NEURAL ATTENTION MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Scaled dot-product attention has become the essence of state-of-the-art deep neural networks for variety of machine learning tasks. We propose a novel perspective of the attention mechanism by reinventing it as a memory architecture for neural networks, namely neural attention memory (NAM). Following the same query-key-value structure, NAM writes a memory matrix via sum of outer products of value and unit key vectors. Reading the memory can be done by multiplying the matrix with a unit query vector. It is possible to implement a mechanism theoretically equivalent to attention with NAM primitives, namely normalized outer-product attention. One benefit of NAM is that its computational complexity is linear to the sequence length while the scaled dot-product attention's complexity is quadratic to it. We evaluate a NAM-based Transformer on long-range arena tasks and show its efficiency and efficacy. Also, we design two NAM-based memory-augmented neural networks namely Long short-term attention memory (LSAM) and NAM Turing machine (NAM-TM). LSAM replaces LSTM's long-term cell state with NAM memory matrix and NAM-TM implements a Turing tape data structure using NAM read/write primitives. We test their capability using four different types of compositional generalization tasks. The experimental results show that both LSAM and NAM-TM outperform traditional Transformer and LSTM, as well as DNC, a complicated memory-augmented neural network model. Moreover, NAM opens up possibilities in a variety of machine learning research problems including hierarchical data modeling, efficient edge inference, and few-shot learning.

1 INTRODUCTION

Scaled dot-product attention (Vaswani et al., 2017) has become a core mechanism of state-of-the-art deep learning models for variety of machine learning tasks, including natural language processing (Devlin et al., 2018), multi-modal task (Li et al., 2019), and graph data processing (Hamilton et al., 2017). Specifically, the Transformers using the self-attention method based on it has replaced recurrent neural network (RNN) based methods by outperforming them. Despite its success, there exist limitations to the mechanism. First, it needs the information of the entire sequence to compute one attention so that its computational complexity becomes quadratic to the length of the sequence. Hence, it is inefficient for long sequence tasks (Tay et al., 2020) or edge inference environments (Tambe et al., 2020). Also, its stateless design enables efficient parallelism but makes it impossible to solve tasks that require memory states. Hence, Transformers fail to generalize the rules that require inductive bias (Dehghani et al., 2018) or compositional generalization (Lake & Baroni, 2018).

There have been studies designing neural networks with external memory to solve algorithmic tasks where Transformers fail. These memory-augmented neural networks (MANN) design differentiable read/write functions that can be trained by back-propagation. Some of them implement basic data structures like stack (Joulin & Mikolov, 2015) and queue (Grefenstette et al., 2015) while some implement complex memory structures using attention mechanisms (Graves et al., 2014; 2016). They outperform generic neural networks in synthetic algorithmic tasks but are considered impractical due to their complexities and inefficiencies.

In this work, we re-invent the attention mechanism as a memory architecture for neural networks, namely *neural attention memory* (NAM). NAM's design objective is to build simple, efficient, yet powerful external memory which also incorporates the attention mechanism. Following the same query-key-value structure of attention, NAM stores key-value pairs to a *memory matrix* via additively writing their outer-products. Reading the memory matrix is simply done by multiplying the matrix

with a unit query vector. We provide mathematical formulation for the read/write primitives, and make theoretical analyses showing that these read and write primitives can replace attention.

One big benefit of NAM is that it can perform attention in a more efficient way. NAM-based attention, namely normalized outer-product attention, has linear computational complexity to the sequence length, which is a goal for efficient long-range Transformers (Choromanski et al., 2020; Beltagy et al., 2020; Kitaev et al., 2020). Mathematically, it is very similar to linear attention (Katharopoulos et al., 2020) in that it computes the multiplication of keys and values first to reduce the complexity. We evaluate NAM-based efficient Transformer in long-range arena (Tay et al., 2020) tasks. Its efficacy is on par with the base Transformer and Linear Transformer, implying that NAM can be an efficient alternative to scaled dot-product attention.

Using NAM read/write primitives, we design two memory-augmented neural networks (MANN), namely Long short-term attention memory (LSAM) and NAM Turing machine (NAM-TM). LSAM is a generic RNN architecture that replaces LSTM's long-term cell state with a memory matrix. Instead of additively writing a vector cell state, LSAM reads and writes the memory matrix using NAM primitives. The design combines strengths of attention and RNN while maintaining the same computational complexity as LSTM. NAM-TM is a MANN for algorithmic tasks, constructing a Turing tape structure. A tape has read and write heads accessing the memory with NAM read/write primitives. They can move along the tape with four actions: NO-OP, LEFT, RIGHT, and JUMP. The actions are implemented as differentiable functions to enable end-to-end training with backpropagation.

We compare LSAM and NAM-TM to others in compositional generalization tasks of number sequence prediction (Nam et al., 2019), sequence reduction, and SCAN (Lake & Baroni, 2018). Specifically, we test their zero-shot generalization capability in length by training the models with shorter sequences and validating them with longer sequences unobservable during training. The evaluation results show that their computational powers are superior to other neural networks, including Universal Transformer (Dehghani et al., 2018) and DNC (Graves et al., 2016). While the generic LSAM model consistently outperforms the others, NAM-TM shows even better results at algorithmic tasks. The results indicate that NAM is a powerful method to implement memory in neural networks.

The efficient, simple, and flexible structure of NAM opens up new possibilities in multiple machine learning research fields. One straightforward application is leveraging NAM's efficiency for edge inference environment. Another possibility is using NAM for hierarchical data modeling by generalizing NAM with *tensor product*. Moreover, memorization of input-output mapping using NAM can be a solution for few-shot learning.

The main contributions of this work are as follows:

- We re-invent the attention mechanism as a memory architecture for neural networks, namely neural attention memory (NAM).
- We present mathematical basis for NAM read/write primitives, and give theoretical proofs that NAM is equivalent to attention in certain conditions.
- We show that NAM can construct an efficient Transformer for long-range sequence tasks.
- We propose two memory-augmented neural network designs of LSAM and NAM-TM and show their capabilities in compositional generalization tasks.

2 Background

2.1 Scaled dot-product attention

Attention mechanisms of deep neural networks (Bahdanau et al., 2014; Luong et al., 2015) provide differentiable methods of choosing (attending) one item from a variable-length sequence. While there are multiple variations of attention mechanism, most of them share the same high-level structure: 1) compute the attention scores of the items, and 2) return the weighted sum of their vector representations using the scores. Among the variations, *scaled dot-product attention* (Vaswani et al., 2017) has been the most successful. For each token, there are a key vector and a value vector associated to it. Given a query vector, the scores are determined by the scaled dot-product of the query and the keys. Then the output is computed by weighted sum of softmaxed scores and the value vectors.

The self-attention mechanism based on the scaled dot-product attention has proven to be very powerful, even replacing the needs of recurrent neural networks (RNN). Since attention reaches every element of a sequence in an $O(1)$ path, it avoids the vanishing gradient problem (Hochreiter, 1998), enjoys high parallelism, and allows huge models with deeply stacked layers (Devlin et al., 2018; Brown et al., 2020). However, its stateless and parallel architecture also bring multiple limitations. First, the computational cost quadratically increases with the sequence length, making it inefficient for long-range contexts (Beltagy et al., 2020; Choromanski et al., 2020; Kitaev et al., 2020; Katharopoulos et al., 2020) and edge inference environments (Tambe et al., 2020). Also, the memory-less architecture lacks inductive bias (Dehghani et al., 2018), making it impossible to generalize inductive algorithmic rules (Kim et al., 2021). There are multitudes of researches to resolve those issues, but a simple and practical solution is yet to be found.

2.2 Memory-augmented neural network

In theory, RNNs are proven to be as powerful as Turing machines (Hyötyniemi, 1996). However, they fail to learn algorithmic patterns that require more that pushdown automata in practice (Nam et al., 2019). Therefore, there have been efforts to augment external memory architecture to neural networks, as known as memory-augmented neural networks (MANN) (Grefenstette et al., 2015; Joulin & Mikolov, 2015; Graves et al., 2014; 2016). The main challenge for MANNs is to design differentiable read and write functions that can be trained via back-propagation. Some MANNs use attention mechanism for differentiable read/write functions. For instance, neural turing machine (NTM) (Graves et al., 2014) and differentiable neural computer (DNC) (Graves et al., 2016) leverage attention mechanism for implementing content-based addressing. However, the addressing mechanisms are often not powerful enough so that they need to be augmented with complex extras, such as a link matrix with $O(N^2)$ cost. Hence, MANNs often become inefficient and complex so that they are considered impractical outside of the algorithmic task domain.

3 Neural Attention Memory

In this section, we define a memory structure reconstructed from the attention mechanism, namely neural attention memory (NAM). We mathematically define the read/write primitives for NAM, and give theoretical proof that it can replace attention.

The main idea of NAM is implementing an attention mechanism via matrix-vector multiplication of a *memory matrix* $M \in \mathbb{R}^{d_v \times d_k}$, a *unit query vector* $q \in \mathbb{R}^{d_k}$ and a *read probability* $0 \leq p_r \leq 1$. Hereby d_v and d_k are feature dimensions of value and key vectors respectively. Then, the read operation (RD) of NAM is defined as follows.

$$r = RD(M, q, p_r) = p_r M q$$

While there can be multiple ways to construct the memory M, the simplest method is *sum of outer products* (or tensor products) of *unit key vectors* $k_i \in \mathbb{R}^{d_k}$ and *value vectors* $v_i \in \mathbb{R}^{d_v}$ as follows.

$$M = \sum_{1}^{S} v_i \otimes k_i = \sum_{1}^{S} v_i k_i^\top$$

Hereby S denotes the number of tokens (or length) in a sequence. Since both query and key vectors are *unit vectors*, reading the memory matrix can act as an attention mechanism when the keys are orthonormal as the following theorem holds.

Theorem 1. *If $k_1 \dots k_S$ are orthonormal unit vectors and i = 1 ... S, RD($M, k_i, 1$) = v_i*

The proof of the theorem is trivial because $v_j k_j^\top k_i$ is v_i when $j = i$ and zero otherwise. However, it is hard to guarantee that all key vectors are orthonormal to each other. Hence, a more general write (*WR*) operation can be defined with a write probability p_w and an erase probability p_e as below.

$$M' = WR(M, k, v, p_w, p_e) = M + p_w v k^\top - p_e M k k^\top$$

Verbally speaking, this write operation adds the new key-value pair $vk^\top$ to the matrix while erasing the existing value Mk. The sum of outer products $\sum v_i k_i^\top$ is a special case of this when $p_w = 1$ and $p_e = 0$ for every k_i and v_i. It provides a softer guarantee as follows, without requiring orthonormality.

Theorem 2. *If k is a unit vector and $M' = WR(M, k, v, 1, 1)$, $RD(M', k, 1) = v$.*

Again, the proof is trivial because $M'k = Mk + vk^\top k - Mkk^\top k = v$. In other words, after a *WR* operation, the RD operation with the same key yields the most recently written value.

These simple and powerful read/write primitives are also efficient. First of all, the computational complexity of each read/write operation is $O(d_k \times d_v)$ which is identical to complexity of other neural network layers and does not grow with the sequence length S. Moreover, the outer product operation can be efficiently computed with modern hardware as it high compute density $O(d_k \times d_v)$ and low amount of memory access $O(d_k + d_v)$. Finally, the space complexity of the memory matrix $O(d_k \times d_v)$ also does not rely on the sequence length, which makes it desirable at resource-constrained edge environment.

4 Efficient Transformer with NAM

4.1 Normalized outer-product attention

One popular limitation of the scaled dot-product attention is that its computational complexity is quadratic to the length of the sequence. This becomes problematic when handling the very long sequences (>1000 tokens) and deploying the Transformers for edge inference. One benefit of NAM is that its computation complexity is *linear* to to the length of the input sequence. Hence, we can use NAM as an efficient attention mechanism.

For scaled dot-product attention, we first compute *dot-products* (inner product) of the keys and the query to get the similarity scores. This computation has quadratic complexity to the sequence length because we need to compute the score of every key-query pair. On the other hand, NAM-based attention can reduce the complexity by changing the order of computation, namely *normalized outer-product attention*. As we do for NAM, the keys and queries are normalized to unit vectors to eliminate the softmax function. Then we first construct a memory matrix by computing sum of outer products of the keys and the values. Finally the attention result can be computed by reading the memory matrix with the queries.

Given the query vector $q \in \mathbb{R}^{d_k}$, and key-value sequences $K \in \mathbb{R}^{S \times d_k}$, $V \in \mathbb{R}^{S \times d_v}$ of length S, the normalized outer-product attention can be computed as follows.

$$Attention_{NAM}(K, V, q, p_w, p_r) = RD(V^\top(p_w \odot \mu(K)), q/||q||, p_r)$$

Hereby $\mu(.)$ is the unit vector normalization function, $p_w \in [0, 1]^S$, $p_r \in [0, 1]$ are write and read probabilities, and $\odot$ is element-wise multiplication. Note that $V^\top(p_w \odot \mu(K))$ is one of the special cases of constructing the memory matrix by setting the erase probabilities to zeros WR operations. It is possible to create a new type of attention by replacing it with a different writing method to construct the memory matrix.

4.2 Linear self-attention with NAM

Using normalized outer-product attention, we can implement an efficient self-attention. The simplest way is setting the read/write probabilities to 1. Given queries $Q \in \mathbb{R}^{S \times d_k}$, keys $K \in \mathbb{R}^{S \times d_k}$, and values $V \in \mathbb{R}^{S \times d_v}$ of a sequence with S tokens, NAM self-attention is computed as follows.

$$SelfAttn_{NAM}(Q, K, V) = (V^\top \mu(K))\mu(Q)$$

Its computational complexity is $O(Sd_kd_v)$ instead of $O(S^2d_k + Sd_v)$ of scaled dot-product self attention. While it is quadratic to the dimensions d_k, d_v, it is often not a problem in practice. Most of large Transformers use multiple attention heads and keep the per-head dimension d_k to small values like 64. Hence, we can keep the computational cost of each attention head and increase the capacity of the model by adding more heads.

Computationally, this is very similar to the self-attention of Linear Transformer (Katharopoulos et al., 2020). Linear Transformer has proven to be as effective as other Transformer implementations (Tay et al., 2020) and we expect our Transformer with NAM self-attention to show similar level of efficacy. The differences from Linear Transformer to ours are: 1) we use unit vector normalization instead of ELU kernel function, 2) so that we do not need to compute the causal masking factor Z (Katharopoulos et al., 2020).

4.3 LONG-RANGE TASK EVALUATION

Although normalized outer-product attention is theoretically capable of replacing scaled dot-product attention, there exists a danger of information loss due to the limited capacity of the memory M. Meanwhile, such an information loss is not an issue for Transformers because they utilize hidden activations from the entire sequence. Also, the orthogonality assumption of the keys cannot hold in practice so that it may suffer from noises. Hence we need empirical evidences that NAM self-attention is an effective alternative.

Table 1: Accuracy (%) and relative training speedup comparison of the three Transformer models in listops (Listops), text classification (Text), pixel-level image classification (Image) tasks.

	Listops		Text		Image	
Model	Acc	Speedup	Acc	Speedup	Acc	Speedup
Transformer	29.75	1	57.28	1	40.16	1
Linear Transformer	36.5	2.59	64.27	2.35	38.76	10.02
NAM Transformer (Ours)	36.1	2.66	63.49	2.44	37.18	10.07

We test our NAM self-attention based Transformer on long-range arena tasks (Tay et al., 2020) that have been used to compare efficient Transformer architectures. Since the previous work has already proven that Linear Transformer is as capable as others, we conduct partial comparison of ours to the original Transformer and Linear Transformer. Table 1 shows that NAM can be an efficient alternative to Transformers. Despite the information loss and missing normalizing factor Z, the task accuracy of NAM is very similar to the others, even surpassing the vanilla Transformer in some cases. We see bigger speedups at the text classification task because the models have much smaller per-head dimension than others (16 vs 64). Recall that the computational complexities of both linear transformer and NAM transformer are quadratic to the per-head dimensions. Image classification result shows that smaller per-head dimension can bring bigger speedups for NAM, but may result in lesser capability. Overall, the normalized outer-product attention is as powerful as the scaled dot-product attention while enjoying greater computational efficiency.

We use the code base of the original benchmark [1] and implemented our NAM encoder by modifying the Linear Transformer implementation. The three models share the same hyperparameters and the only differences come from the attention algorithms. The evaluations are run on the Ubuntu 20.04 system with RTX 3080. The results differ from the original work because we used smaller batch sizes due to the limited VRAM capacity. All other experimental setups, including hyperparameters, are identical to the original benchmark (Tay et al., 2020). Details are available in the source code included as a supplementary material.

5 NAM AUGMENTED NEURAL NETWORKS

In this section, we design two types of memory-augmented neural networks (MANN) based on NAM, namely *Long Short-term Attention Memory* (LSAM) and *NAM Turing Machine* (NAM-TM). LSAM is a generic recurrent neural network architecture derived from LSTM (Hochreiter & Schmidhuber, 1997). In LSAM, the long-term cell state of LSTM is replaced with a NAM memory matrix. NAM Turing Machine is a MANN design for algorithmic tasks inspired by Neural Turing Machine (Graves et al., 2014). Its read and write heads can move along the tape with four actions: left, right, no-op, and jump. Their implementations are based on NAM read/write primitives. We evaluate the models with tasks of number sequence prediction, sequence reduction, and SCAN. We test their *compositional generalization* capability by splitting the data based on the sequence lengths.

5.1 LONG SHORT-TERM ATTENTION MEMORY

Long Short-term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) leverages two recurrent state vectors: the short-term hidden state and the long-term cell state. To mitigate the problems

[1] github.com/google-research/long-range-arena

of vanishing/exploding gradients, the cell state is additively updated using forget and input gates. Then the output gate selectively reads the cell state to determine the hidden state. Long Short-term Attention Memory (LSAM) follows the same principle. Instead of using the vector cell state, it leverages the memory matrix $M_t \in \mathbb{R}^{d^2}$ which is also additively updated using the NAM write primitive. The hidden state $h_t \in \mathbb{R}^d$ is retrieved by reading the memory matrix. Given the input $x_t \in \mathbb{R}^d$, the update rule $M_t, h_t = LSAM(x_t, M_{t-1}, h_{t-1})$ is defined as follows.

$$
\begin{aligned}
[q_t : k_t : v_t] &= W_{qkv}[x_t : h_{t-1}] + b_{qkv} \\
< p_r, p_w > &= \sigma(W_{rw}[x_t : h_{t-1}] + b_{rw}) \\
M_t &= WR(M_{t-1}, \mu(k_t), v_t, p_w, p_w) \\
h_t &= RD(M_t, \mu(q_t^i), p_r)
\end{aligned}
$$

Hereby : is the concatenate operator, $\sigma(.)$ is the sigmoid function, and $W_{qkv}, W_{rw}, b_{qkv}, b_{rw}$ are trainable weights and biases. Although the memory matrix M_t has much higher capacity than a vector cell state, the computational complexity of LSAM is identical to that of LSTM. This is because WR and RD have complexity of $O(d^2)$ which is identical to that of matrix-vector multiplication of the weights and the states. Like Transformers, we can design multi-headed LSAM by concatenating per-head states M_t^i and h_t^i. Bidirectional LSAM is also possible by splitting the multiple heads into two directions. The backward heads are updated in opposite by the rule of $M_t^j, h_t^j = LSAM(x_t, M_{t+1}^j, h_{t+1}^j)$.

The LSAM architecture combines the strengths of recurrent neural networks (RNN) and Transformers. Since LSAM follows the RNN design so that it enjoys strengths of RNNs such as low computational cost for inference and recurrent inductive bias. Additionally, it benefits from the strengths of attention because reading and writing the memory matrix natively incorporates the attention mechanism.

5.2 NAM Turing Machine

Neural Turing Machine (NTM) (Graves et al., 2014) is one of the early neural networks that implement external memory structure with differentiable read and write methods. It is a basis of Differentiable Neural Computer (DNC) (Graves et al., 2016) which has proven to be effective at solving a variety of algorithmic tasks such as answering synthetic questions and finding shortest paths. Their external memory matrix is accessed by *read and write heads* using differentiable attention mechanisms.

Hereby we design NAM Turing Machine (NAM-TM) which adopts the design principles of NTM and DNC. The main idea of NAM-TM is to treat the tape state $T = [v_1, v_2, ...v_n, \mathbf{0}, ...] \in \mathbb{R}^{L \times d}$ as a memory matrix as follows.

$$
T = [v_1, v_2, ...v_n, \mathbf{0}, ...] = \sum_i v_i e_i^\top \qquad (v_i \in \mathbb{R}^d, e_i \in \mathbb{R}^L)
$$

Hereby e_i are *standard basis vectors* $< 0, 0, ...1, ..., 0 >$ of $\mathbb{R}^L$ where L is the size of the tape. This tape state can now be accessed by using NAM read/write primitives.

NAM-TM is a differentiable function that takes the tape state T, read and write heads $H_r, H_w \in \mathbb{R}^L$, and the input vector $x \in \mathbb{R}^d$ and produces the read output $R \in \mathbb{R}^d$ along with the updated tape and head states T', H'_r, H'_w.

$$
R, T', H'_r, H'_w = NAMTM(T, H_r, H_w, x)
$$

The read and write heads are positional vectors to attend the memory matrix for reading and writing the states. At each time step, positions can be updated by four actions: LEFT, RIGHT, NO-OP, and JUMP. They are controlled by a controller neural network $nn_control(x)$ which emits read and write probabilities p_r, p_w, action probabilities $p_{right}, p_{left}, p_{noop}, p_{jump}$ for each head and a unit jump query vector $q_{jump} \in \mathbb{R}^d$. Given the controller outputs and the value $v = W_v x$ to write, reading and writing the memory are conducted by NAM primitives as follows.

$$
\begin{aligned}
R &= RD(T, H_r, p_r) \\
T' &= WR(T, H_w, v, p_w, p_w)
\end{aligned}
$$

Then, each head is updated to the next position based on the action probabilities. LEFT and RIGHT actions can be performed by the differentiable $roll$ function, which is a linear transformation $\mathbb{R}^L \longrightarrow \mathbb{R}^L$ mapping e_i to e_{i+1}.

$$(H_{LEFT}, H_{RIGHT}) = (roll^{-1}(H), roll(H))$$

The jump position H_{JUMP} is determined by reading the transpose of a key tape K with the unit query vector q_{jump}. The key tape is written in the same way as the tape T, but it stores the corresponding unit key vector derived using the weight matrix $W_k \in \mathbb{R}^{d \times d}$ and the unit vector normalization $\mu(.)$.

$$K' = WR(K, H_w, \mu(W_k x), p_w, p_w)$$
$$H_{JUMP} = RD(K'^{\top}, q_{jump}, 1)$$

One can understand the transpose of the key tape $K^{\top} = \sum_i e_i k_i^{\top}$ as a jump table. Reading $K^{\top}$ with a key vector k_i returns the corresponding position vector e_i if the keys are orthonormal. Finally, the next head position H' is updated as a weighted sum of the positions.

$$H' = p_{noop} \times H + p_{left} \times H_{LEFT} + p_{right} \times H_{RIGHT} + p_{jump} \times H_{JUMP}$$

While all of the computations are technically done with a fixed tape length L, none of the trainable parameters depend on the value of L. That is, a NAM-TM trained on certain length L can be applied to any tape length L' without any modification nor re-training. Theoretically, it can be extended to infinite-dimensional Hilbert spaces.

There are multiple strengths in NAM-TM design compared to the memory structures of NTM and DNC. First, unlike NTM and DNC, NAM-TM's building blocks are simple and computationally efficient. In the previous sections, the efficiency of NAM read/write primitives are already shown. Additionally, contemporary deep learning frameworks have efficient low-overhead implementations of the $roll$ transformation. Second, NAM-TM's addressing mechanism is based on the query-key-value attention mechanism of NAM, which is more powerful than the content-based attention mechanism used in NTM and DNC. Finally, NAM-TM's design is flexible in that it is easy to add/remove transition rules of the read/write heads. For example, the JUMP transition rule is optional in that a Turing machine only requires LEFT and RIGHT transitions in theory. One can also add another transition rule for the head positions if the rule can be implemented with differentiable functions.

5.3 Compositional Generalization Tasks

We test the computational powers of LSAM and NAM-TM using three types of algorithmic tasks in compositional generalization setups. First, number sequence prediction (Nam et al., 2019) task (NSP) is a suite of synthetic problems to predict the following digits of the numerical sequences. It can test the compositional generalization capability by testing/validating the models with the longer decimal numbers that are never observed during the training stage. In this setup, many models often suffer from drastic fall of test/validation accuracy, due to lack of inductive bias (Kim et al., 2021). We use two representative sequences from NSP: Fibonacci (Fib) and Palindrome (Palin). The two tasks require the generalization of digital addition and sequence reversal rules, respectively.

Next is a sequence reduction (Reduce) task with a simple rule: given the sequence of digits, the target is the reduced sequence by skipping the zeros. This is a task that can be easily solved by a proper Turing machine. We also use a similar compositional generalization setup by testing/validating using the longer target sequences that are longer than any target sequences in the training dataset.

For NSP and Reduce tasks, we use training datasets with $d = 1 \dots 10$ decimal digit sequences in little-endian order. Then we validate/test the models with two validation sets (ID, OD-easy) and a test set (OD-hard) for each task. An in-distribution (ID) validation set consists of $d = 5 \dots 10$-digit sequences, and an out-of-distribution validation set (OD-easy) consists of $d = 11 \dots 13$-digit sequences. The harder out-of-distribution test set (OD-hard) consists of $d = 14 \dots 16$-digit sequences, challenging the models with generalization to longer contexts. A training dataset has 25600 samples, and each validation/test set has 2048 samples.

The last task is SCAN (Lake & Baroni, 2018) task which has familiarized the concept of compositional zero-shot generalization. SCAN consists of simplified natural language input sequences and the corresponding output action sequences. Among many data split methods, we use train/test split by length [2] to be consistent with the other tasks. Since SCAN only has a limited number of sequences, we only have two datasets of train and test.

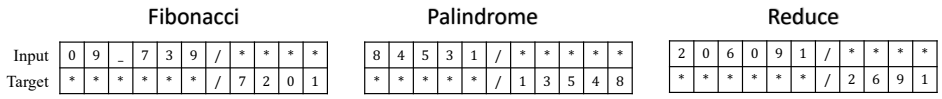

Figure 1: Input and output sequence examples of the tasks. The Fibonacci sequence is shown in the little-endian order.

We format the problems as masked sequence completion problems as shown in Figure 1. An input sequence consists of input tokens followed by masks, and an output sequence consists of target tokens following the masks. Since the input and output sequences are 1:1 matched, we can compare with baseline models without sequence-to-sequence structures like DNC. We choose four baseline models to compare: a bidirectional Transformer encoder (TF) (Devlin et al., 2018), an 2-layer LSTM model with attention (Bahdanau et al., 2014), an Universal Transformer (Dehghani et al., 2018), and a differentiable neural computer (DNC). The TF model follows the architecture of BERT$_{medium}$ and the hyperparameters of the other models are adjusted to have similar parameter counts. Our LSAM and NAM-TM networks have two LSAM layers and two NAM-TM layers respectively, whose hyperparameters are also adjusted to have similar parameter counts. As an ablation study, we also evaluate NAM-TM design without the JUMP transition (No Jmp). All experiments are run with PyTorch 1.10 on the system with Ubuntu 20.04 and RTX 3080. Each experiment takes less than five hours to run 200 training epochs. The implementation details and hypereparameters can be found at the source code [3]

5.4 Evaluation Result

Table 2: Sequence accuracy (%) comparison on the compositional generalization tasks.

	Model	TF	LSTM	UT	DNC	LSAM	NAM-TM	No Jmp
	Parameters	28.7M	31.5M	26.0M	40.2M	25.8M	23.3M	22.2M
Palin	ID	100%	100%	100%	100%	100%	100%	100%
	OD-easy	65.6%	100%	99.2%	100%	100%	100%	100%
	OD-hard	19.0%	100%	5.2%	100%	100%	100%	100%
Fib	ID	98.9%	46.2%	99.1%	46.9%	100%	97.4%	40.4%
	OD-easy	0.4%	8.3%	21.3%	1.8%	39.9%	**89.7%**	19.8%
	OD-hard	0.0%	0.0%	0.1%	0.0%	2.9%	**71.5%**	1.1%
Reduce	ID	99.6%	99.5%	98.7%	100%	99.9%	100%	100%
	OD-easy	0.2%	91.5%	15.7%	95.7%	96.8%	**100%**	**100%**
	OD-hard	0.0%	63.6%	0.0%	60.1%	77.6%	**100%**	**100%**
SCAN	Train	99.8%	99.9%	99.6%	99.3%	99.5%	99.6%	99.9%
	Test	0.0	14.5%	4.5%	8.6%	**14.9%**	11.2%	9.6%

Table 2 shows the sequence accuracy comparison of the models in compositional generalization tasks. We report the *sequence accuracies* of the models, where a wrong prediction of one token is counted as a failed prediction of the entire sequence. To avoid cherry-picking, the epochs of the best OD-easy validation accuracies are presented for the evaluation results. However, we present the results of the best test accuracy for SCAN tasks because there are only two available datasets (Train, Test).

[2] github.com/brendenlake/SCAN

[3] Included in supplementary materials. Will be open-sourced.

Although LSAM's architecture is not specifically designed for those algorithmic tasks, LSAM performs consistently better than the other baselines. Surprisingly, it performs better than DNC, which is a MANN model designed for such algorithmic problems. This is not a matter of model capacity because LSAM has a slightly smaller parameter count. The results imply that the computational power of NAM memory architecture is superior to that of DNC's memory architecture.

As expected, NAM-TM performs significantly better in algorithmic tasks (Palin, Fib, and Reduce) possibly due to its specialized architecture. Especially, it finds easier to solve OD-hard problems, whereas the other models experience steep performance decline. A potential explanation is that the action-based positional transitions provide robustness in long-context cases. However, NAM-TM is not the best-performing model for the SCAN tasks.

NAM-TM remains effective at palindrome and reduction tasks even without the JUMP transition (No Jmp). Hence, LEFT and RIGHT transitions seem to be enough for emulating simple Turing machines, but more complex transition rules can augment the computational power of NAM-TM. This suggests extending NAM-TM to variety of tasks by augmenting specialized transition actions.

6 Further Applications

Since NAM is a simple and flexible basic block for memory architecture, there are infinite number of applications. Hereby we suggest a few possible extensions of NAM. While we do not evaluate the ideas in this work, we leave them as possible future work.

Hierarchical data modeling Tensor product, the key operation behind NAM, is not limited to two-dimensional outer product. Any dimensional tensors T_i can construct a memory tensor M by performing *sum of tensor products* $M = \sum_i T_i \otimes k_i$, with the unit key vectors k_i. For example, a document tensor D can constructed by sum of tensor products of sentence-level keys and sentence matrices S_i, each of which is also a sum of outer products of word embeddings and word-level keys. *Nested attention* to such a hierarchical memory can be conducted by two *inner products* with a sentence-level unit query vector q_s and a word-level unit query vector q_w.

Efficient edge inference While Transformers are very successful in many NLP tasks, deploying such models for edge inference has been a challenging task since Transformer's computation and memory cost per each time step varies with the sequence length. However, NAM's cost does not depend the sequence length at all. Also, the outer product operation is more compute intensive than the dot products, making NAM more friendly to high-throughput NPUs. Hence, NAM can be an efficient Transformer alternative for edge inference.

Few-shot learning Given a input-output vector pair (x, y), it is possible to conduct *one-shot learning* of a weight matrix W by $WR(W, y/|x|, x/|x|, 1, 1)$. This is because Wx is *guaranteed* to return y by Theorem 2. Therefore, we can implement one-shot or few-shot learning as *memorization* to NAM. This aligns well with the human behavior since we conduct few-shot learning by memorizing, not by repetitive training.

7 Conclusion

We proposed a redesign of attention mechanism to construct a differentiable memory for neural networks, namely neural attention memory (NAM). Following the same query-key-value structure of scaled dot-product attention, NAM first writes the memory matrix by adding outer products of key-value pairs. Then we can read it by multiplying the matrix with a unit query vector. The first strength of NAM is that its computational complexity does not rely on the sequence length. The long-range arena evaluations showed that NAM based attention, namely normalized outer-product attention, is an efficient and effective alternative for scaled dot-product attention. Next, NAM can be a powerful basis for constructing MANN models. We designed two NAM-based MANNs: LSAM for generic sequential tasks and NAM-TM for algorithmic tasks. In compositional generalization tasks, both outperformed other baselines such as Universal Transformer and DNC, indicating that NAM is a more powerful mechanism for implementing memory in DNNs. Finally, it opens up further research possibilities in the fields of hierarchical data modeling, edge inference, and few-shot learning.

REFERENCES

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

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

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

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.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626): 471–476, 2016.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. *Advances in neural information processing systems*, 28, 2015.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf`.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02): 107–116, 1998.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Heikki Hyötyniemi. Turing machines are recurrent neural networks. *Proceedings of step*, 96, 1996.

Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. *Advances in neural information processing systems*, 28, 2015.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pp. 5156–5165. PMLR, 2020.

Segwang Kim, Hyoungwook Nam, Joonyoung Kim, and Kyomin Jung. Neural sequence-to-grid module for learning symbolic rules. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 8163–8171, 2021.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*, pp. 2873–2882. PMLR, 2018.

Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. Visualbert: A simple and performant baseline for vision and language, 2019. URL https://arxiv.org/abs/1908.03557.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

Hyoungwook Nam, Segwang Kim, and Kyomin Jung. Number sequence prediction problems for evaluating computational powers of neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4626–4633, 2019.

Thierry Tambe, Coleman Hooper, Lillian Pentecost, Tianyu Jia, En-Yu Yang, Marco Donato, Victor Sanh, Paul N. Whatmough, Alexander M. Rush, David Brooks, and Gu-Yeon Wei. Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference, 2020. URL https://arxiv.org/abs/2011.14203.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.