Recurrent Memory Transformer

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Transformer-based models show their effectiveness across multiple domains and tasks. The self-attention allows to combine information from all sequence elements into context-aware representations. However, global and local information has to be stored mostly in the same element-wise representations. Moreover, the length of an input sequence is limited by quadratic computational complexity of self-attention. In this work, we propose and study a memory-augmented segment-level recurrent Transformer (Recurrent Memory Transformer). Memory allows to store and process local and global information as well as to pass information between segments of the long sequence with the help of recurrence. We implement a memory mechanism with no changes to Transformer model by adding special memory tokens to the input or output sequence. Then Transformer is trained to control both memory operations and sequence representations processing. Results of experiments show that our model performs on par with the Transformer-XL on language modeling for smaller memory sizes and outperforms it for tasks that require longer sequence processing. This makes Recurrent Memory Transformer a promising architecture for applications that require learning of long-term dependencies and general purpose in memory processing, such as algorithmic tasks and reasoning.

1 Introduction

Transformers (Vaswani et al., 2017) have been widely adopted across multiple domains and tasks (Radford et al., 2018; Dong et al., 2018; Devlin et al., 2019; Dosovitskiy et al., 2021; Ramesh et al., 2021; Jaegle et al., 2021). The key component of Transformer layer is a self-attention. Self-attention allows to update each sequence element representation with information from all other elements in the sequence. As a result, rich contextual representation for every element is generated at the end of encoding. This way, global sequence-level and local information are stored in a single representation. However, this mixing of two types of information in a single representation has limitations. Distributed storage of global features across all sequence elements results in global features "blurring" and makes it harder to access them.

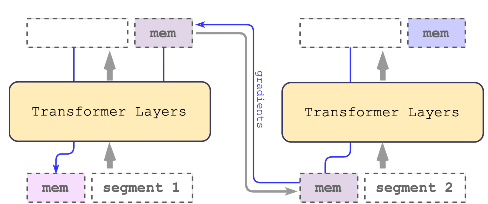


Figure 1: **Recurrent Memory Transformer.** Memory is added as tokens to the input sequence and memory output is passed to the next segment. During training gradients flow from the current segment through memory to the previous segment.

Another well-known deficiency of Transformers is poor scaling of self-attention with input sequence length that hurts its applications to long inputs (Child et al., 2019; Guo et al., 2019; Dai et al., 2019; Beltagy et al., 2020; Ainslie et al., 2020; Zaheer et al., 2020; Wang et al., 2020; Choromanski et al., 2020).

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

Our work introduces a memory-augmented segment-level recurrent Transformer named Recurrent Memory Transformer (RMT). RMT uses a memory mechanism based on special memory tokens (Burtsev et al., 2020) added to the input sequence. Memory tokens provide additional reserved capacity to the model that could be used to process information which is not directly representing any element in the input sequence. To process long sequences, we split them into segments and pass memory states from a previous to a current segment. This memory passing makes the model recurrent and removes the input sequence length limitations. RMT model can theoretically work with infinite lengths but, in practice, it is limited by memory capacity and the efficiency of memory access/update operations. Our implementation of both memory and recurrence in RMT requires no changes to the Transformer model because modifications are made only to the input and output sequences of the model.

We tested RMT on the tasks that require a global information about the whole input sequence to be solved. We use copy, reverse, and associative retrieval tasks in the setting where the input sequence is split into segments. RMT and Transformer-XL perfectly solve these tasks, but exceeding some value of sequence length, RMT starts to outperform Transformer-XL. Also, we experimentally show that the proposed Recurrent Memory Transformer requires less memory size to perform closely to Transformer-XL on language modeling tasks. RMT code and experiments are available[1].

Contributions

- In this study we augment Transformer with token based memory storage and segment-level recurrence.
- We experimentally evaluate proposed architecture as well as vanilla Transformer and Transformer-XL on memory-intensive tasks such as copy, reverse, associative retrieval and language modeling. We show that RMT outperforms Transformer-XL for sequence processing tasks and on par with Transformer-XL on language modeling but requires less memory.
- We analysed how the Transformer model learns to use memory. Specific interpretable memory read-write patterns of attention are shown.

2 Related work

In our study we add a memory to general purpose attention based neural architecture. Memory is a recurrent topic in neural networks research. It had started from the early works (McCulloch and Pitts, 1943; Stephen, 1956) and significantly progressed in 90's with introduction of *Backpropagation Through Time* learning algorithm (Werbos, 1990) and *Long-Short Term Memory* (LSTM) (Hochreiter and Schmidhuber, 1997) neural architecture. Today memory-augmented neural networks (MANNs) usually rely on some kind of recurrent external-memory which is separate from the model's parameters. *Neural Turing Machines* (NTMs) (Graves et al., 2014) and *Memory Networks* (Weston et al., 2014) are equipped with a storage for vector representations that can be accessed with an attention mechanism. Memory Networks (Weston et al., 2014; Sukhbaatar et al., 2015) were designed to enable reasoning by sequential attention over to the content of a memory. NTMs followed by *Differentiable Neural Computer* (DNC) (Graves et al., 2016) and *Sparse DNC* (Rae et al., 2016) are implemented as recurrent neural networks able to write to memory storage over time. All these models are differentiable and can be trained via backpropagation through time (BPTT). Parallel line of research extends recurrent neural networks such as LSTM with data structures like stacks, lists, or queues (Joulin and Mikolov, 2015; Grefenstette et al., 2015). MANN architectures with a more advanced addressing mechanisms such as address-content separation and multi-step addressing were proposed in (Gulcehre et al., 2016, 2017; Meng and Rumshisky, 2018). The Global Context Layer model (Meng and Rumshisky, 2018) uses the idea of address-content separation to solve the difficulty of training content-based addressing in the canonical NTM.

Recent rise of Transformer models also resulted in introduction of a number of new memory architectures. *Transformer-XL* (Dai et al., 2019) introduces a segment-level recurrence at the level of hidden representations. These representations of a sequence are computed and stored in the cache to be reused as an extended context for the next segment. *Compressive Transformer* (Rae et al.,

[1]anonymous link. Code, raw experiments results and hyperparameters are provided in supplementary materials.

2019) adds the second layer of memory to Transformer-XL. This memory compresses and stores information from the cache. *∞-former* (Martins et al., 2021) utilizes continuous-space attention and represents input sequence as a continuous signal to make long-term memory unbounded. *Memory Layers* (Lample et al., 2019) model has a product key memory layer instead of a feed-forward layer within Transformer block to increase model capacity.

In many variations of Transformer different sorts of global representations are added. Among them are *Star-Transformer* (Guo et al., 2019), *Longformer* (Beltagy et al., 2020), *GMAT* (Gupta and Berant, 2020), *Extended Transformer Construction* (ETC) (Ainslie et al., 2020) and *Big Bird* (Zaheer et al., 2020). All these architectures re-design self-attention mechanism to reduce it computational complexity with and ensure input coverage with the help of global representations. *Memory Transformer* (Burtsev et al., 2020) keeps Transformer model intact and adds memory by extending input sequence with special memory tokens. Perceiver IO (Jaegle et al., 2021) maps an entire arbitrary input to the fixed number of latent representations. Transformer layers do further processing over latent memory representations only.

Segment-level recurrence in Transformers is actively explored in a number of studies. Transformer-XL, Compressive Transformer keep previous states and re-use them in subsequent segments. Ernie-Doc (Ding et al., 2021) improves processing by using same-layer recurrence instead of attending to previous layer outputs of a precedent segment. Memformer (Wu et al., 2020) introduces a dedicated memory module to keep previous hidden states in summarized representations. Memformer uses two special layers added to the Transformer model. Memory cross-attention layer reads from memory and memory slot attention layer updates it. MART (Lei et al., 2020) has a similar approach as Memformer but uses memory update rules analogous to LSTM (Hochreiter and Schmidhuber, 1997) and GRU (Cho et al., 2014). FeedBack Transformer (Fan et al., 2020) goes further with full, and not segment-level, recurrence. FeedBack Memory merges past hidden representations from all layers into a single vector and makes it accessible to the computations at any layer. The disadvantage of full recurrence is that it is less parallelizable. FeedBack Memory requires every sequence element to be processed sequentially. In segment-level recurrent models, all elements of a segment are processed by Transformer layers in parallel. Only segments are processed sequentially. Staircase Transformer (Ju et al., 2021) combines segment-level recurrence and depth recurrence. Staircase models use the output for previous segments and pass them as input for the next segment. Our Recurrent Memory Transformer is based on special memory tokens similar to Memory Transformer, segment-level recurrence as in Transformer-XL, and depth-recurrent mechanism for memory processing similar to Staircase.

3 Recurrent Memory Transformer

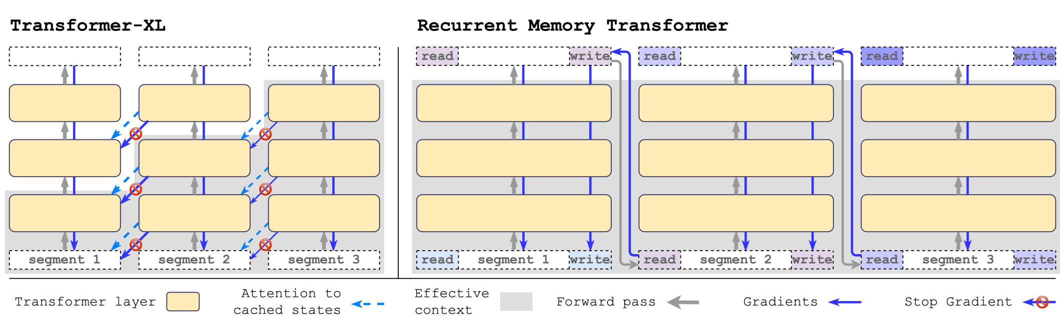


Figure 2: **Comparison of Recurrent Memory Transformer and Transformer-XL architectures.** Recurrent Memory Transformer augments Transformer with global memory tokens and passes them to allow a segment-level recurrence. Special read/write memory tokens are added to the input sequence. Multiple memory tokens can be used in each read/write block. Updated representations of write memory are passed to the next segment. During training, RMT uses BPTT to propagate gradient to previous segments through memory tokens representation. Recurrence with memory has no limitations on effective context length, whereas Transformer-XL can use only finite context with cached states. All RMT memory/recurrence operations are made on the input and output level of the Transformer model.

3.1 Background: Transformer-XL

Transformer-XL (Dai et al., 2019) extends Transformer model with state re-use cache mechanism for segment-level recurrence and relative position encoding. Input sequence is split on segments processed sequentially. Hidden states computed for the previous segment M^n are cached for each transformer layer n. The input of the layer n consists of the last m states from the cached memory and output of previous Transformer layer for the current segment τ:

$$\tilde{H}_\tau^{n-1} = [SG(M_{-m:}^{n-1}) \circ H_\tau^{n-1}], \tag{1}$$

here, SG stands for stop-gradient, $\circ$ denotes concatenation. Cached states allow to increase effective context size of Transformer model and save on compute operations.

Then, $\tilde{H}_\tau^{n-1}$ goes to Transformer layer to produce layer n outputs for segment τ:

$$H_\tau^n = \text{TransformerLayer}(Q_\tau^n, K_\tau^n, V_\tau^n),$$
$$Q_\tau^n = W_q^n H_\tau^{n-1}; K_\tau^n = W_k^n \tilde{H}_\tau^{n-1}; V_\tau^n = W_v^n \tilde{H}_\tau^{n-1}.$$

In Transformer-XL, self-attention layers are modified to use relative position encodings to improve generalization to longer attention lengths. The overall architecture is shown in the Figure 2.

3.2 Memory and recurrence

Memory augmented Transformers such as GMAT, ETC, Memory Transformer (Gupta and Berant, 2020; Ainslie et al., 2020; Burtsev et al., 2020) proposed to use special global tokens as storage for representations. Usually, memory tokens are added to the beginning of the input sequence. However, in decoder-only architectures the causal attention mask makes impossible for memory tokens at the start of the sequence to collect information from the subsequent tokens. On the other hand, if memory tokens are placed at the end of the sequence then preceding tokens unable to access their representations. To solve this problem we add a recurrence to the sequence processing. Representations of memory tokens placed at the end of the segment are used as an input memory representations at the start as well as at the end of the next segment.

In the Recurrent Memory Transformer input is augmented with special `[mem]` tokens, processed in a standard way along with the sequence of tokens. Each memory token is a real-valued vector. m memory tokens are added at the beginning of the segment tokens representations H_τ^0 and the same m tokens are added at the end:

$$\tilde{H}_\tau^0 = [H_\tau^{mem} \circ H_\tau^0 \circ H_\tau^{mem}],$$
$$\bar{H}_\tau^N = \text{Transformer}(\tilde{H}_\tau^0),$$
$$[H_\tau^{read} \circ H_\tau^N \circ H_\tau^{write}] := \bar{H}_\tau^N,$$

here N is a number of Transformer layers.

The starting group of memory tokens functions as a read memory that allows sequence tokens to attend to memory states produced at the previous segment. The ending group works as a write memory that can attend to all current segment tokens and update representation stored in the memory. As result, H_τ^{write} contains updated memory tokens for the segment τ.

Segments of the input sequence are processed sequentially. To enable recurrent connection between segments, we pass outputs of the memory tokens from the current segment to the input of the next segment:

$$H_{\tau+1}^{mem} := H_\tau^{write},$$
$$\tilde{H}_{\tau+1}^0 = [H_{\tau+1}^{mem} \circ H_{\tau+1}^0 \circ H_{\tau+1}^{mem}].$$

Both memory and recurrence in the RMT are based only on global memory tokens. It allows to keep the backbone Transformer unchanged and make RMT memory augmentation compatible with any model from the Transformer family. Memory tokens operate only on the input and output of the model. In this study we implement RMT on top of the original Transformer-XL code. Both architectures are shown in Figure 2.

Recurrence in the RMT is different compared to the Transformer-XL because the former stores only m memory vectors per segment. On the other hand, the Transformer-XL stores $m \times N$ vectors per segment. Also, in the RMT model memory representations from the previous segment are processed by Transformer layers together with the current segment tokens. This makes memory part of RMT effectively deeper in a number of applied Transformer layers $\tau \times N$. Additionally, we allow all memory tokens in the read/write block to access all other tokens in the same block. The causal attention mask is applied only to tokens of the input sequence. The RMT attention mask is shown in Figure 6 (d).

We train the RMT with Backpropagation Through Time (BPTT). During backward pass, unlike in Transformer-XL, memory gradients are not stopped between segments. The number of previous segments to backpropagate is a hyperparameter of a training procedure. We vary BPTT unroll in our experiments from 0 to 4 previous segments. Increasing this parameter is computationally expensive and requires a lot of GPU RAM. However, such techniques as gradient checkpointing could be used to alleviate this problem.

4 Experiments

We designed our experiments to evaluate the ability of Recurrent Memory Transformers to preserve long-term dependencies across multiple input segments. The first set of experiments includes copy, reverse, associative retrieval and quadratic equations tasks. The second addresses language modeling task for word-level on WikiText-103 (Merity et al., 2017) and for character-level on enwik8 (Mahoney, 2006). We compare Recurrent Memory Transformer with Transformer and Transformer-XL models.

4.1 Algorithmic Tasks

Firstly, we evaluate RMT on algorithmic tasks that require information about the whole input sequence to be solved successfully. In a recurrent setting, the model has to keep information about all previous segments to make predictions.

In the copy task, an input sequence should be replicated twice after a special start-to-generate token. In the reverse task, an input sequence should be generated in a reverse order. Input for the associative retrieval task consists of N key-value pairs. Then one key is randomly selected, and the task is to produce an appropriate value for the selected key. Another task is to solve quadratic equations. One example consists of an equation, its solution with discriminant, and an answer. The task is to generate a solution and answer, while only answer quality is evaluated.

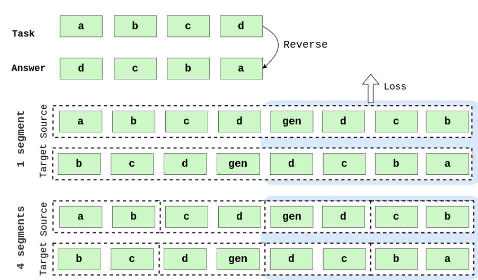


Figure 3: **Reverse task in one and four segments setting for decoder-only models.** Dotted lines show segment borders.

For all tasks, input and output sequences are split into segments and processed by models sequentially. Datasets for algorithmic tasks were randomly pre-generated, and the same data was used in all experiments.

Transformer-XL and RMT are decoder-only Transformer models. We do not compute loss over the input sequence before the start-to-generate token. The loss is computed over target sequence segments only. This procedure is illustrated in Figure 3.

4.2 Language Modeling

We use two standard benchmarks for language modeling: WikiText-103 and enwik8. WikiText-103 (Merity et al., 2017) is used for word-level language modeling and contains 103M words from English Wikipedia articles. Enwik8 (Mahoney, 2006) is used for character-level and consists of 10^8 first bytes of XML text dump of the English Wikipedia.

We compare Recurrent Memory Transformer with decoder-only Transformer and Transformer-XL as baselines. Model size and training parameters are selected to match Transformer-XL paper. For Wikitext-103 an input context length was set to 150 tokens, and for enwik8 it was set to 512 characters.

Another set of experiments inspected how RMT handles long-term dependencies and recurrence. We increased the number of segments and recurrent steps by making segments smaller (50 tokens for WikiText-103, 128 characters for enwik8). The increased number of recurrent steps makes language modeling tasks harder for RMT because information has to be stored in the same amount of memory for more time steps.

4.3 Implementation details

Our RMT implementation is based on Transformer-XL repository[2]. We also use WikiText-103, enwik8 data and processing from this repository. Language modeling experiments follow the same model and training hyperparameters as Transformer-XL. WikiText-103 experiments use 16-layer Transformers, enwik8 – 12 layer Transformers. We refer to Transformer-XL with memory size equal to zero as a Baseline.

With this experimental setup we were able to reproduce results for the Transformer-XL model close to the original paper (see Appendix A.4 for enwik8 and Table 2 for WikiText-103).

5 Results

Baseline, Tr-XL, RMT perform perfectly in the single segment setting on copy and reverse tasks (Figure 4). In this case models do not need recurrence because the whole sequence is available. When the number of segments is larger than one, non-recurrent baseline struggles to solve tasks, but both memory models demonstrate ability to retain required information from the previous segments in memory.

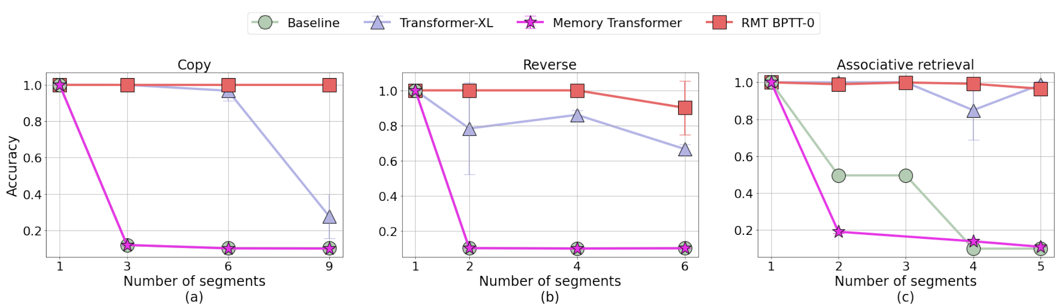


Figure 4: **RMT outperforms Transformer-XL on Copy and Reverse tasks as number of segments increases.** Panels show test set per-character accuracy on copy, reverse, and associative retrieval tasks (from left to right). Memory/cache size equals to the length of a segment for the both models. RMT does not pass gradients between segments in this experiment. MT results are the same as for the Baseline. Source/target sequence lengths for copy, reverse and associative retrieval tasks: 24/48, 24/24, 10/1.

As the number of segments increases, Recurrent Memory Transformer starts to outperform Transformer-XL with memory sizes less than the number of all previous tokens. With the number of segments up to 6 mean accuracy of Transformer-XL drops by up to 0.2 points, and with 9 segments score plunges to an accuracy score of 0.2, close to the baseline without memory.

Associative retrieval results are similar with the number of segments up to 4. RMT manages to solve the task with Transformer-XL closely behind. However, in the setting with 5 segments, RMT performance slightly decreases and Transformer-XL average accuracy rises higher.

On the quadratic equations task (Table 1) we have checked that it is possible to solve the task with the Transformer baseline and no segmentation used. The baseline in this case defines upper bound

[2]`https://github.com/kimiyoung/transformer-xl`

for this task. Enabling recurrence with multiple segments RMT solves the task perfectly, while Transformer-XL finds the task challenging.

Results of experiments on word-level language modeling on WikiText-103 are shown in Table 2. In the first section with a segment length of 150, Tr-XL and RMT outperform the baseline and Memory Transformer (MT) by a large margin. It shows the significance of increased effective context length by Tr-XL cache or RMT memory for language modeling.

Table 1: Test set solve rate for quadratic equations. Input sequence length is 180 tokens and consists of quadratic equation, solution, and answer. The input sequence is split into a number of segments with an answer as the last segment. Accuracy is 1.0 if the full answer is predicted correctly.

Model	memory	segments	Acc$_{\pm \text{std}}$
Baseline	0	1	0.99 $\pm$ 0.01
Tr-XL	30	6	0.93 $\pm$ NA
RMT	30	6	0.99 $\pm$ 0.002

MT is slightly better than the Transformer baseline. This is due to the fact that MT adds special memory tokens only to the beginning of an input sequence. Autoregressive MT has no way to write to memory because of the causal attention mask and, therefore, is unable to pass information between segments. Thus, in the autoregressive setting MT could be seen as equivalent to the prefix/promt-tuning (Li and Liang, 2021; Lester et al., 2021).

Table 2: Test set perplexity on WikiText-103. Average perplexity for the best performed variations of RMT models reported (see full results in Appendix A.5). Underlined values show Tr-XL and RMT models with close results. RMT models with smaller memory sizes achieve similar scores to Tr-XL models with larger memory.

Model	memory	segment len	PPL$_{\pm \text{std}}$
Baseline	0	150	29.95 $\pm$ 0.15
MT	10	150	29.63 $\pm$ 0.06
Tr-XL (paper)	150	150	**24.0**
Tr-XL (ours)	150	150	24.12 $\pm$ 0.05
Tr-XL	75	150	$\underline{24.68}$ $\pm$ 0.01
Tr-XL	25	150	25.57 $\pm$ 0.02
RMT BPTT-3	10	150	$\underline{25.04}$ $\pm$ 0.07
RMT BPTT-2	25	150	$\underline{24.85}$ $\pm$ 0.31
Baseline	0	50	39.05 $\pm$ 0.01
Tr-XL	100	50	25.66 $\pm$ 0.01
Tr-XL	50	50	$\underline{26.54}$ $\pm$ 0.01
Tr-XL	25	50	27.57 $\pm$ 0.09
Tr-XL	10	50	$\underline{28.98}$ $\pm$ 0.11
RMT BPTT-1	1	50	$\underline{28.71}$ $\pm$ 0.03
RMT BPTT-3	10	50	$\underline{26.37}$ $\pm$ 0.01

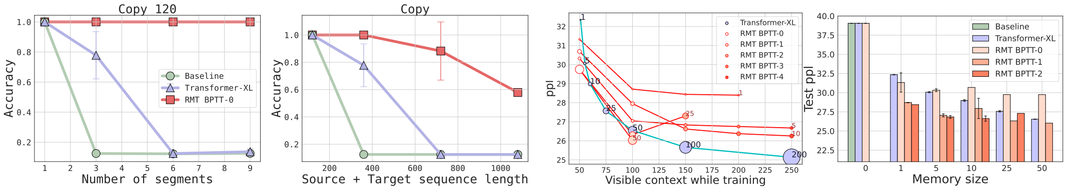
RMT improves over MT memory mechanism with read/write blocks. The best RMT models with memory size 10 and 25 show similar performance as Transformer-XL with a memory size equal to 75. RMT learns to use smaller memory more effectively than Transformer-XL. Additionally, the smaller memory size of RMT leads to reducing required GPU memory for running the model.

Decreasing the size of segments to 50, we force models to work with longer recurrent dependencies as the number of recurrent steps increases. RMT with memory consisting of a single vector shows similar results to Transformer-XL with memory size 10. It is worth noting that Transformer-XL memory consists of hidden representations from all layers (in this case, it is 10×16 vectors) when RMT memory is only `memory_size` vectors. Transformer-XL with memory size 50 and RMT with memory size 5 shows similar perplexity values.

On enwik8 RMT models with memory size 5 and Transformer-XL with memory size 40 show similar results. Confirming that RMT learns to use smaller amounts of memory representation more effectively. All experiments results on enwik8 dataset are in Appendix A.4 and Table 3.

We analyze how number of segments, sequence length, length of training context, and memory size affect models' performance on different tasks in Figure 5. As we split sequence into more segments it becomes more crucial to be able to pass information between segments. For the copy task, we split 360 tokens sequence into multiple segments. In Figure 5a we observe that Tr-XL performance starts to degrade and eventually falls to the baseline model performance as the number of segments increases. In contrast, RMT continues to solve the task perfectly. In the more extreme setting, when we keep memory size fixed, but increase the total length of sequence Tr-XL fails shortly, while RMT starts to degrade on 720 tokens sequence length (Figure 5b).

Recurrent Memory Transformer learns to make predictions depending on `#BPTT_unrolls` previous segments $+1$ current segment. Transformer-XL does not use BPTT and relies only on `memory_size` cached states and current segment making in total: `memory_size` $+$ `segment_length` tokens. In



(a) Increasing number of segments for a fixed sequence length of 360 tokens.

(b) Increasing sequence length up to 1080 tokens with 120 tokens segment and memory size 60.

(c) Dependency of test set perplexity on visible context length at training time.

(d) Increased memory size and deeper recurrence result in better performance.

Figure 5: (a) RMT is able to solve copy task perfectly with multiple recurrent steps, while Tr-XL fails. (b) RMT learns to use memory of the same fixed size more effectively than TR-XL as sequence length increases. (c&d) WikiText-103 with 50 tokens segment length. (c) Marker size corresponds to memory size. Visible context at training time can be increased by enlarging Tr-XL cache or using more BPTT unrolls for RMT. Increasing visible context leads to lower perplexity for both models. (d) Test set perplexity for different memory sizes. When memory size is zero Tr-XL and RMT are just baseline Transformer models without recurrence.

Figure 5c, we compare RMT and Tr-XL according to the described value of visible context at training time.

RMT with a single memory vector could be trained to achieve lower perplexity as Transformer-XL with memory size 10. It means that RMT can learn to store information from previous observations more compactly. Another observation is that RMT with memory sizes 10 and 25 performs worse but closely to Transformer-XL even when Transformer-XL has access to more non-compressed states (50, 100, 200) from previous segments. Furthermore, we observed instabilities and out-of-memory issues during RMT training for a larger number of BPTT unrolls and memory sizes.

Recurrent Memory Transformer does not benefit from increasing memory size from 5 to 50, but results of Transformer-XL better scale with memory size (Figure 5d). RMT models with memory size 5 have close results to Transformer-XL with cache 50, confirming that RMT learns to store more compact representations. Dynamic of RMT perplexity suggests that there is some optimal memory size for RMT to solve the task, and further increase does not add much. Training RMT with one BPTT unroll drastically improves its results showing the importance of BPTT training (Figure 5d).

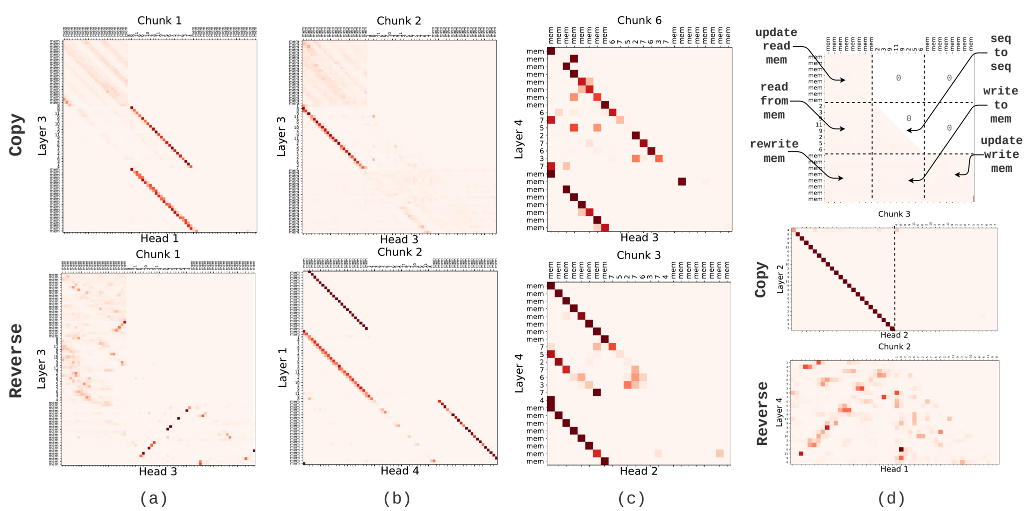


Figure 6: Selected attention map patterns of memory models. (a) - write to memory, (b) - read from memory (RMT, segment length=24, memory size=24), (c) - rewrite from read memory to write memory (RMT, segment length=8, memory size=8), (d) - read from previous hidden states (Transformer-XL, segment length=24, memory size=24)

To get an understanding of memory operations, learned by models during training algorithmic tasks, it is useful to look at attention maps. Figure 6 shows some heatmaps from attention layers of models trained on copy and reverse. The darkness of each pixel shows how much the element from the corresponding row "attends" to its column.

In each RMT attention map sequence tokens are preceded by read memory, located at the top left corner, and followed by write memory at the bottom right. Lines at the central part of (a), top image shows classic attention of token sequence to itself, but the bottom line represents the operation of writing of sequence tokens to memory in straight order. When completing reverse, the (a), bottom image model learns to write the sequence in the reversed order, which is in line with common sense.

When it comes to reproducing the target sequence, model accesses memory, Figure 6 (b) and writes to the output sequence. Another operation (c) is rewriting from read memory to write memory. It is commonly used by RMT in settings with larger number of segments to keep information about recent segments longer.

Transformer-XL mechanism of accessing memory (d) does not allow straightforward writing to memory without changing sequence token representations. Sequential reading from cache is represented by straight lines on Transformer-XL attention maps.

Using token representations as storage harms model performance in tasks with larger number of segments. On reverse with 4 segments Transformer-XL with limited memory size 6, Figure 8 (b) attempts to mix representations of tokens and read multiple symbols from one cached state in next segments. This results in the average accuracy of 0.8 on the given task. Despite having the same memory size, RMT manages to compress the whole segment in memory tokens Figure 8 (a) and achieve mean accuracy 1.0.

Visualizations from Figure 6 and Figure 8 provide evidence to support our hypotheses that Tr-XL has to mix representations from previous and current segments in the same hidden states to pass information between segments. Also, visualizations show how memory tokens in RMT help mitigate such kind of mixing.

RMT ability of sequence compression to memory is illustrated in Figure 7. For copy with 6 segments RMT compresses and then reads the sequence of 12 tokens with just 6 memory tokens. For Transformer-XL decreasing memory size harms the accuracy score significantly with number of segments larger than 2.

6 Conclusions

In this paper we introduced Recurrent Memory Transformer a simple recurrent memory augmentation of Transformer model. RMT is implemented by extension of an input sequence with special global memory tokens and segment-level recurrence.

In our experiments we compared RMT with Transformer baseline and Transformer-XL which is a well-known modification of Transformer for long sequences. RMT almost perfectly solves Copy, Reverse as well as quadratic equations tasks for sequences consisting of multiple segments outperforming Transformer-XL. It also demonstrates quality for associative retrieval task on par with Transformer-XL. As expected, baseline Transformer fails to solve these tasks for multi-segment settings.

RMT trained as language model significantly ahead of Transformer baseline and shows quality metrics similar to Transformer-XL but for up to 10 times smaller memory size. Experimental results demonstrate that for fixed memory size backpropagating gradients for more segments improves performance of RMT. Analysis of attention maps suggests that better RMT performance can be related to more effective storage of input representations in dedicated memory tokens compared to mixing representations storage in Transformer-XL. Overall, results of the study show that dedicated memory storage and recurrence provided by Recurrent Memory Transformer make it a promising architecture for applications that require learning of long-term dependencies and general purpose in-memory processing, such as algorithmic tasks and reasoning. Furthermore, we believe that RMT could open the way for adding memory and recurrence to other models in the Transformer family.

References

Joshua Ainslie, Santiago Ontanon, Chris Alberti, Philip Pham, Anirudh Ravula, and Sumit Sanghai. Etc: Encoding long and structured data in transformers, 2020.

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL `https://proceedings.neurips.cc/paper/2016/file/9f44e956e3a2b7b5598c625fcc802c36-Paper.pdf`.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Mikhail S Burtsev, Yuri Kuratov, Anton Peganov, and Grigory V Sapunov. Memory transformer. *arXiv preprint arXiv:2006.11527*, 2020.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers, 2019.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-4012. URL `https://aclanthology.org/W14-4012`.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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, 2019. URL `https://aclweb.org/anthology/papers/N/N19/N19-1423/`.

SiYu Ding, Junyuan Shang, Shuohuan Wang, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. ERNIE-Doc: A retrospective long-document modeling transformer. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2914–2927, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.227. URL `https://aclanthology.org/2021.acl-long.227`.

Linhao Dong, Shuang Xu, and Bo Xu. Speech-transformer: A no-recurrence sequence-to-sequence model for speech recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5884–5888, 2018. doi: 10.1109/ICASSP.2018.8462506.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=YicbFdNTTy`.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. Addressing some limitations of transformers with feedback memory. *arXiv preprint arXiv:2002.09402*, 2020.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John 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, October 2016. ISSN 00280836. URL `http://dx.doi.org/10.1038/nature20101`.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory, 2015.

Caglar Gulcehre, Sarath Chandar, Kyunghyun Cho, and Yoshua Bengio. Dynamic neural turing machine with soft and hard addressing schemes. *arXiv preprint arXiv:1607.00036*, 2016.

Caglar Gulcehre, Sarath Chandar, and Yoshua Bengio. Memory augmented neural networks with wormhole connections. *arXiv preprint arXiv:1701.08718*, 2017.

Qipeng Guo, Xipeng Qiu, Pengfei Liu, Yunfan Shao, Xiangyang Xue, and Zheng Zhang. Star-transformer, 2019.

Ankit Gupta and Jonathan Berant. Gmat: Global memory augmentation for transformers. *arXiv preprint arXiv:2006.03274*, 2020.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `https://doi.org/10.1162/neco.1997.9.8.1735`.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021.

Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets, 2015.

Da Ju, Stephen Roller, Sainbayar Sukhbaatar, and Jason Weston. Staircase attention for recurrent processing of sequences. *arXiv preprint arXiv:2106.04279*, 2021.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys, 2019.

Jie Lei, Liwei Wang, Yelong Shen, Dong Yu, Tamara L. Berg, and Mohit Bansal. Mart: Memory-augmented recurrent transformer for coherent video paragraph captioning, 2020.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.243. URL `https://aclanthology.org/2021.emnlp-main.243`.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353. URL `https://aclanthology.org/2021.acl-long.353`.

Matt Mahoney. Large text compression benchmark, 2006. URL `http://www.mattmahoney.net/dc/text.html`.

Pedro Henrique Martins, Zita Marinho, and André FT Martins. ∞-former: Infinite memory transformer. *arXiv preprint arXiv:2109.00301*, 2021.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Yuanliang Meng and Anna 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.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=Byj72udxe`.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018. URL `https://www.cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf`.

Jack W Rae, Jonathan J Hunt, Tim Harley, Ivo Danihelka, Andrew Senior, Greg Wayne, Alex Graves, and Timothy P Lillicrap. Scaling memory-augmented neural networks with sparse reads and writes, 2016.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling, 2019.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8821–8831. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/ramesh21a.html`.

C Stephen. Kleene. representation of events in nerve nets and finite automata. *Automata studies*, 1956.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. URL `http://papers.nips.cc/paper/7181-attention-is-all-you-need`.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10): 1550–1560, 1990.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks, 2014.

Qingyang Wu, Zhenzhong Lan, Jing Gu, and Zhou Yu. Memformer: The memory-augmented transformer. *arXiv preprint arXiv:2010.06891*, 2020.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We mention training instabilities and GPU RAM issues in Section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [No] The proposed model and method do not have any specific impacts. All general negative societal impacts applicable to the field could be potentially relative.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We include code, training scripts, and raw experimental data in the supplementary material. The supplemental materials would be published on github with the final version of the paper. Instructions for language modeling data&experiments are taken from Tr-XL repo.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4.3, Appendix A, and provided supplementary material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All the key experiments results are reported with std. Furthermore, we provide raw experimental data in the supplementary materials.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We used different GPUs depending on the task: 1080Ti, V100, A100. We provide this information in Appendix A for each task.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We refer to the original Tr-XL code and Tr-XL paper. We use it for establish baselines and setting our methods. See Section 4.3
   (b) Did you mention the license of the assets? [No] Tr-XL licence is Apache 2.0 and available at its github repo.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Our code is in the supplemental material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] We used publicly available Tr-XL code (Apache 2.0) and datasets.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] We use either synthetic data or datasets collected from the Wikipedia (Wikitext-103, enwik8).
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Training details and additional results

A.1 Algorithmic tasks

Datasets were randomly generated by uniformly sampling tokens from dictionary into task sequences and generating targets accordingly to the tasks. After generation, datasets are fixed for all experiments.

Copy and reverse use sequences of sizes 24, 40, 120, 240, and 360, making total copy/reverse input length 48/72, 80/120, 240/360, 480/720, 720/1080. The associative retrieval task consists of 4 key-value pairs and one randomly selected key; the answer consists of one value. Train, validation and test sizes of copy 24, reverse 24 and associative retrieval datasets are 100000, 5000 and 10000.

Transformer-XL had the same cache size on training and validation to match RMT.

For training all models on copy and reverse, we used constant learning rate 1e-4 with reduction on plateau with decay factor of 0.5. Copy and reverse were solved by models with 4 layers and 4 heads, associative retrieval models had 6 layers and 4 heads. Models with the same context size and memory size were trained for the same number of steps and the same training parameters.

Experiments with sequence length 24 were conducted on a single Nvidia GTX 1080 Ti GPU from 1 hour to 2-3 days. Copy and reverse on longer sequence lengths were done on more powerful Tesla V100 using 1-3 devices with training time varying from 1 hour to 3-4 days.

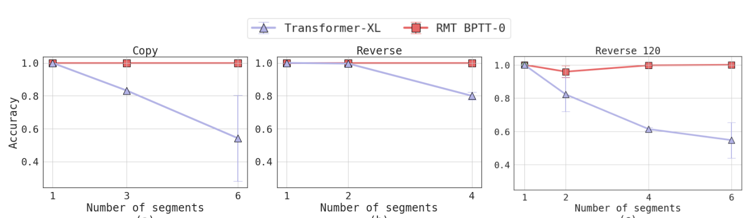


Figure 7: Test set per-character accuracy on copy (a), reverse with a sequence length 24 (b) and 120 (c). Memory size is limited to half of the length of a segment for (a) and (b) and to 60 tokens for (c). Target sequence length equals source length for reverse and doubles for the copy.

A.2 Associative retrieval

We used code for the task dataset generation from (Ba et al., 2016)[3].

A.3 Quadratic equations

This dataset consists of equations with integer coefficients with step-by-step solutions using a discriminant. Process of equation generation is started from uniformly sampling real roots x_1, x_2 from -100 to 100. The answer of an equation is represented as x_1, x_2. Next, we find the equation as multiplication of two parentheses $(x - x_1)(x - x_2) = 0$, which is expanded to $x^2 - (x_1 + x_2)x + x_1x_2 = 0$. Next, we multiply all coefficients by a random natural number α from 1 to 10. The final equation form is $\alpha x^2 - \alpha(x_1 + x_2)x + \alpha x_1x_2 = 0$. A dataset sample is made of these stages in reversed order. We also provide a string with the discriminant calculation to help find the equation roots. 20 percent of equations in the dataset do not have real roots.

Example equation string:

*-4*x^2+392*x-2208=0*,

solution string:

*x^2-98*x+552=0;D=98^2-4*1*552=7396=86^2;x=(98-86)/2=6;x=(98+86)/2=92* ,

and answer:

6,92

Each solution step is tokenized on char level and padded to the length of 30 tokens. The total length of each training sample is 180, the dataset has 100000 training, 10000 validation and 20000 test samples.

For this task we used models with 6 layers, 6 heads and segment sizes 180 and 30. Trained was performed with the same schedule as copy and reverse on a single GTX 1080 ti for 1-2 days. Memory size for RMT and Transformer-XL was chosen equal to the segment length.

A.4 Enwik8

We verified our experimental setup by reproducing Transformer-XL results on enwik8 dataset (Table 3). We used 12-layer Baseline (Transformer), Transformer-XL, RMT in all enwik8 experiments. All results on enwik8 dataset are in Table 3. We used 2 NVIDIA A100 80Gb GPUs, training time varied from 10 to 30 hours depending on sequence length, memory size, and number of BPTT unrolls.

A.5 WikiText-103

We used 16-layer models in all experiments on WikiText-103 dataset. Training hyperparameters were used from (Dai et al., 2019) and authors PyTorch scripts[4]. All results on WikiText-103 dataset are in Table 4. In most of the WikiText-103 experiments, we used 2 NVIDIA A100 80Gb GPUs, training time varied from 10 to 30 hours depending on sequence length, memory size, and number of BPTT unrolls.

[3]`https://github.com/GokuMohandas/fast-weights/blob/539fb10e3c384d5f782af2560bf28631cd0eaa61/fw/data_utils.py`

[4]`https://github.com/kimiyoung/transformer-xl`

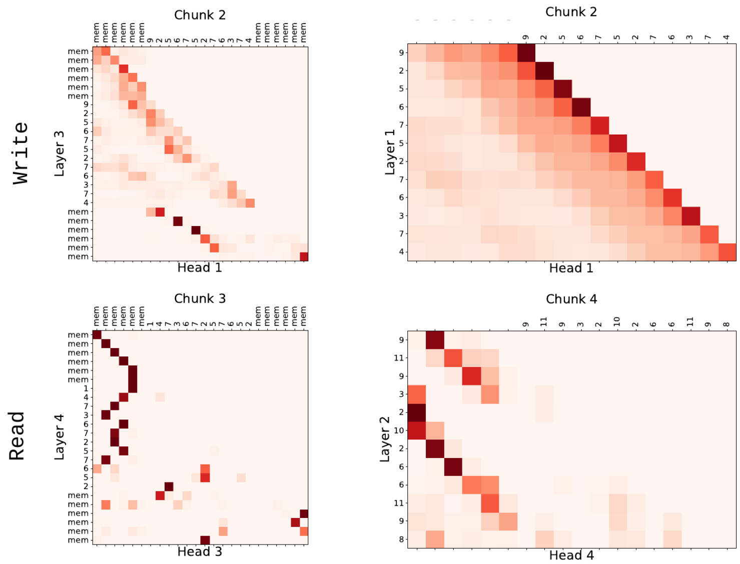
Table 3: Test set bits-per-character on enwik8. Our experimental setup shows similar scores to the original paper (Dai et al., 2019) with segment length 512.

MODEL	MEMORY	SEGMENT LEN	BPC$_{\pm \text{STD}}$
TR-XL (DAI ET AL., 2019)	512	512	1.06
TR-XL (OURS)	512	512	1.071
TR-XL	200	128	1.140
TR-XL	100	128	1.178
TR-XL	75	128	1.196
TR-XL	40	128	1.230 $\pm$ 0.001
TR-XL	20	128	1.261
TR-XL	10	128	1.283 $\pm$ 0.001
RMT BPTT-1	5	128	1.241 $\pm$ 0.002
RMT BPTT-2	5	128	1.231 $\pm$ 0.002
RMT BPTT-1	10	128	1.240 $\pm$ 0.006
RMT BPTT-2	10	128	1.228 $\pm$ 0.003
RMT BPTT-0	20	128	1.301
RMT BPTT-1	20	128	1.229
RMT BPTT-2	20	128	1.222

Table 4: Test set perplexity on WikiText-103. All experiments with RMT and Tr-XL models.

Model	memory	segment len	PPL$_{\pm \text{std}}$
Baseline	0	150	29.95 $\pm$ 0.15
MT	10	150	29.63 $\pm$ 0.06
MT	25	150	29.67 $\pm$ 0.03
MT	75	150	29.69 $\pm$ 0.02
MT	150	150	29.82 $\pm$ 0.35
Tr-XL (paper)	150	150	24.0
Tr-XL (ours)	150	150	24.12 $\pm$ 0.05
Tr-XL	75	150	24.68 $\pm$ 0.01
Tr-XL	25	150	25.57 $\pm$ 0.02
RMT BPTT-0	10	150	26.85 $\pm$ 0.02
RMT BPTT-1	10	150	25.92 $\pm$ 1.07
RMT BPTT-2	10	150	25.32 $\pm$ 0.61
RMT BPTT-3	10	150	25.04 $\pm$ 0.07
RMT BPTT-0	25	150	29.73
RMT BPTT-1	25	150	24.91
RMT BPTT-2	25	150	24.85 $\pm$ 0.31
Baseline	0	50	39.05 $\pm$ 0.01
Tr-XL	200	50	25.14
Tr-XL	100	50	25.66 $\pm$ 0.01
Tr-XL	50	50	26.54 $\pm$ 0.01
Tr-XL	25	50	27.57 $\pm$ 0.09
Tr-XL	10	50	28.98 $\pm$ 0.11
Tr-XL	5	50	30.06 $\pm$ 0.07
Tr-XL	1	50	32.35 $\pm$ 0.03
RMT BPTT-0	1	50	31.33 $\pm$ 1.26
RMT BPTT-1	1	50	28.71 $\pm$ 0.03
RMT BPTT-2	1	50	28.44
RMT BPTT-3	1	50	28.40 $\pm$ 0.03
RMT BPTT-0	5	50	30.32 $\pm$ 0.18
RMT BPTT-1	5	50	27.05 $\pm$ 0.20
RMT BPTT-2	5	50	26.83 $\pm$ 0.18
RMT BPTT-3	5	50	26.75 $\pm$ 0.26
RMT BPTT-4	5	50	26.67 $\pm$ 0.03
RMT BPTT-0	10	50	30.69 $\pm$ 0.01
RMT BPTT-1	10	50	27.95 $\pm$ 1.32
RMT BPTT-2	10	50	26.62 $\pm$ 0.34
RMT BPTT-3	10	50	26.37 $\pm$ 0.01
RMT BPTT-4	10	50	26.25 $\pm$ 0.19
RMT BPTT-0	25	50	29.75
RMT BPTT-1	25	50	26.32
RMT BPTT-2	25	50	27.31
RMT BPTT-0	50	50	29.75
RMT BPTT-1	50	50	26.03

B Operations with Memory



(a) (b)

Figure 8: Approaches to compression and decompression of sequence with length 12 and memory with size 6. (a) - RMT, (b) - Transformer-XL.