as "wd.", dropout as "do.". The model is trained for n_{iters} iterations.

	d_{model}	d_{FF}	n_{heads}	n_{layers}	batch s.	learning rate	wd.	do.	n_{iters}
LSTM	200	-	-	1	256	$10 * 10^{-4}$	-	0.5	20k
DNC	200	-	-	1	256	$10 * 10^{-4}$	-	0.5	20k
Transformer	128	256	4	11	512	$1.5 * 10^{-4}$	0.0025	0.1	30k
+ rel	128	256	4	11	512	$1.5 * 10^{-4}$	0.0025	0.1	30k
+ rel + gate	256	512	1	14	512	$2 * 10^{-4}$	0.01	0.5	30k
+ abs/rel + gate	256	512	1	14	512	$2 * 10^{-4}$	0.01	0.5	30k
+ geom. att.	128	256	4	11	512	$1.5 * 10^{-4}$	0.0025	0.1	30k
+ geom. att. + gate	256	512	1	14	512	$1.5 * 10^{-4}$	0.01	0.5	30k

D Additional analysis

In the main text, we only had space to show the gate and attention activity of a gated network with absolute/relative positional encodings in Figure 4 and Fig. 5.

Here we show the corresponding visualization for the geometric attention variant in Figures 6 and 7, as well as the attention map for baseline Transformer with relative positional encoding in Figure 3. Please directly refer to the caption of the figures for the corresponding analysis. In general, the visualization for our geometric model variant is interpretable unlike that of the baseline Transformer model.

One might wonder how geometric attention is able to focus next to its diagonal, if there is no positional embedding. By combining direction sensitivity with the bias, it is possible to focus on such elements. Even in cases where this is not possible, there might be alternative solutions relying solely on direction and content.

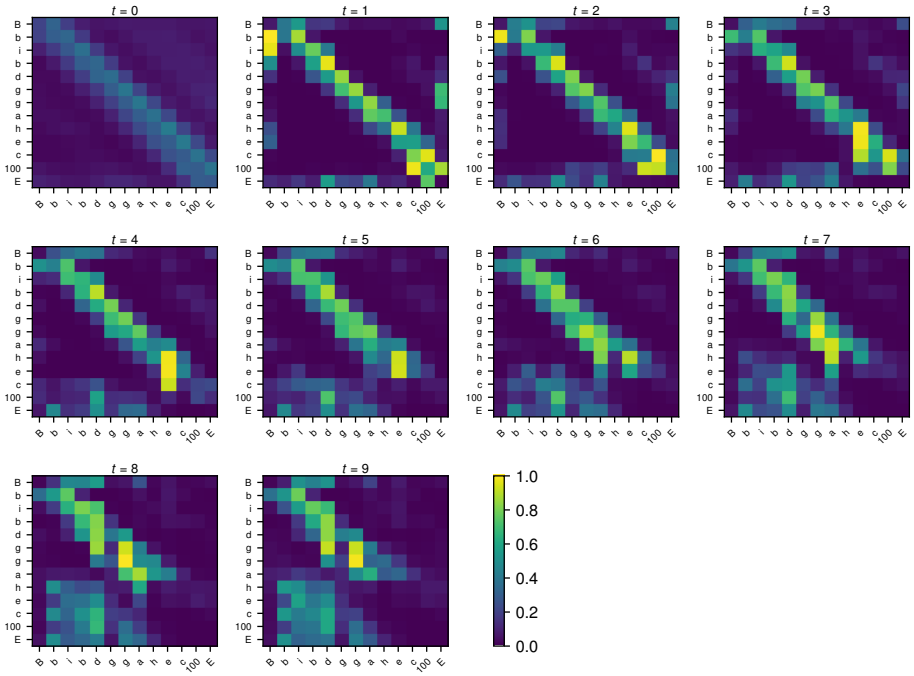

Figure 3: Attention map for every computation step for a baseline Transformer with relative positional encoding. The attention pattern gets blurry very quickly and the network does not generalize to longer sequences.

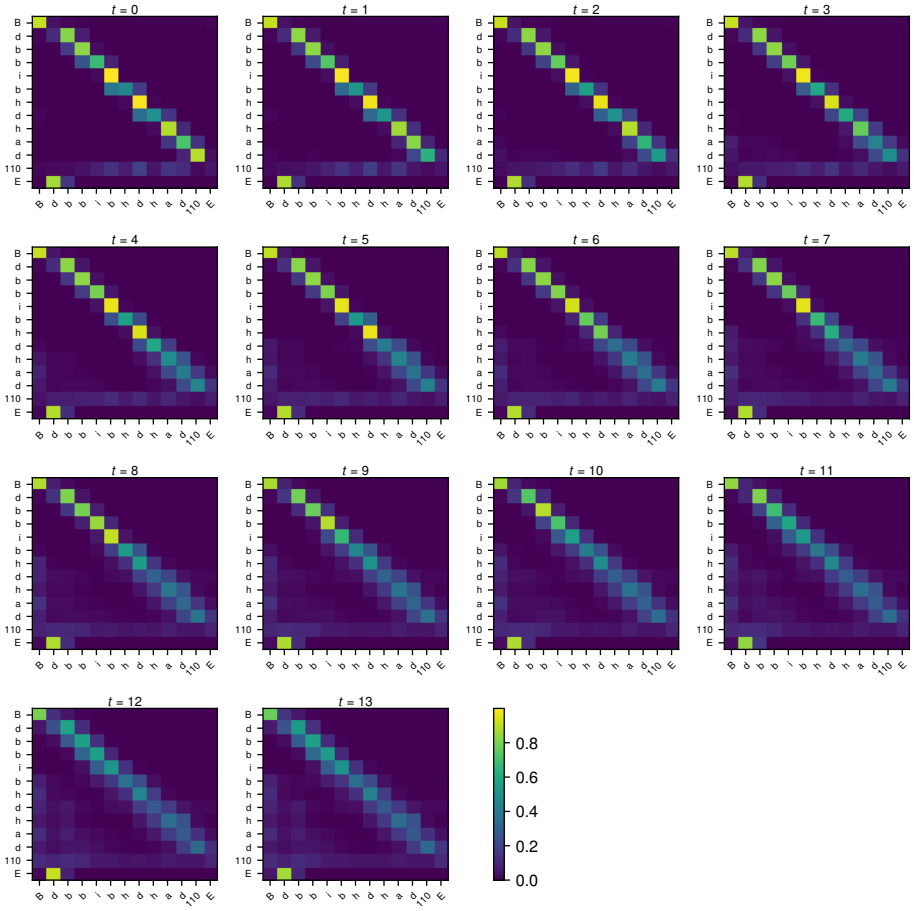

Figure 4: Attention map for every computation step for a Transformer with gating and relative/absolute positional encoding (presented in Figure 2). The attention pattern is relatively stable over time, and its gets blurrier after the given column is processed and updated. The gate sequence for the same input can be seen on Figure 5

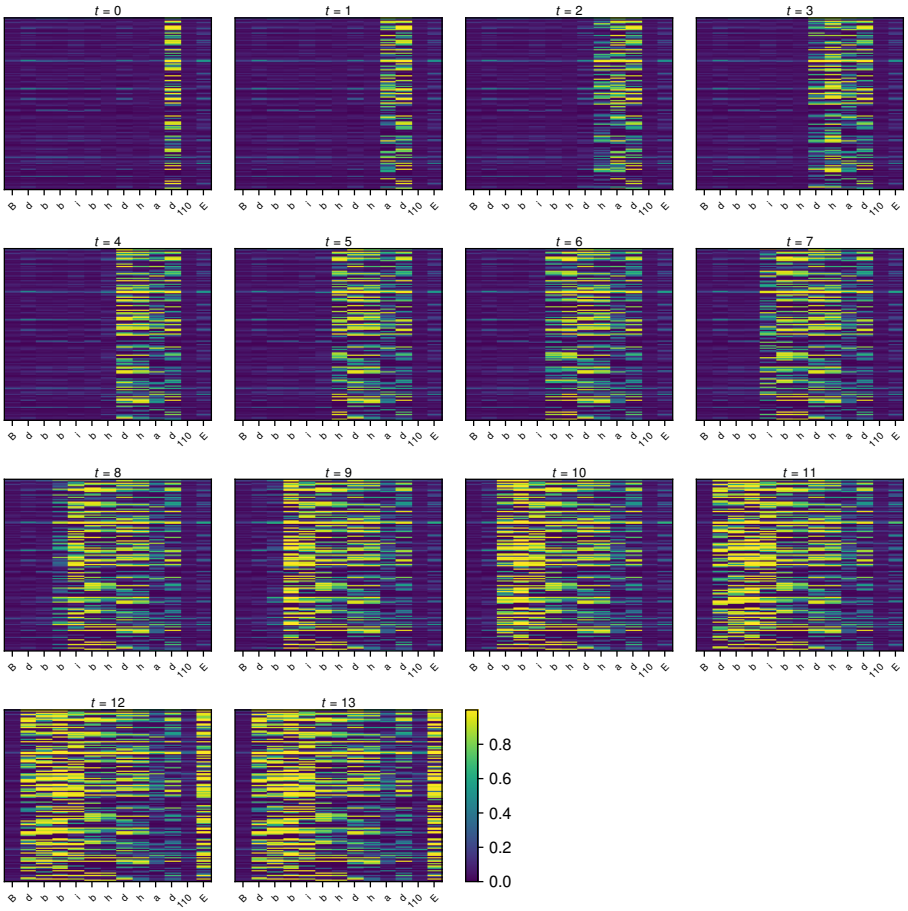

Figure 5: Gates for every computation step for a Transformer with gating and relative/absolute positional encoding. The gates are closed until all arguments of the given operations are not ready. The attention maps for the same input can be seen on Figure 4.

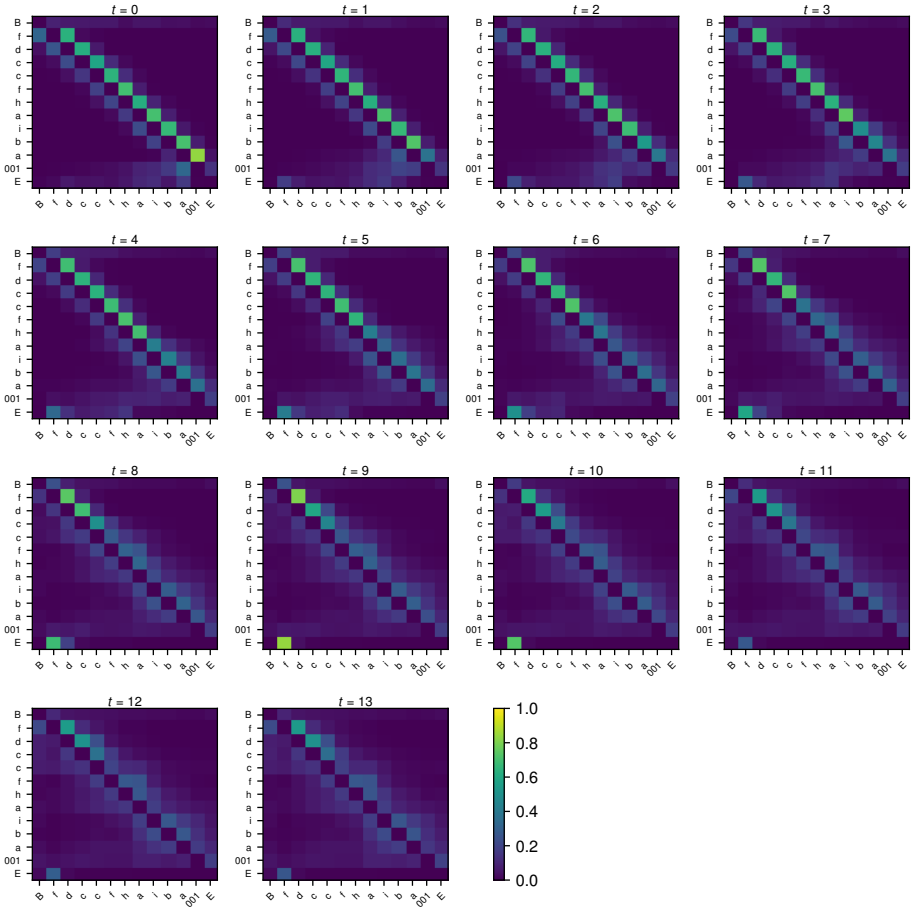

Figure 6: Attention map for every computation step for a Transformer with geometric attention and copy gate (TCF). It can be seen that the network correctly and clearly focuses on the last element of the sequence and the last sharp read happens in step 10 - corresponding to the 10 function calls in the example. The gate sequence for the same input can be seen in Figure 7

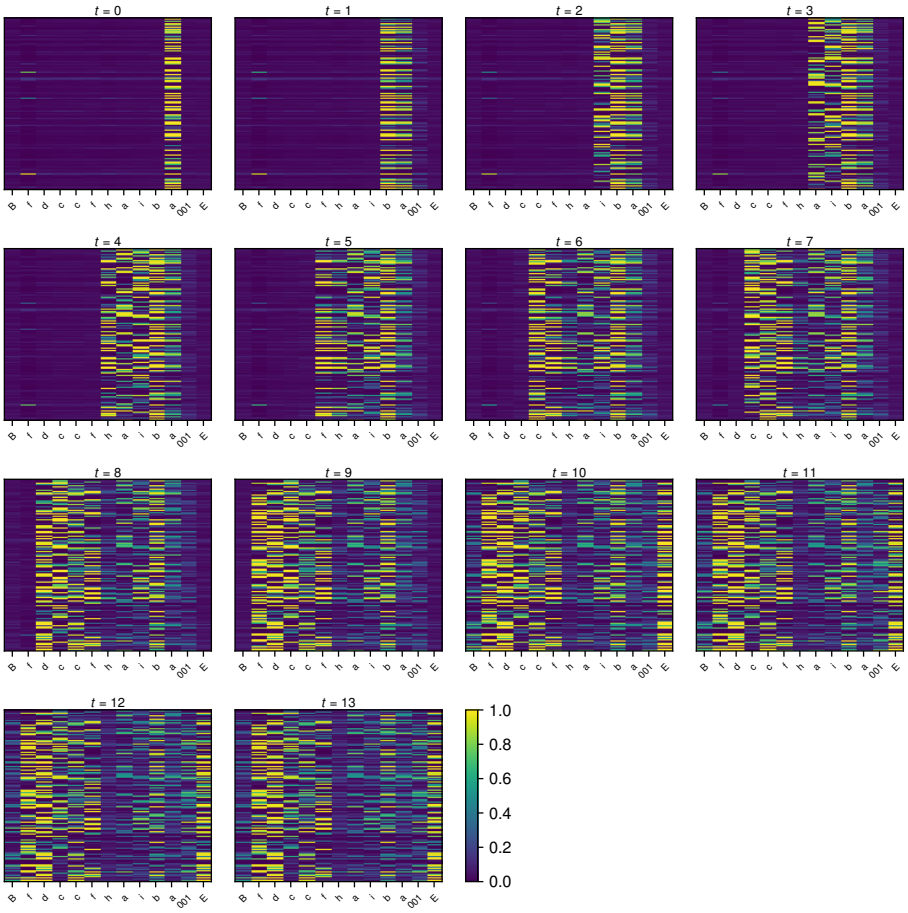

Figure 7: Gates for every computation step for a Transformer with geometric attention and copy gate (TCF). The gates remain closed until all arguments of the given operations are not ready. The attention maps for the same input can be seen in Figure 6.