

---

# Memory Transformer

---

**Mikhail S. Burtsev**

Neural Networks and Deep Learning Lab  
Moscow Institute of Physics and Technology  
Dolgoprudny, Russia  
`burtsev.ms@mpt.ru`

**Grigory V. Sapunov**

Intento, Inc.  
Berkeley, CA 94704  
`gs@inten.to`

## Abstract

Transformer-based models have achieved state-of-the-art results in many natural language processing (NLP) tasks. The self-attention architecture allows us to combine information from all elements of a sequence into context-aware representations. However, all-to-all attention severely hurts the scaling of the model to large sequences. Another limitation is that information about the context is stored in the same element-wise representations. This makes the processing of properties related to the sequence as a whole more difficult. Adding trainable memory to selectively store local as well as global representations of a sequence is a promising direction to improve the Transformer model. Memory-augmented neural networks (MANNs) extend traditional neural architectures with general-purpose memory for representations. MANNs have demonstrated the capability to learn simple algorithms like Copy or Reverse and can be successfully trained via backpropagation on diverse tasks from question answering to language modeling outperforming RNNs and LSTMs of comparable complexity. In this work, we propose and study two extensions of the Transformer baseline (1) by adding memory tokens to store non-local representations, and (2) creating memory bottleneck for the global information. We evaluate these memory augmented Transformers on machine translation task and demonstrate that memory size positively correlates with the model performance. Attention patterns over the memory suggest that it improves the model’s ability to process a global context. We expect that the application of Memory Transformer architectures to the tasks of language modeling, reading comprehension, and text summarization, as well as other NLP tasks that require the processing of long contexts will contribute to solving challenging problems of natural language understanding and generation.

## 1 Introduction

Transformers [22] are extremely successful in a wide range of natural language processing and other tasks. Due to the self-attention mechanism transformer layer can be trained to update a vector representation of every element with information aggregated over the whole sequence. As a result, rich contextual representation for every token is generated at the end of encoding. However, a combination of local and global information in the same vector has its limitations. Distributed storage of global features results in "blurring" and makes it harder to access them. Another well-known deficiency of Transformers is poor scaling of attention span that hurts its applications to long sequences.

In our work, we propose and study *MemTransformer* (Memory Transformer), a straightforward and very simple modification of the Transformer that has the potential to solve the above-mentioned problems. We augment the Transformer baseline by adding `[mem]` tokens at the beginning of the input sequence and train the model to see if it is able to use them as universal memory storage. To

assess the capacity of proposed memory implementation, we additionally studied the *MemBottleneck* model that removes attention between sequence elements, thus making memory the only channel to access global information about the sequence.

Our work lies at the intersection of two research directions Memory-augmented neural networks (MANNs) and Transformers.

The history of memory augmentation in neural networks is pretty long. Classic *Long-Short Term Memory* (LSTM) [12] can be seen as a simple yet powerful form of fine-grained memory augmentation with a single memory value per LSTM cell and memory control logic implemented by internal learnable gates. Thus, in LSTMs, computations are heavily intertwined with memory. In contrast to that, memory-augmented neural networks incorporate external-memory, which decouples memory capacity from the number of model parameters. *Neural Turing Machines* (NTMs) [6] and *Memory Networks* [23] are among the best-known MANNs that provide powerful random access operations over external memory. Memory Networks [23, 21] are trained to iteratively reason by combining sequence representation with embeddings in long-term memory with the help of attention. NTMs, and their successors *Differentiable Neural Computer* (DNC) [7] and *Sparse DNC* [19] are recurrent neural networks equipped with a content-addressable memory, similar to Memory Networks, but with the additional capability to write to memory over time. The memory is accessed by a controller network, typically an LSTM. The full model is differentiable and can be trained via back-propagation through time (BPTT). There is also a line of work to equip neural networks (typically, LSTMs) with data structures like stacks, lists, or queues [13, 8]. MANN architectures with a more advanced addressing mechanisms such as address-content separation and multi-step addressing were proposed in [10, 9, 18].

Family of Transformer models have been recently applied to many deep learning tasks and proved to be extremely successful for the language modeling tasks. The core element of Transformers is self-attention that allows updating representation for every element with information aggregated over the whole sequence. Self-attention scales as  $O(N^2)$  with a sequence length, and as a result, it is severely limited in application to long sequences.

Several recent approaches try to solve this problem by adding some kinds of memory elements to their architecture. *Transformer-XL* [5] adds segment-level recurrence with state reuse, which can be seen as a sort of memory. During training, the hidden state sequence computed for the previous segment is fixed and cached to be reused as an extended context when the model processes the next segment. *Compressive Transformer* [20] extends the ideas of Transformer-XL by incorporating the second level of the memory into the architecture. Memory on the second level stores information from the short-term memory of the first level in compressed form. *Memory Layers* [17] replace a feed-forward layer with a product key memory layer, that can increase model capacity for a negligible computational cost.

Among the most recent transformers with global representations are *Star-Transformer* [11], *Longformer* [2], and *Extended Transformer Construction* (ETC) [1]. All these architectures reduce full self-attention to some local or patterned attention and combine it with a sparse global attention bottleneck. For example, Longformer uses selected tokens such as [CLS] or tokens for question marks to accumulate and redistribute global information to all other elements of the sequence. Our MemTransformer and MemBottleneck Transformer models can be seen as more general limit cases for this class of models. Both have unspecific [mem] tokens that can store global or copy of local information. MemTransformer has full self-attention over the memory+input sequence. In contrast, MemBottleneck has full both-way attention between the input sequence and memory but no attention between sequence tokens.

## 2 Memory Transformer

### 2.1 Background: Transformer architecture

The core of the original Transformer architecture ([22]) is a scaled dot-product attention:

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

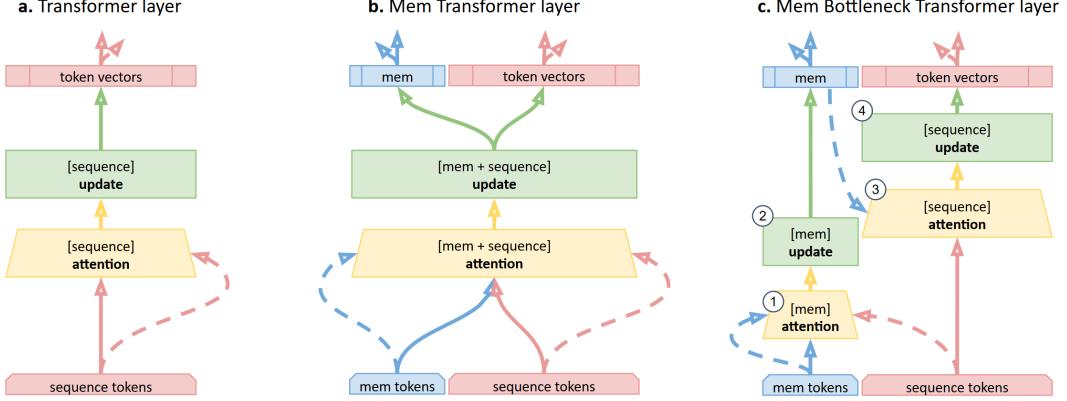


Figure 1: **Memory modifications of Transformer architecture.** (a) *Transformer layer*. For every element of a sequence (solid arrow), self-attention produces aggregate representation from all other elements (dashed arrow). Then this aggregate and the element representations are combined and updated with a fully-connected feed-forward network layer. (b) *Memory Transformer* (MemTransformer) prepends input sequence with dedicated `[mem]` tokens. This extended sequence is processed with a standard Transformer layer without any distinction between `[mem]` and other elements of the input. (c) *Memory Bottleneck Transformer* (MemBottleneck Transformer) uses `[mem]` tokens but separates memory and input attention streams. At the first step, representations of `[mem]` tokens are updated (2) with the attention span (1) covering both memory and input segments of the sequence. Then representations of input elements are updated (4) with memory attention (3) only. Thus information flow is distributed to representations of elements only through the memory.

here  $Q$ ,  $K$ , and  $V$  are queries, keys, and values which are the same in the case of encoder with self-attention, and  $d_k$  is a scaling factor.

The Transformer uses multi-head attention with  $h$  heads:

$$\begin{aligned} \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V), \\ \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O, \end{aligned} \quad (2)$$

here  $W_i^Q$ ,  $W_i^K$ ,  $W_i^V$ , and  $W^O$  are matrices of learnable transformations of the input vectors.

Then the output of the multi-head attention sublayer with a residual connection became normalized. Then there is a normalized feed-forward sublayer with a residual connection:

$$\begin{aligned} A &= \text{LayerNorm}(X + \text{MultiHead}(X, X, X)) \\ H &= \text{LayerNorm}(A + \text{FeedForward}(A)). \end{aligned} \quad (3)$$

The process of calculating self-attention can be seen as a two-step processing flow (see fig. 1a):

1. **Self-attention.** Calculate attention between all elements of the sequence.
2. **Update.** For every element of the sequence aggregate weighted representations of all other elements and perform further element-wise transformations.

## 2.2 Simple MemTransformer

The first proposed model is a simple extension of a baseline Transformer we call *MemTransformer*. It minimally changes the original model architecture.

The idea is to add  $m$  special `[mem]`ory tokens to the standard input (see fig. 1b) then process them in a standard way. So, the input vectors  $X$  became the concatenation of the memory token vectors  $X^{mem}$  and the original input token vectors  $X^{seq}$ :

$$X = [X^{mem}; X^{seq}] \in \mathbb{R}^{(n+m) \times d}, X^{mem} \in \mathbb{R}^{m \times d}, X^{seq} \in \mathbb{R}^{n \times d}.$$

This modification can be applied independently to encoder and/or decoder. The rest of the Transformer stays the same with the multi-head attention layer processing the extended input.

### 2.3 MemBottleneck Transformer

In the MemTransformer input and [mem] tokens are updated inside the same traditional self-attend and update processing flow. In this case, representations of the input sequence elements potentially might be updated “as usual” without attending to the content of the memory. Here, global information can propagate in a “peer to peer” manner. To block this distributed information flow and separate storage and processing of global and local representations, we add a memory bottleneck step to the traditional Transformer layer. The resulting *MemBottleneck* Transformer has two-staged processing flow (see fig. 1c).

**1. Memory update.** First, calculate attention between every memory token and full sequence of memory  $X^{mem}$  and input  $X^{seq}$  (see Step 1 on the fig. 1c):

$$A^{mem} = \text{LayerNorm}(X^{mem} + \text{MultiHead}(X^{mem}, X^{mem+seq}, X^{mem+seq})) \quad (4)$$

here  $A^{mem}$  is an output of *AttentionSublayer* and  $X^{mem+seq} = [X^{mem}; X^{seq}]$ . Then update memory token representations with information aggregated from the sequence and the memory itself with a residual connection, and perform further element-wise transformations (see Step 2 on the fig. 1c):

$$H^{mem} = \text{LayerNorm}(A^{mem} + \text{FeedForward}(A^{mem})). \quad (5)$$

**2. Sequence update.** Calculate attention between sequence and memory (Step 3 on the fig. 1c):

$$A^{seq} = \text{LayerNorm}(X^{seq} + \text{MultiHead}(X^{seq}, X^{mem}, X^{mem})). \quad (6)$$

Then update sequence token representations with information aggregated from memory only and perform further element-wise transformations (Step 4 on the fig. 1c):

$$H^{seq} = \text{LayerNorm}(A^{seq} + \text{FeedForward}(A^{seq})). \quad (7)$$

In other words, the memory “attends” to itself and a sequence, and the sequence “attends” only to the memory. This should force the model to accumulate and re-distribute global information through memory. Computations for MemTransformer scales linearly with the size of the sequence  $O(N)$  if the size of the memory is constant when for the traditional transformer it scales as  $O(N^2)$ .

## 3 Results and discussion

We use a standard machine translation benchmark WMT-14 DE-EN [3] in our experiments. One epoch of training covered a full train set of 4.5M pairs.

Two model sizes were studied, *small* with  $N = 4$  and *base* with  $N = 6$  layers in the encoder. The decoder has the same number of layers as the encoder. The small setup had parameters  $d_{model} = 128$ ,  $d_{ff} = 512$ ,  $h = 8$ ,  $P_{drop} = 0.1$ ,  $warmup_{steps} = 4000$ , and the base setup had  $d_{model} = 512$ ,  $d_{ff} = 2048$ ,  $h = 8$ ,  $P_{drop} = 0.1$ ,  $warmup_{steps} = 32000$ . For all experiments batch size was 64. Dataset was tokenized with TFDS subword text encoder<sup>1</sup> and dictionary of 32K per language. Training times on a single NVIDIA GTX 1080 Ti GPU were about 3.5 hours per epoch for small and 14.5 hours per epoch for base models.

Our main focus was to explore variations of memory augmented Transformer, so the majority of experiments were done on small models with just 4 layers in the encoder and 4 layers in the decoder to reduce the amount of necessary computation. Every modification was run 3 times to evaluate the reproducibility of training. All values reported in the paper are averaged over these 3 runs if the opposite is not stated.

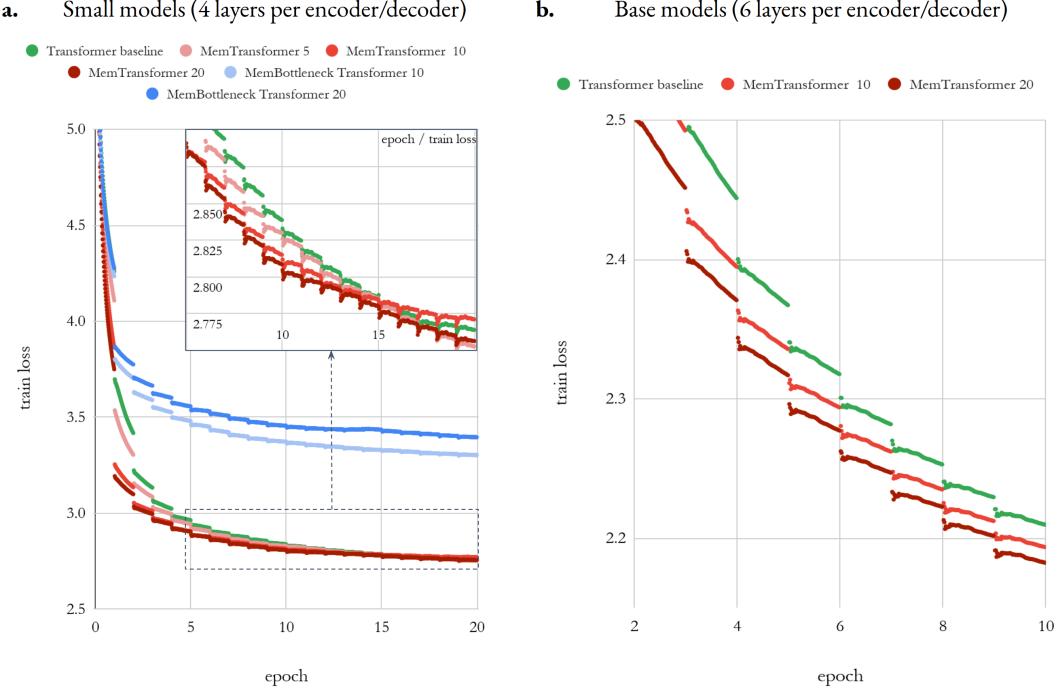
### 3.1 Performance metrics

The main hypothesis of our study says that adding memory to multilayered encoder-decoder architectures should result in better performance for sequence processing tasks such as machine translation. Indeed, learning curves presented in Figure 2 show that simple MemTransformer with 5, 10, or 20

---

<sup>1</sup>[https://www.tensorflow.org/datasets/api\\_docs/python/tfds/features/text/SubwordTextEncoder](https://www.tensorflow.org/datasets/api_docs/python/tfds/features/text/SubwordTextEncoder)

[mem] tokens has faster convergence compared to the Transformer baseline of the same size. This supports our intuition that self-attention could be trained to utilize representations of extra memory elements that are not related to the input sequence to improve the quality of encoding.



**Figure 2: Training curves for MemTransformer and MemBottleneck Transformer compared to the Transformer baseline.** (a) Training loss for 4-layer MemTransformer with 5, 10, or 20 [mem] tokens in encoder converges faster than the baseline model over the first 10 epochs. MemBottleneck models stalls with significantly higher loss relative to baseline. In the period of 10-20 epochs learning curve for some runs becomes unstable. (b) MemTransformer with 6 encoder layers outperforms baseline when using both 10 and 20 [mem] tokens.

The MemTransformer results suggest that if memory extends but not intervene in the Transformer sequence processing, then it is beneficial. But to what extent processing of relations between elements of the sequence can be abstracted to memory? Experiments with MemBottleneck Transformer (see fig. 2a.) shows that it is possible, but performance suffers. This can be due to the more complex architecture of the MemBottleneck that has twice more layers in the encoder part (see fig. 1c.). So, it is more difficult and longer to train compared to baseline. On the other hand, degraded performance can also be attributed to the insufficient throughput of the memory bottleneck. So, there might be a trade-off between the size of the bottleneck and the complexity of learning a deeper network. From the experiments, we see that MemBottleneck 10 learns faster and has lower loss compared to MemBottleneck 20, which points to the complexity of training but not the bottleneck width as a major factor limiting performance.

An increase in the memory size can have opposite effects for the performance of MemTransformer model as well. Closer inspection of the learning curves for small models (inset of fig. 2a.) shows that up to 10-12 epochs performance correlates with memory size, but at the end of the training, the average loss is the lowest for the model with 5 [mem] tokens. For the base models with 6 layers (see fig. 2b), a bigger memory size gives an advantage over the first 10 epochs of training. We see that 4-layer models with large memory learn faster but still end up with the same scores as small memory versions. We can speculate that for the model, it is easier to learn with larger memory, but it is hard to utilize extra memory tokens.

To verify the training results, we calculated BLEU scores on the validation set to estimate the quality of translation produced by the models. BLEU scores reported in Table 1 are in good agreement with the learning curves shown in Figure 2. After 20 epochs of training, small MemTransformer models have similar scores and clearly outperform the Transformer baseline. Base 6-layer MemTransformer

Table 1: **Performance of baseline and memory models on WMT-14 DE-EN translation task.**  
 Values represent an average of BLEU 4 scores for 3 runs of every model evaluated on a validation set of 2000 segments.

Small models (4 layers per encoder/decoder) (20 epochs)	Base models (6 layers per encoder/decoder) (10 epochs)
Transformer (baseline)	19.01
MemTransformer 5	<b>19.17</b>
MemTransformer 10	19.15
MemTransformer 20	19.14
MemBottleneck Transformer 10	11.20
MemBottleneck Transformer 20	10.41
MemTransformer Enc(0) Dec(10)	18.81
MemTransformer Enc(10) Dec(10)	19.07

with 10 memory tokens improves the baseline, and doubling the memory up to 20 tokens results in an even higher score of 25.58. This is a modest but solid performance compared to recent results in the literature, which mainly vary between 20 and 30 of the BLEU score (see, for example [14]).

Storing of representations in dedicated memory should be useful not only during encoding but also during decoding. MemTransformer model can be easily run with memory augmented decoder. As the simplest implementation of such memory, we just prepended target sequence with [mem] tokens. Note that we left the Transformer decoder as is, so the model was forced to learn the full target sequence prediction, starting with [mem] tokens first, and then continuing with the tokens for translation. Forward masking was applied to the full input sequence. Indeed, this modification increased difficulty of the learning task, and MemTransformer Enc(0) Dec(10) with 10 [mem] tokens in decoder performed a bit worse than the baseline (see the Table 1). But adding memory to decoder allowed MemTransformer Enc(10) Dec(10) model slightly improve the baseline scores.

Memory models have better scores after training, but do they require memory for inference? If the performance of trained MemTransformer will stay the same for the inference without [mem] tokens, then memory was only needed to improve training and not used for the processing of an input sequence. Results of memory lesion experiments presented in Figure 3 demonstrate that removing [mem] tokens from MemTransformer input leads to a dramatic drop in BLUE score, from 25.07 to 11.75 for the MemTransformer 10 and from 25.58 to 3.87 for MemTransformer 20 (both models have 6-layers in the encoder). This is an indicator that the presence of [mem] tokens is critical for MemTransformer during inference.

Another important question is related to the universality of the learned memory controller. Is it able to utilize memory of arbitrary size, or can work only with the memory capacity it was trained? Memory lesions data (see fig. 3) suggest that MemTransformer learns a solution that is partially robust to the variations in the size of the memory. BLEU score of MemTransformer 10 with 5 [mem] tokens shows it is still able to translate with acceptable quality. On the other hand, if we add 20 more [mem] tokens to the same model, it will have scores that are lower even compared to the case when the model is evaluated without memory at all. Interestingly, the model trained with a larger memory size of 20 has weaker generalization abilities. It is evident from the more steep decline of performance with the deviation of memory size from the one used during training.

### 3.2 Attention patterns in memory

To gain a better understanding of the inner workings of memory augmented transformers and following previous studies [16, 4], we visually explored attention patterns. Kovaleva et al. [16] introduced five categories of self-attention and suggested that only "heterogeneous" patterns that spread attention across all input tokens might extract non-trivial information about the linguistic structure. Numerous observations of attention maps across baseline Transformer and MemTransformer models allow us to conclude that the overall structure and distribution of pattern types in the sequence to sequence part of the attention mechanism are similar. Thus, for further analysis, we skip sequence to sequence attention and focus on memory to sequence, memory to memory, and sequence to memory attention patterns. All attention maps for selected models are presented in the Appendix.

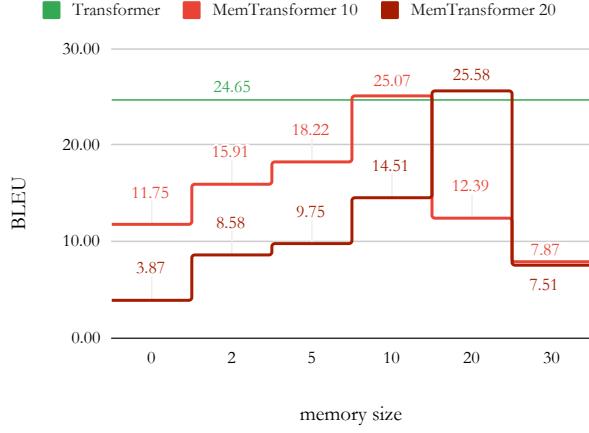


Figure 3: **Memory lesions.** Performance of the models trained with memory gradually degrades if the memory size is changed during inference. Base 6-layer MemTransformers outperforms baseline (green line) for the same memory capacity as used for training but if memory size is decreased or increased then BLEU score significantly deteriorates.

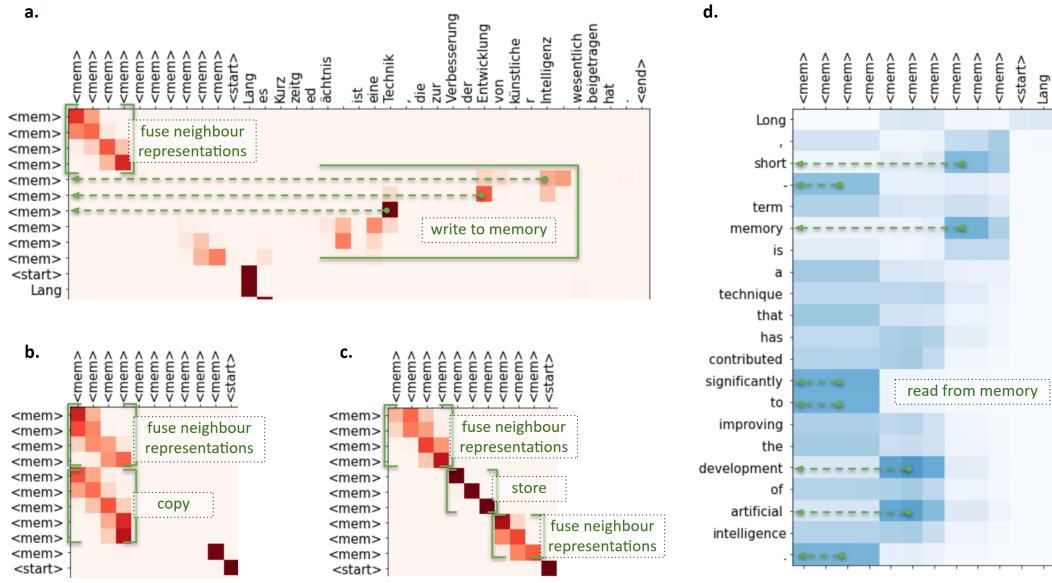


Figure 4: **Operations with memory learned by MemTransformer 10.** (a) The pattern of self-attention in the 3<sup>rd</sup> encoder layer. Here, [mem] tokens in the central segment of memory (on the left) attend to the vector representations of tokens Technik, Entwicklung, Intelligenz (and some others). This attention values are consistent with the *writing* of selected token vectors to the [mem] tokens. Activity in the left top corner that involves first four tokens might indicate *fusion* of neighbour vectors by pairwise summation of [mem] tokens. (b) In the next 4<sup>th</sup> layer of the same encoder similar fusion operation with the same [mem]'s is repeated. A parallel diagonal activity just below the fusion pattern can be attributed to *copy* operation. (c) Another attention head in the same encoder layer demonstrates combination of fusion and *store* operations. Sharp self-attention of three tokens in the middle results in adding vectors to themselves. (d) Attention pattern in decoder layer 4 over the output of 6<sup>th</sup> encoder layer suggest that vectors of [mem] tokens are selectively *read* and added to the output token representations during decoding.

**Memory to sequence attention** makes it possible to selectively update vectors stored in `[mem]` token positions with representations of input sequence elements. Such an update is a form of soft *write to memory* operation. Indeed, we found many patterns consistent with writing from sequence to memory in all MemTransformer models. One of them is shown in Figure 4a. Write to memory type of attention is more frequent in the first layers and almost absent in the deeper part of the encoder.

**Memory to memory attention** allows recombining vectors of `[mem]` tokens. We found a few common patterns related to in-memory processing. The most common arrangement of activity is diagonal. The diagonal can be blurred (or "soft"), making local *fusion* of the neighboring memory representations possible. Examples of this operation can be seen in the left top corner of the figures 4a., 4b. and 4c. If diagonal attention is sharp (see 4c. in the middle), then corresponding memory vectors are added to themselves, so their content is amplified and became more error-tolerant. This can be seen as a *store* operation. Another possible operation is a block *copy* (examples can be found in the Appendix). It is usually manifested as a vertical block of attention. In this case, a number of consequent `[mem]` vectors are updated with the same values aggregated from some another sequence of `[mem]` vectors. A copy operation can also be performed in a `[mem]` to `[mem]` manner with preserving a source order as in figure 4b. or with reversing the order (see Appendix for examples).

**Sequence to memory attention** implements *read from memory* operation and can be found in the first layers of the encoder, but it is more pronounced in the middle layers of the decoder. A typical example of the memory "reading" is presented in Figure 4d. Note, that during decoding token representation is updated by reading from a block of subsequent `[mem]` tokens.

**The overall pipeline of memory processing** is similar for the different runs and sizes of MemTransformer. It consists of writing some information from the input sequence to the memory in the first layers of the encoder, then followed by memory processing in the intermediate layers and amplification in the output layers of the encoder. During decoding, information is read from memory. Here, the highest "reading" activity is commonly observed in the intermediate decoder layers. Interestingly, memory-related attention patterns usually have a block structure. For example, patterns form the particular MemTransformer 10 presented in Figure 4 suggest that the model had learned to split memory into three blocks. Commonly, the same memory operation is applied to all `[mem]`s of the same block by one particular head. During memory processing, the model can operate in a blockwise manner, as in Figure 4b, where the "block(1-3)" is copying to the "block(4-6)". We speculate that the block structure of memory processing might reduce error rate because the memory representation is "averaged" over the `[mem]`s of the block during reading (see fig. 4d). Experiments with MemBottleneck architecture show that the model might be able to learn how to copy representations of the input sequence into the memory of the fixed size and use only this memory during decoding.

## 4 Conclusions

We proposed and studied two memory augmented architectures *MemTransformer* and *MemBottleneck Transformer*. Qualitative analysis of attention patterns produced by the transformer heads trained to solve machine translation task suggests that both models successfully discovered basic operations for memory control. Attention maps show evidence for the presence of memory read/write as well as some in-memory processing operations such as copying and summation.

A comparison of machine translation quality shows that adding general-purpose memory in MemTransformer improves performance over the baseline. Moreover, the speed of training and final quality positively correlates with the memory size. On the other hand, MemBottleneck Transformer, with all self-attention restricted to the memory only, has significantly lower scores after training.

Memory lesion tests demonstrate that the performance of the pre-trained MemTransformer model critically depends on the presence of memory. Still, the memory controller learned by the model degrades only gradually when memory size is changed during inference. This indicates that the controller has some robustness and ability for generalization.

Our memory augmentation approach is universal and can extend almost any "encoder-decoder with attention" framework. We see as a particularly promising combination of MemTransformer with the recurrency mechanisms [5, 20] to process long sequences. It can be also applied to the tasks that depend on the multi-hop reasoning or planning. In this cases memory should help to store and process representations for the intermediate stages of the solution.

## References

- [1] J. Ainslie, S. Ontanon, C. Alberti, P. Pham, A. Ravula, and S. Sanghai. Etc: Encoding long and structured data in transformers, 2020.
- [2] I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer, 2020.
- [3] O. Bojar, C. Buck, C. Federmann, B. Haddow, P. Koehn, J. Leveling, C. Monz, P. Pecina, M. Post, H. Saint-Amand, R. Sorice, L. Specia, and A. s. Tamchyna. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics.
- [4] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning. What Does BERT Look at? An Analysis of BERT’s Attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286, Florence, Italy, 2019. Association for Computational Linguistics.
- [5] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context, 2019.
- [6] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines, 2014.
- [7] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A. P. Badia, K. M. Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, and D. Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, Oct. 2016.
- [8] E. Grefenstette, K. M. Hermann, M. Suleyman, and P. Blunsom. Learning to transduce with unbounded memory, 2015.
- [9] C. Gulcehre, S. Chandar, and Y. Bengio. Memory augmented neural networks with wormhole connections. *arXiv preprint arXiv:1701.08718*, 2017.
- [10] C. Gulcehre, S. Chandar, K. Cho, and Y. Bengio. Dynamic neural turing machine with soft and hard addressing schemes. *arXiv preprint arXiv:1607.00036*, 2016.
- [11] Q. Guo, X. Qiu, P. Liu, Y. Shao, X. Xue, and Z. Zhang. Star-transformer, 2019.
- [12] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.
- [13] A. Joulin and T. Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets, 2015.
- [14] J. Kasai, J. Cross, M. Ghazvininejad, and J. Gu. Parallel machine translation with disentangled context transformer. *arXiv preprint arXiv:2001.05136*, 2020.
- [15] G. Kobayashi, T. Kurabayashi, S. Yokoi, and K. Inui. Attention Module is Not Only a Weight: Analyzing Transformers with Vector Norms. *arXiv:2004.10102 [cs]*, 2020.
- [16] O. Kovaleva, A. Romanov, A. Rogers, and A. Rumshisky. Revealing the Dark Secrets of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4356–4365, Hong Kong, China, 2019. Association for Computational Linguistics.
- [17] G. Lample, A. Sablayrolles, M. Ranzato, L. Denoyer, and H. Jégou. Large memory layers with product keys, 2019.
- [18] Y. Meng and A. Rumshisky. Context-aware neural model for temporal information extraction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 527–536, 2018.
- [19] J. W. Rae, J. J. Hunt, T. Harley, I. Danihelka, A. Senior, G. Wayne, A. Graves, and T. P. Lillicrap. Scaling memory-augmented neural networks with sparse reads and writes, 2016.
- [20] J. W. Rae, A. Potapenko, S. M. Jayakumar, and T. P. Lillicrap. Compressive transformers for long-range sequence modelling, 2019.
- [21] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus. End-to-end memory networks, 2015.

- [22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is All you Need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
- [23] J. Weston, S. Chopra, and A. Bordes. Memory networks, 2014.

## A Attention maps for memory augmented transformers

In this section we present attention maps for two representative cases of MemTransformer and MemBottleneck transformer models. For both models we use the same input sequence.

**Input sequence:** *Langes Kurzzeitgedächtnis ist eine Technik, die zur Verbesserung der Entwicklung von künstlicher Intelligenz wesentlich beigetragen hat.*<sup>2</sup>

**Predicted translation MemTransformer 10:** *Long, short-term memory is a technique that has contributed significantly to improving the development of artificial intelligence.*

**Predicted translation MemBottleneck 20:** *The short time memory is a technique that has helped to improve the development of artificial intelligence in a lot of sense.*

**Reference:** *Long-term short-term memory is a technique that has contributed significantly to improving the development of artificial intelligence.*<sup>3</sup>

A short guide to help with interpretation of attention maps is shown on the Figure 5.

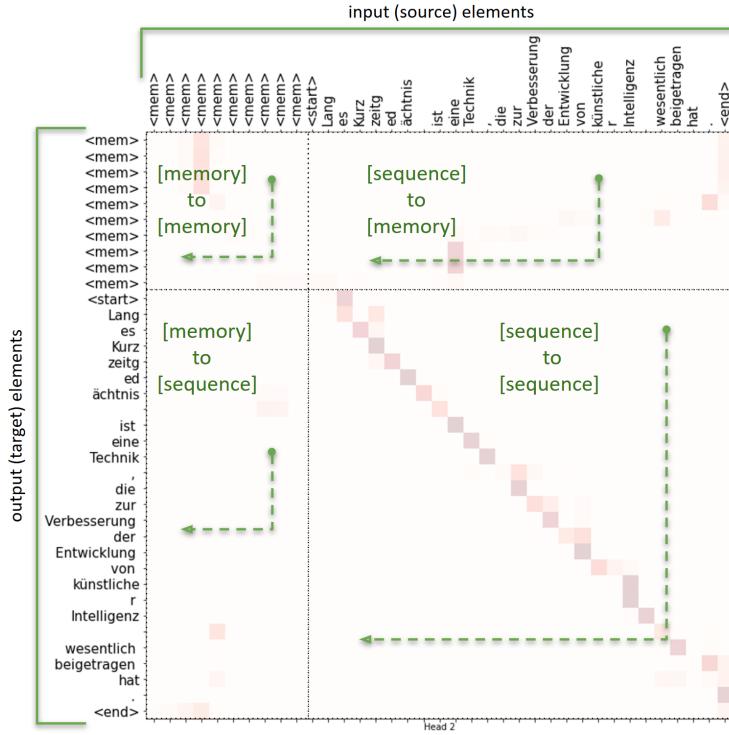


Figure 5: **How to read Memory Transformer attention map.** Attention values indicate how elements of input sequence (on the top) contribute to the update of representation for specific output element (on the left). Attention map for memory augmented transformer can be split into four blocks: (1) *update* - [sequence] to [sequence]; (2) *write* - [sequence] to [memory]; (3) *read* - [memory] to [sequence]; (4) *process* - [memory] to [memory].

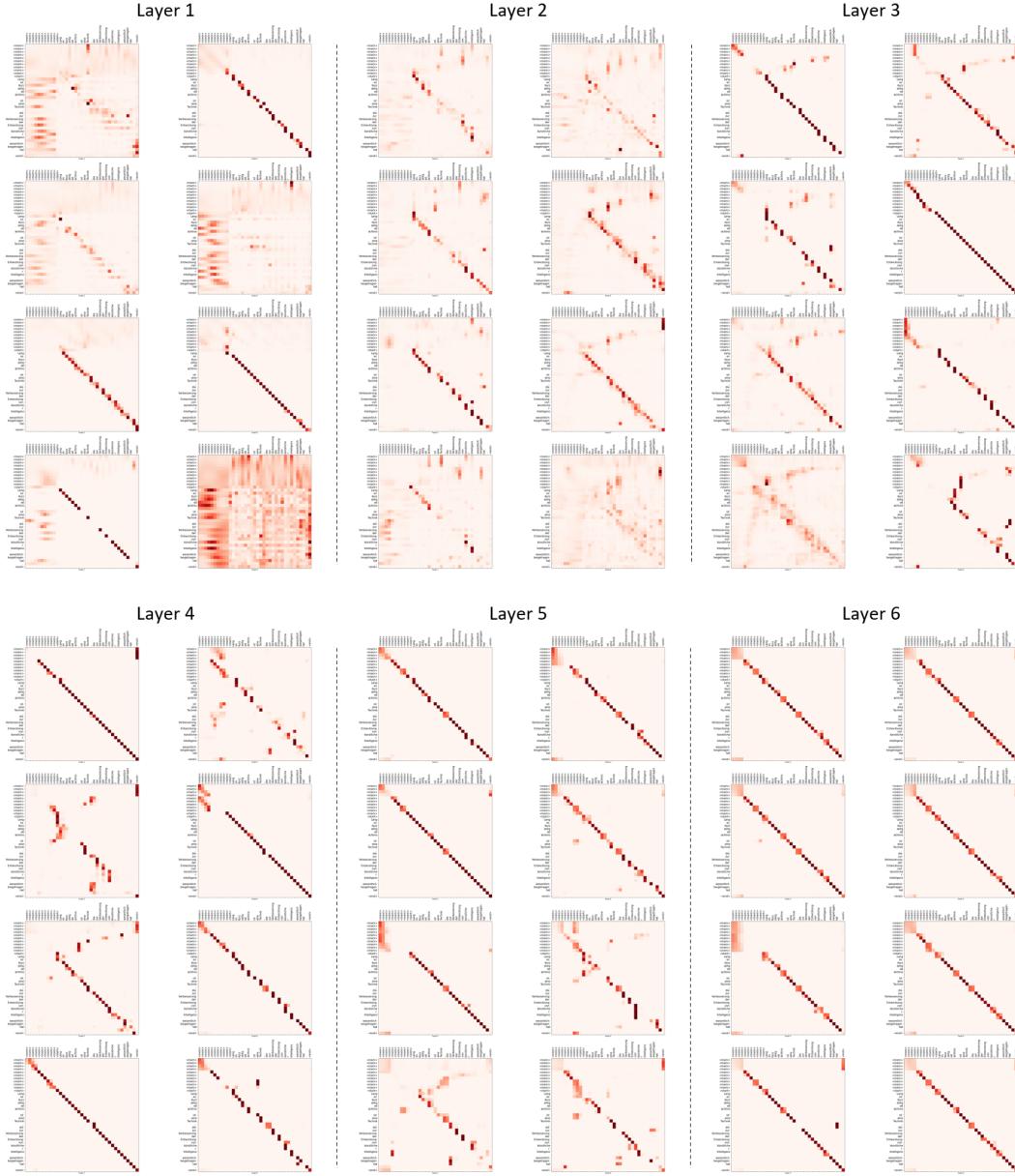
### A.1 MemTransformer attention maps

Visualisation of attention maps for MemTransformer 10 (see Section 2.2) with a memory size of 10 is presented on the Figure 6 for 6 layers of encoder and on the Figure 7 for 6 layers of decoder. Every transformer layer has 8 attention heads. The model was trained for 10 epochs on WMT-14 DE-EN [3] dataset.

<sup>2</sup>[https://de.wikipedia.org/wiki/Long\\_short-term\\_memory](https://de.wikipedia.org/wiki/Long_short-term_memory)

<sup>3</sup><https://translate.google.com>

**MemTransformer 10 encoder**  
(self-attention maps)



**Figure 6: MemTransformer 10 encoder attention maps.** As the model encodes an input sequence the change of attention patterns related to memory can be interpreted as a *read-process-store* pipeline. Heads in layers 1 to 3 have many *read to memory* patterns. Patterns consistent with *in memory processing* are more frequent in layers 3-6. The last layer is dominated by diagonal attention that can be seen as an amplification of calculated representations.

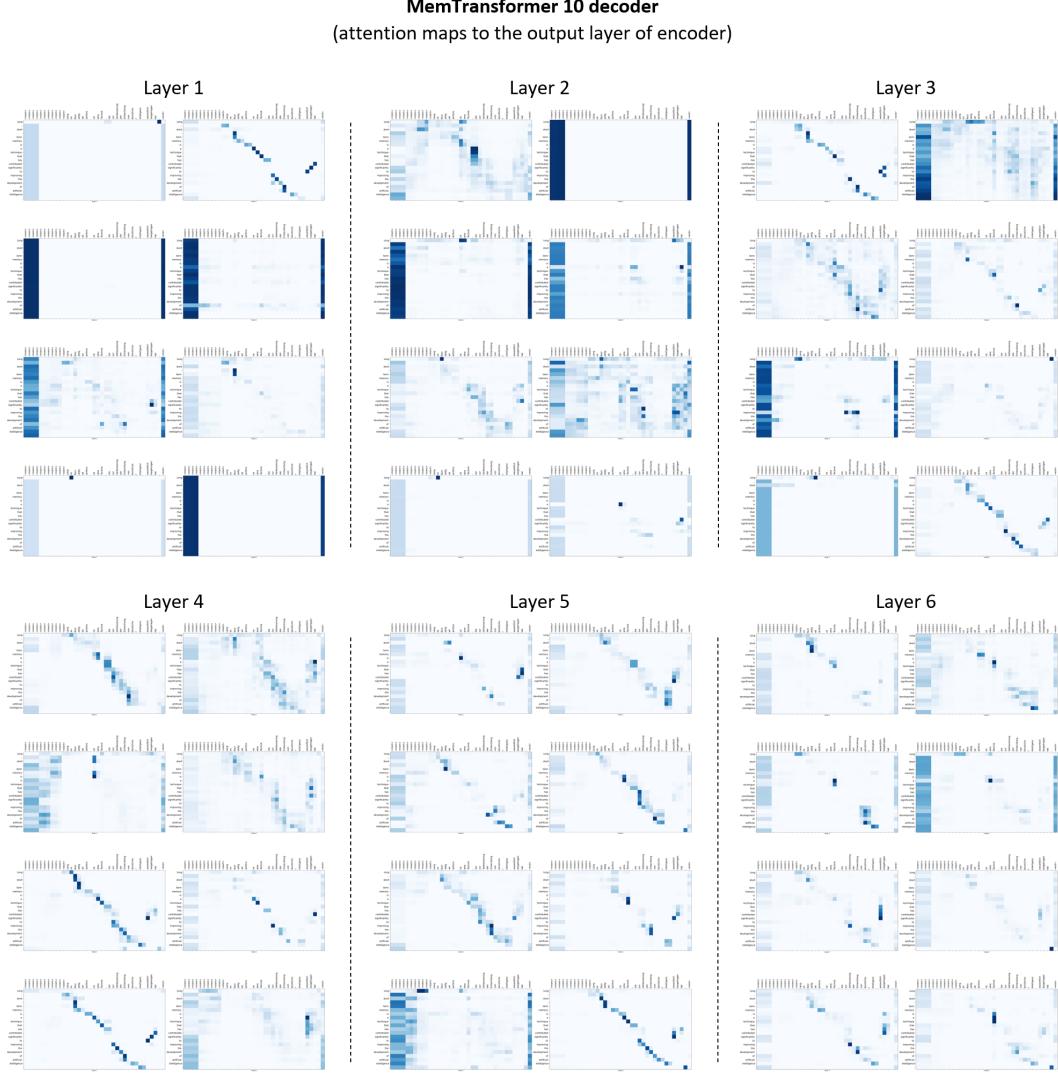
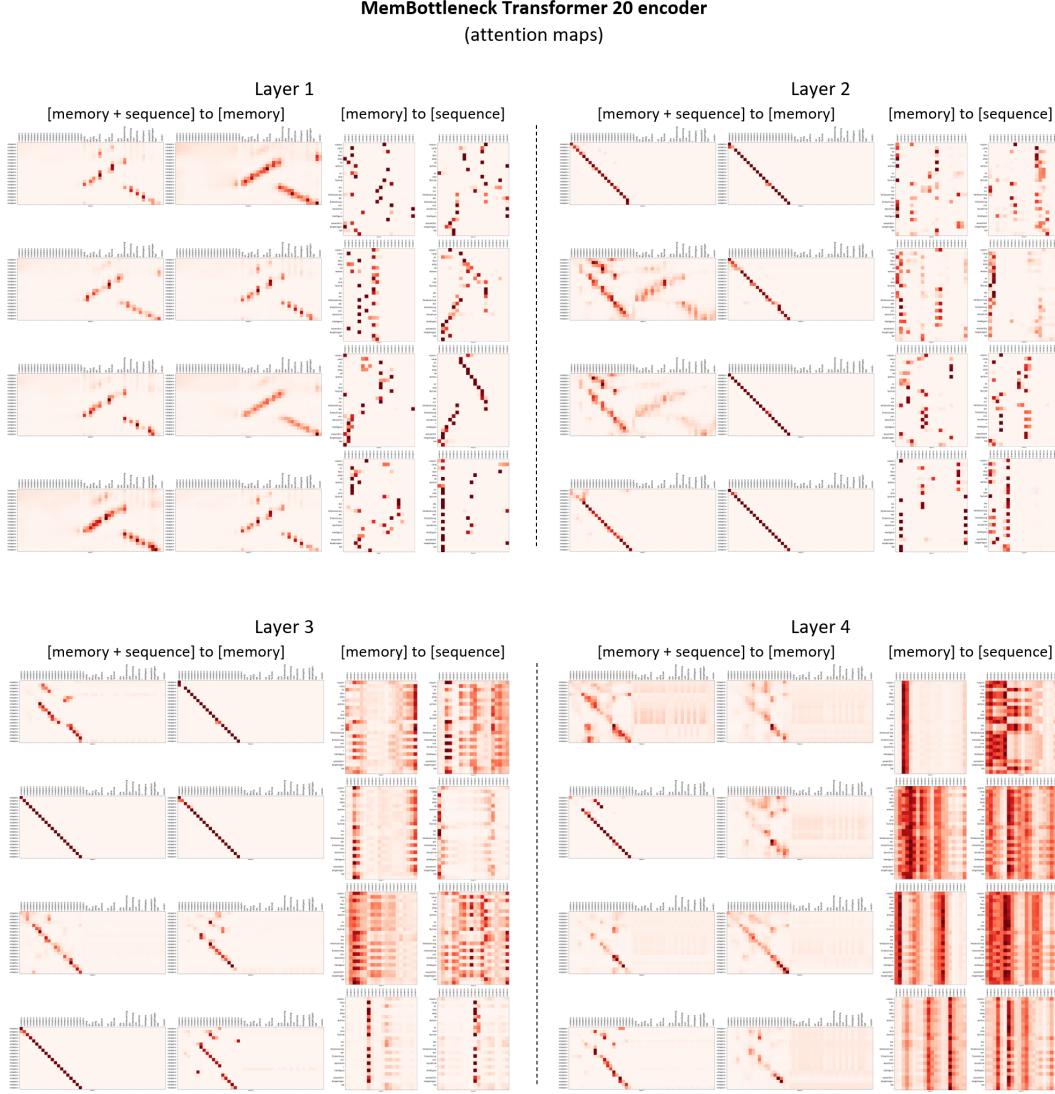


Figure 7: **MemTransformer 10 decoder attention maps.** Every layer of the decoder has heads with signs of memory reading activity. Reading patterns suggest that the representations in memory are locally grouped in 3 blocks.

## A.2 MemBottleneck Transformer attention maps

Attention patterns generated by MemBottleneck Transformer architecture (see Section 2.3) strongly suggest that the model learned to copy a given sequence into a memory, process it and use only this representations of input for decoding. The main idea of MemBottleneck is a restriction of global information exchange to memory. Therefore, an update for representations of the input sequence elements can access representations of other elements only by writing into and then reading them from memory. To do that, MemBottleneck uses two different transformer sub-layers each with its' own set of attention heads (see fig. 1c).

Encoder attention maps (see fig. 8) suggest that, as expected, representations for the input elements are copied into memory in layers 1 and 2. Surprisingly, after that they are not properly updated anymore and the decoder mostly attends to the content of memory (see fig. 9). This impressive outcome shows that transformer can be trained to read and process all the information about the input sequence in memory only.



**Figure 8: MemBottleneck 20 encoder attention maps.** In the 1st layer, all attention heads of memory sub-layer ([memory+sequence] to [memory]) read from the input sequence. Only 2 heads of memory sub-layer in the layer 2 reads from the input, but all others are diagonal to amplify content of the memory. No more input reading is present in layers 3 and 4. Notably, all heads of the 1st layer memory attention have patterns that split into three blocks. The top block has sparse attention over the whole sequence without preserving the order. The middle block reads the first half of the sequence in the reverse order, and the bottom block reads the rest in the proper order. This suggests encoding of global information in the top block and local information in the middle and bottom blocks. Layer 3 of memory sub-layer has sharp amplifying diagonals, and something like shifting operations represented by broken diagonals. Layer 4 of memory sub-layer demonstrates mainly heterogeneous patterns which indicates in memory processing. Maps of attention which belongs to the sequence sub-layer ([memory] to [sequence]) of MemBottleneck layer degrade to vertical lines in layers 3 and 4. This is a sing that these attention heads are bypassed as follows from [15].

**MemBottleneck Transformer 20 decoder**  
 (attention maps to the output layer of encoder)

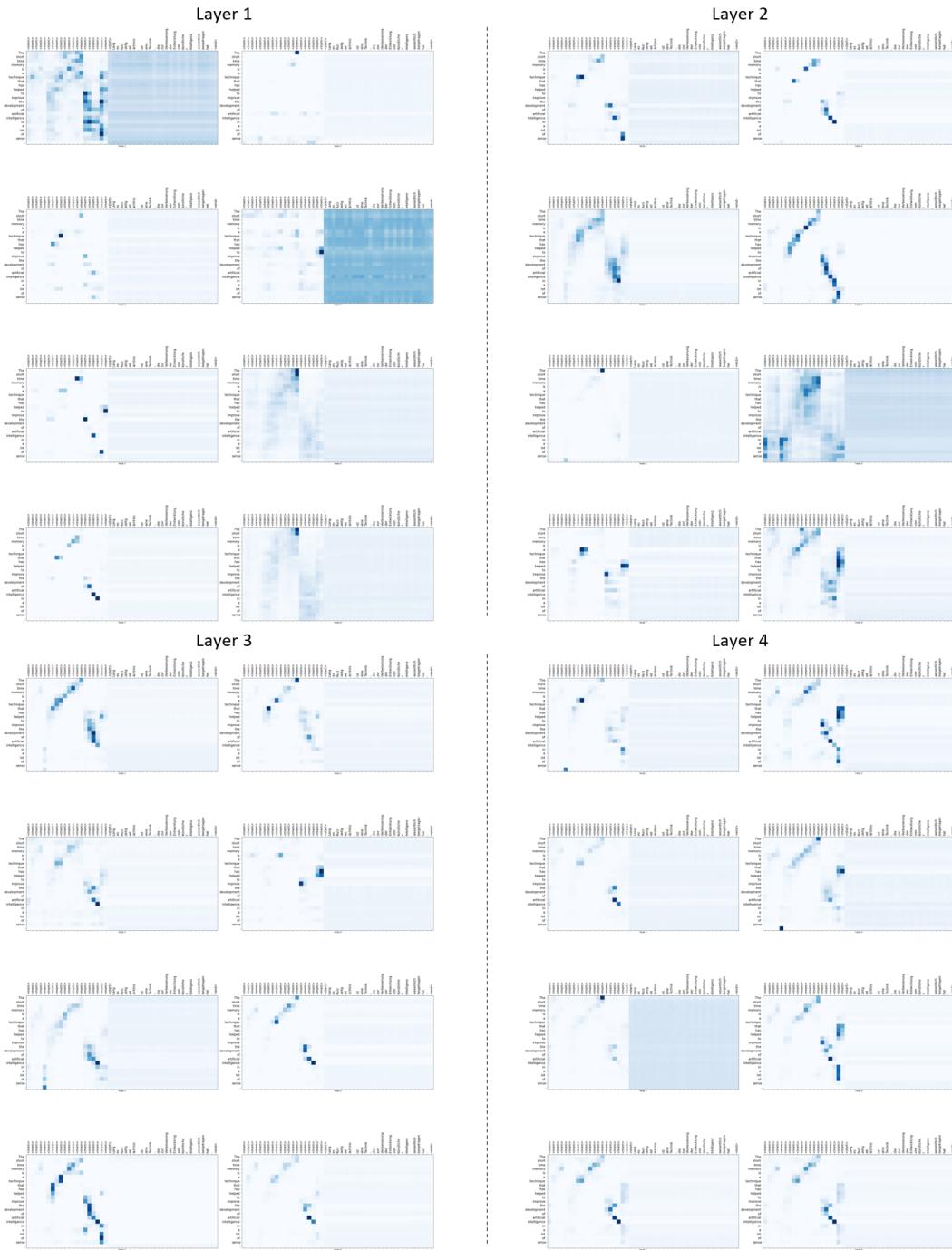


Figure 9: **MemBottleneck 20 decoder attention maps.** At the decoding phase, almost all heads attend to the content of memory but not on the representations of sequence elements.